Fachhochschule Kiel

Fachbereich Informatik und Elektrotechnik

Bachelorthesis

im Studiengang Informationstechnologie und Internet

Thema: Linuxbasiertes Mehrkanal-Audiosystem mit niedriger Latenz

eingereicht von: Henrik Langer
eingereicht am: 21.12.2015

Erstprüfer: Prof. Dr. Robert Manzke
Zweitprüfer: Prof. Dr.-Ing. habil. Kißig

Abstract

In this Thesis, the Linux driver architecture for a novel multichannel, low-latency pulse Pulse-Code-Modulation (PCM) sound system has been developed and evaluated.
The hardware of the system is based on the AD1938 audio codec by Analog Devices and has been developed in a separate effort to this thesis, apart from audio codec selection considerations.
The system offers four stereo audio outputs and two stereo inputs, operating with 24bits dynamic range and either 48, 96, or 192kHz sampling rate.
The development of the driver architecture includes ALSA (Advanced Linux Sound Architecture) device drivers that use the ASoC (ALSA System on Chip) layer, sound server settings, device tree overlays and capes, register maps and real-time patches to the kernel. A detailed overview and introduction to these topics is given in the Thesis.
The overall system has been evaluated with regard to technical sound parameters and latency to gauge its usefulness as a powerful new platform for audio development projects such as embedded digital effect processors for musicians.
To demonstrate the possibilities of the audio system, a multichannel digital delay effect has been developed, which uses all available audio channels and is based on the open source C++ flow-based programming library DSPatch. It is available to the public on an adapted Debian distribution image.

Danksagung

Diese Bachelorarbeit konnte nur durch die aufwendige zur Verfügungsstellung der Hardware, insbesondere des Soundkartenprototyps und der Revision davon, durch Prof. Dr. Robert Manzke durchgeführt werden.
Weiterhin wurden technische Geräte, wie Oszilloskop, und Räumlichkeiten in der Fachhochschule Kiel bereitgestellt. Dafür ein großes Dankeschön.
Auch ein Dankeschön an RF William Hollender, den Entwickler des SuperAudioBoards[1], welcher durch seine Informationen zum Design der Hardware und Evaluationsmethoden sehr hilfreich war.

[1] `https://hackaday.io/project/5912-teensy-super-audio-board`

Inhaltsverzeichnis

1 Einleitung

1.1 Überblick

Diese Bachelorarbeit wurde im Rahmen meines Studiengangs Informationstechnologie und Internet in Zusammenarbeit mit der Fachhochschule Kiel erstellt. Die genutzte und zu testende Hardware, insbesondere die ersten beiden Revisionen der AD1938 AudioCard und das SuperAudioBoard, wurden von Prof. Dr. Robert Manzke, aus dem Institut für Angewandte Informatik, entwickelt bzw. zur Verfügung gestellt. Da die Bachelorarbeit anfangs auf Basis des Raspberry Pi 2 Model B durchgeführt werden sollte und zu diesem Zeitpunkt noch keine Schaltpläne offen gelegt waren, gab es vor Beginn Zweifel, ob ein Mehrkanal-Audiosystem auf dessen Basis tatsächlich realisierbar ist. Zufälligerweise ist zu dieser Zeit das Open Source Projekt SuperAudioBoard[1] von RF William Hollender veröffentlicht worden, welches einen Stereo Ein- und Ausgang besitzt und einen Cirrus Logic CS4272 Audiocodec nutzt, welcher anfangs auch für eine neu zu entwickelnde Platine in Betracht kam. Durch Nachbau der Platine von Prof. Dr. Robert Manzke entstand eine gute Grundlage zur Evaluation der neu entwickelten Platine. Die Hardwareentwicklung der neuen Mehrkanal-Platine wurde komplett von Prof. Dr. Robert Manzke durchgeführt.

1.2 Gegebenheiten und Motivation

Aufgrund des enormen Anstiegs der Rechenperformance und des geringen Preises bieten sich heutzutage eingebettete Systeme in sehr vielen Anwendungsbereichen an. Speziell der Raspberry Pi hat sich im Hobby- und Lehrbereich schnell durchgesetzt. Durch die Offenheit der Hard- und Software entstanden so diverse Projekte wie Media-Center[1], Supercomputer[2], Wetterballons[3] und vieles[4] mehr. Auch im Audiobereich sind Projekte, wie z.B. SonicPi[5] entstanden, jedoch war der standardmäßig integrierte, analoge PWM (Pulsweitenmodulation) Audioausgang qualitativ recht schlecht[6]. Allerdings bietet der Raspberry Pi ein integriertes, digitales PCM-Audiointerface an, wodurch Projekte, wie HiFiBerry[7] entstanden sind, deren Zielsetzung eine professionelle Klangqualität durch eine extra Platine mit einem Audiocodec war. Allerdings existierte zu diesem Zeitpunkt noch kein Mehrkanal-Audiosystem auf Basis des Raspberry Pi 2 Model B, woraus sich die Themenidee dieser Bachelorarbeit ergab. Um das Mehrkanal-Audiosystem für möglichst viele Anwendungsgebiete, wie z.B. Musikeffekt-Geräte, nutzen zu können, sollte es eine gute technische Klangqualität und eine geringe Latenz, im besten Fall unter 10 Millisekunden, aufweisen. Zu Evaluierungszwecken wurde während der Bachelorarbeit das SuperAudioBoard als Referenz zu Grunde gelegt.

[1] http://www.raspbmc.com/index.html
[2] https://www.southampton.ac.uk/mediacentre/features/raspberry_pi_supercomputer.shtml
[3] https://www.raspberrypi.org/blog/pi-in-the-sky-2/
[4] https://hackaday.io/projects/tag/raspberry%20pi
[5] http://sonic-pi.net/
[6] http://www.crazy-audio.com/2014/07/sound-quality-of-the-raspberry-pi-b/
[7] https://www.hifiberry.com/

1.3 Hypothese

Aufgrund der hohen Verfügbarkeit von Mehrkanal-Audiocodecs und zunehmend großer Auswahl von Einplatinencomputern, welche eine digitale Audioschnittstelle besitzen, sollte es möglich sein, ein hochqualitatives Mehrkanal-Audiosystem mit niedriger Latenz preiswert zu realisieren. Durch die so neu entstandene Anwendungsvielfalt und Offenheit könnten Einplatinencomputer im Zusammenhang mit Linux eine gute und transparente Plattform für neue Audiotechnologien bieten.

1.4 Ablauf

Diese Bachelorarbeit beginnt mit einer kurzen Einleitung in die notwendigen technischen Grundlagen. Anschließend wird detaillierter auf die Entwicklung der Audiotreiber und die damit entstandenen Probleme eingegangen. Danach folgt die Evaluierung, die insbesondere die Klangqualität und die Latenz in den Fokus nimmt. Abschließend folgt das Fazit und der Ausblick mit einem Demonstrationsprojekt, wodurch die Möglichkeiten des neu entwickelten Mehrkanal-Audiosystems dargestellt werden sollen.

2 Grundlagen

2.1 Puls-Code-Modulation (PCM)

Durch die Puls-Code-Modulation wird ein zeit- und wertkontinuierliches, analoges Signal in ein zeit- und wertdiskretes, digitales Signal umgewandelt. Jeder Wert wird in einem regelmäßigem, durch den Takt vorgegebenen, Zeitintervall abgetastet (zeitdiskretes Signal). Durch Wandlung und Quantisierung des analogen Wertes, wird der digitale Wert ermittelt. Die maximale Abweichung von dem analogen Wert ist immer abhängig von der Samplingtiefe. Im professionellen Audiobereich sind dies meistens 24 Bit. Daraus ergibt sich bei der Quantisierung im Intervall $[0, 1] = \{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$ ein maximaler Fehler von $1/(\frac{2^{24}}{2}) \approx 1,192 \cdot 10^{-7}$.

2.2 Inter-IC Sound (I^2S)

Die Inter-IC Sound Schnittstelle wurde von Philips zur Übertragung von seriellen, digitalen Audiodaten entwickelt. Laut der I^2S-Protokollspezifikation[2] ist die Anzahl der zu übertragenden Audiokanäle auf 2 (Stereo) festgelegt. Da die I^2S-Schnittstelle eine synchrone Schnittstelle ist, werden Takt und Signal über eigene Datenleitungen übertragen. Weiterhin handelt es sich um eine Simplex-Schnittstelle, somit können Daten nur in eine Richtung übertragen werden. Hieraus ergeben sich die folgenden drei Signalleitungen (für eine Richtung):

Name	Funktion
Serial-Clock (SCK)	Taktet die komplette Datenübertragung. Die Taktfrequenz ergibt sich aus dem Produkt der Abtastrate, Samplingtiefe und der Anzahl der Kanäle. Beispiel: 48 kHz Abtastrate, 24 Bit Samplingtiefe, 2 Audiokanäle Dadurch ergibt sich eine Taktfrequenz von $48kHz \cdot 24Bit \cdot 2 = 2,304MHz$.
Word-Select (WS)	Taktsignal, um aktiven Audiokanal zu signalisieren. Die Taktfrequenz entspricht daher immer genau der Abtastrate der zu übertragenen Audiodaten.
Serial-Data (SD)	Audiodaten im PCM-Format. Die Audiokanäle werden alternierend im Datenstrom kodiert.

Tabelle 2.1: I²S Signalleitungen und deren Funktion

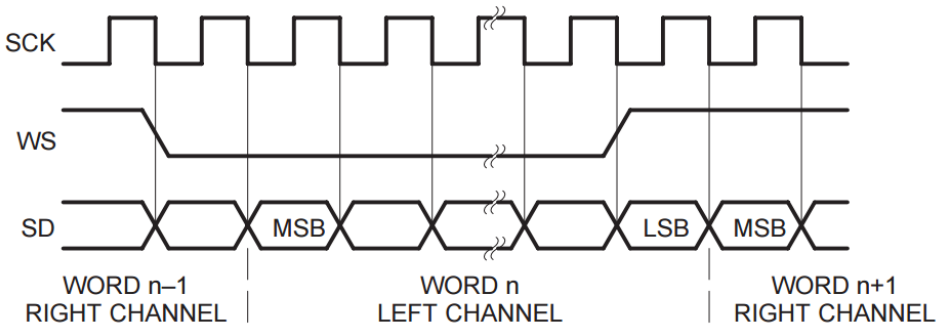

Abbildung 2.2: I²S Basic Interface Timing[2, S. 1 Figure 1.]

Detaillierte Informationen können der I²S-Busspezifikation[2] entnommen werden.

2.3 Time Division Multiplexing (TDM)

Im Gegensatz zu I²S, nutzt TDM das Word-Select Taktsignal nicht, um zwischen linkem und rechtem Kanal zu unterscheiden, sondern um den Beginn einer Gruppe von Audiokanälen zu signalisieren. Voraussetzung hierfür ist, dass sich Sender und Empfänger

auf die Anzahl der zu übertragenden Audiokanäle, deren Samplingtiefe bzw. der TDM-Slotbreite und der Abtastrate geeinigt haben. Haben Sender und Empfänger beispielsweise eine Abtastrate von 48 kHz, eine TDM-Slotbreite von 32 Bit und 8 Audiokanäle festgelegt, signalisiert der Beginn einer Periode des Word-Select-Signals den Beginn des ersten Audiokanals. Nach 32 übertragenen Bits folgt der nächste Audiokanal. Da I^2S nach dem gleichen Prinzip arbeitet, jedoch maximal nur 2 Kanäle zulässt, kann es als eine Untermenge von TDM betrachtet werden. Eine genaue Protokollspezifikation ist im AM335X Datenblatt[3] verfügbar.

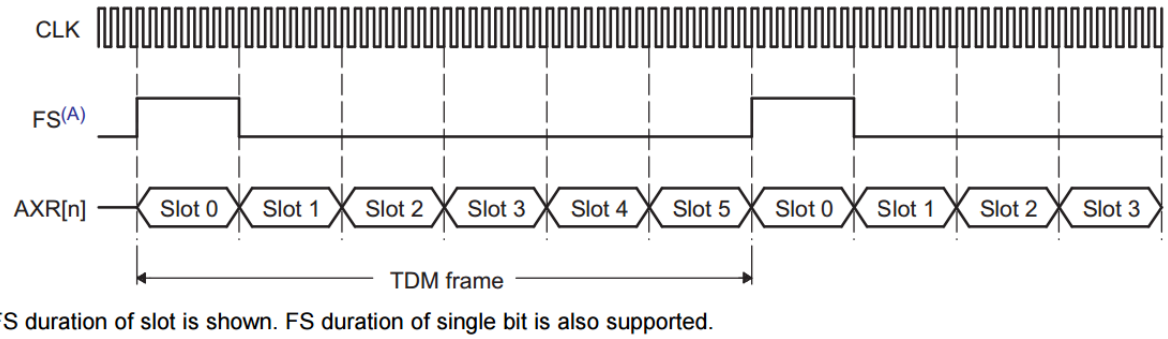

Abbildung 2.3: TDM Format–6 Channel TDM Example[3, S. 4550 Figure 22-8.]

Durch die höhere Anzahl der Audiokanäle müssen beim TDM-Protokoll die hohen Taktfrequenzen, welche für die Bitclock benötigt werden, Beachtung finden. Sollen beispielsweise 8 TDM-Slots mit jeweils einer Breite von 32 Bit und einer Abtastrate von 48 kHz übertragen werden, wird eine minimale Taktfrequenz von
$f_{CLK} = 8 \cdot 32 \cdot 48kHz = 12,288MHz$ benötigt. Entsprechend verdoppelt sich die benötigte Taktfrequenz bei einer Abtastrate von 96 kHz.

2.4 Serial Peripheral Interface (SPI)

Das Serial Peripheral Interface[4] ist ein synchroner, vollduplexfähiger, serieller Datenbus nach dem Master-Slave-Prinzip. Aufgrund der Flexibilität des Übertragungsprotokolls wird der SPI-Bus in diversen Bereichen eingesetzt. An dem Bussystem darf nur ein Master existieren, der einen Slave über den Chip-Select-Not-Eingang (CSN) aktiviert, um miteinander zu kommunizieren. Theoretisch können so unendlich viele Slaves, abhängig von der Anzahl der CSN-Ausgänge des Busmasters, am Bussystem angeschlossen wer-

den. Ein Nachteil ist, dass SPI die 4 Signalleitungen SCK (Serial-Clock), SDI (Serial-Data-In), SDO (Serial-Data-Out) und CSN (Chip-Select-Not) benötigt.

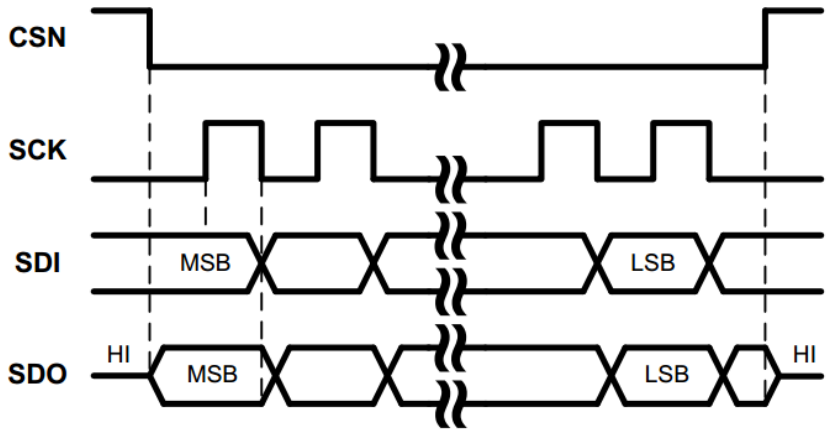

Abbildung 2.4: ST SPI Signal Beschreibung[4, S. 7 Figure 2.]

Im Datenblatt des AD1938 Audiocodecs[5] und des Raspberry Pi Einplatinencomputers[6] werden die Signale der SPI-Schnittstelle auch als CCLK bzw. SCLK (Serial-Clock), CIN bzw. MISO (Master Input, Slave Output), COUT bzw. MOSI (Master Output, Slave Input) und CLATCH bzw. CE (Chip-Enable) bezeichnet.

2.5 Inter-Integrated Circuit (I^2C)

I^2C[7] ist ähnlich wie SPI, ein serieller, digitaler Datenbus nach dem Master-Slave-Prinzip. Im Gegensatz zu SPI, ist bei I^2C jedoch das Übertragungsprotokoll fest definiert. Aufgrund der darin vorgeschriebenen Start-, Stopp-, und Adress-Bits, ist zwar die Teilnehmeranzahl am Datenbus begrenzt, jedoch werden nur die beiden Signalleitungen Serial-Clock-Line (SCL) und Serial-Data-Line (SDA) benötigt. Zum Verständnis folgt eine Abbildung eines kompletten Datentransfers aus der I^2C-Bus Spezifikation:

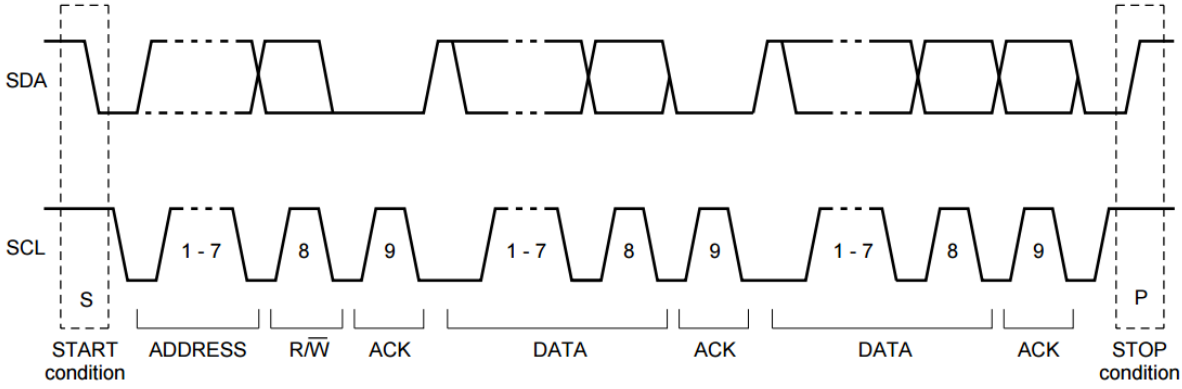

Abbildung 2.5: Beispiel eines kompletten I²C Datentransfers[7, S. 13 Figure 9.]

In Abbildung 2.5 ist die festgelegte Definition der einzelnen Bits gut zu erkennen. Durch die 7 festgelegten Adress-Bits können maximal $2^7 = 128$ Teilnehmer am Bus angeschlossen werden. Weiterhin legt das I²C-Protokoll nach jedem übertragenen Byte ein Bestätigungsbit des Empfängers fest. Ist das Lese-/Schreibbit gesetzt, antwortet der Slave direkt nach dem Bestätigungsbit mit einem Byte Daten. Andernfalls antwortet der Master nach dem Bestätigungsbit mit einem Byte Daten.

2.6 Hardware- und Softwareplattformen

2.6.1 Teensy 3.1

Der Teensy 3.1[8] ist ein Arduino kompatibler Microcontroller, welcher von PJRC entwickelt wurde. Er basiert auf dem verhältnismäßig leistungsstarken MK20DX256VLH7[9] Cortex-M4 SoC. Da der Teensy 3.1 ein Arduino kompatibler Microcontroller ist, können die komplette Arduino Compiler-Toolchain und die Arduino-Bibliotheken genutzt werden. Weiterhin verfügt der Teensy 3.1 über eine I²S-, SPI- und I²C-Schnittstelle, weshalb er sich besonders für erste Funktionstests der Audiocodecs eignet.
Ein Betriebssystem ist für die Hardware nicht geeignet, daher wird die Entwicklung direkt auf Hardwareebene in der Programmiersprache C durchgeführt. Allerdings entfällt durch die Vielfältigkeit und den hohen Abstraktionsgrad der Arduino-Bibliotheken eine zeitaufwendige Einarbeitung in die Hardware. Daher ist der Teensy 3.1 insbesondere für Prototyping bestens geeignet ist.

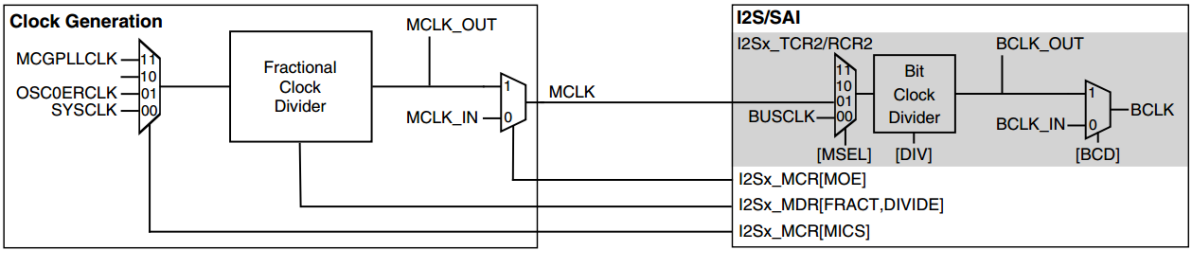

Abbildung 2.6: I²S Taktgenerierung des Teensy 3.1[9, S. 166 Figure 5-8.]

Abbildung 2.6 stellt die Möglichkeiten der Taktgenerierung des Teensy 3.1 dar. Als Quelle für die Masterclock können intern generierte Takte oder ein externes Taktsignal (*MCLK_IN*) genutzt werden. Der Takt der *MCGPLLCLK* wird, wie auch die Takte des Systembus, vom MCG-Modul, welches wiederum ein internen oder externen Quarz als Quelle nutzt, über eine Phasenregelschleife (PLL) abgeleitet. Der Takt der *OSC0ERCLK* wird im Gegensatz dazu, direkt, ohne Nutzung des MCG-Moduls, von einem internen oder externen Quarz abgeleitet. Der Takt der *SYSCLK*, wird ähnlich wie der Takt der *MCGPLLCLK*, auch vom MSG-Modul abgeleitet, jedoch ohne Phasenregelschleife. Die *SYSCLK* taktet auch die CPU[9, S. 155 ff. Kapitel 5.]. Anschließend kann über einen Teiler die gewünschte Taktfrequenz der Masterclock konfiguriert werden. Die Bitclock und Frameclock der I²S-Schnittstelle können asynchron für Senden und Empfangen generiert werden. Dadurch können Audiodaten in verschiedenen Abtastraten und Formaten aufgenommen bzw. wiedergegeben werden[9, S. 135 ff. Kapitel 3.9.6.2.].

2.6.2 Raspberry Pi 2 Model B

Der Raspberry Pi 2 Model B[10] ist ein Einplatinencomputer basierend auf einem BCM2836 SoC von Broadcom, welcher von der Raspberry Pi Foundation entwickelt wurde. Im Gegensatz zu den Vorgängermodellen besitzt der Raspberry Pi 2 Model B eine 900MHz Quad-Core ARM Cortex-A7 CPU, wodurch sich die Rechenleistung, insbesondere bei mehreren Prozessen bzw. Threads, vervielfacht hat[11]. Die Peripherie hat sich im Vergleich zum Vorgänger Raspberry Pi Model B+ nicht geändert und ist somit komplett kompatibel. Der Raspberry Pi 2 Model B besitzt ein integriertes, digitales Audio Interface, welches I²S unterstützt. Leider ist die Schnittstellenspezifikation wenig umfangreich, wodurch die meisten Möglichkeiten und Limits der Hardware nur durch Recherche der einzelnen Register ermittelt werden können.

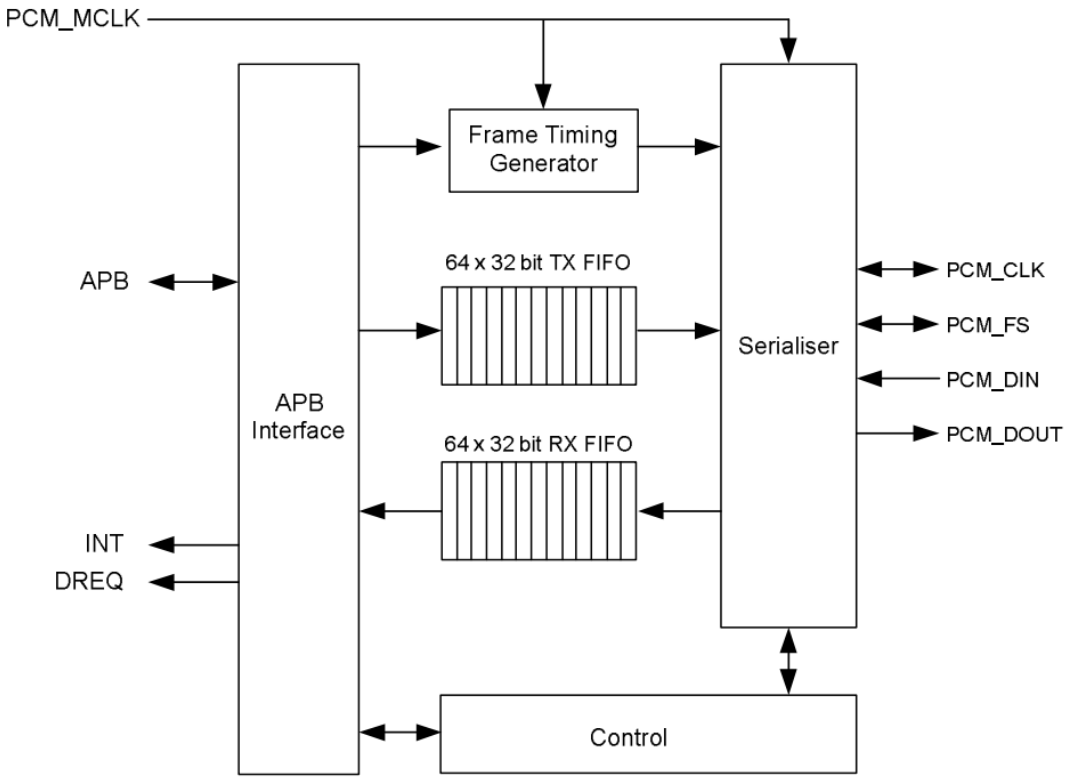

Abbildung 2.7: Raspberry Pi PCM Audio Interface Blockdiagramm[6, S. 120 Figure 8.2.]

Aus Abbildung 2.7 kann man entnehmen, dass das digitale Audiointerface sowohl als Master wie auch als Slave konfiguriert werden kann (die Bitclock *PCM_CLK* und die Frameclock *PCM_FS* sind bidirektional gekennzeichnet). Die Masterclock *PCM_MCLK* kann weiterhin intern oder über ein externes Taktsignal generiert werden.
Für den Raspberry Pi 2 stehen diverse Betriebssysteme zur Verfügung[1]. Das von der Raspberry Pi Foundation selbst entwickelte Raspbian[2] basiert auf der Debian Wheezy Distribution mit einem 3.18 Linux-Kernel, welches auch für diese Bachelorarbeit verwendet wurde. Viele Hersteller von Einplatinencomputern bevorzugen die Debian Linux-Distribution, da sie durch eine lange Testphase als besonders stabil eingestuft wird[3]. Allerdings hat dies den Nachteil, dass die Softwarepakete aus den offiziellen Paketquellen häufig veraltet sind, was sich ebenfalls in dieser Bachelorarbeit zeigte (siehe 6.2).

[1] `https://www.raspberrypi.org/downloads/`
[2] `https://www.raspbian.org/`
[3] `https://wiki.debian.org/WhyDebian`

2.6.3 BeagleBone Green

Der BeagleBone Green[12] basiert auf einer Sitara AM3358 CPU von Texas Instruments und ist komplett kompatibel zum BeagleBone Black, hat jedoch statt einem HDMI-Ausgang eine Schnittstelle für Grove Sensoren[4]. Die Rechenperformance ist durch den Einkernprozessor allerdings um ein Vielfaches geringer als vom Raspberry Pi 2 Model B und vergleichbar mit dem Vorgänger Raspberry Pi Model B+[13]. Ein Vorteil ist die umfangreiche Peripherie, welche über externe Pins verfügbar ist. Beim Raspberry Pi 2 Model B stehen beispielsweise nur 40 externe Pins zur Verfügung[5]. Beim BeagleBone Green hingegen 2 mal 46 Pins, wobei (bis auf Pins, welche die Stromversorgung betreffen) sogar jeder Pin durch Multiplexing bis zu 8 verschiedene Funktionen haben kann[14, S. 84 ff.].
Die Auswahl des BeagleBone Green als Alternativplattform, geschah aufgrund des in der AM3358 CPU integriertem *Multichannel Audio Serial Ports* (McASP). Dieser hat weitaus mehr Möglichkeiten als herkömmliche I^2S- bzw. PCM-Schnittstellen und unterstützt hardwaremäßig das TDM-Protokoll[3, S. 4550 Kapitel 22.3.3.1.].
Ist der McASP als Master konfiguriert, können die Bit- und Frameclocks intern vom Systemtakt abgeleitet oder von einem externen Taktsignal über die Masterclocks *AHCLKX* bzw. *AHCLKR* generiert werden. Im Slave Modus werden diese wiederum extern getaktet[3, S. 4556 ff. Kapitel 22.3.5.], was der folgenden Abbildung zu entnehmen ist:

[4] `http://www.seeedstudio.com/depot/s/grove.html`
[5] `http://elinux.org/RPi_Low-level_peripherals`

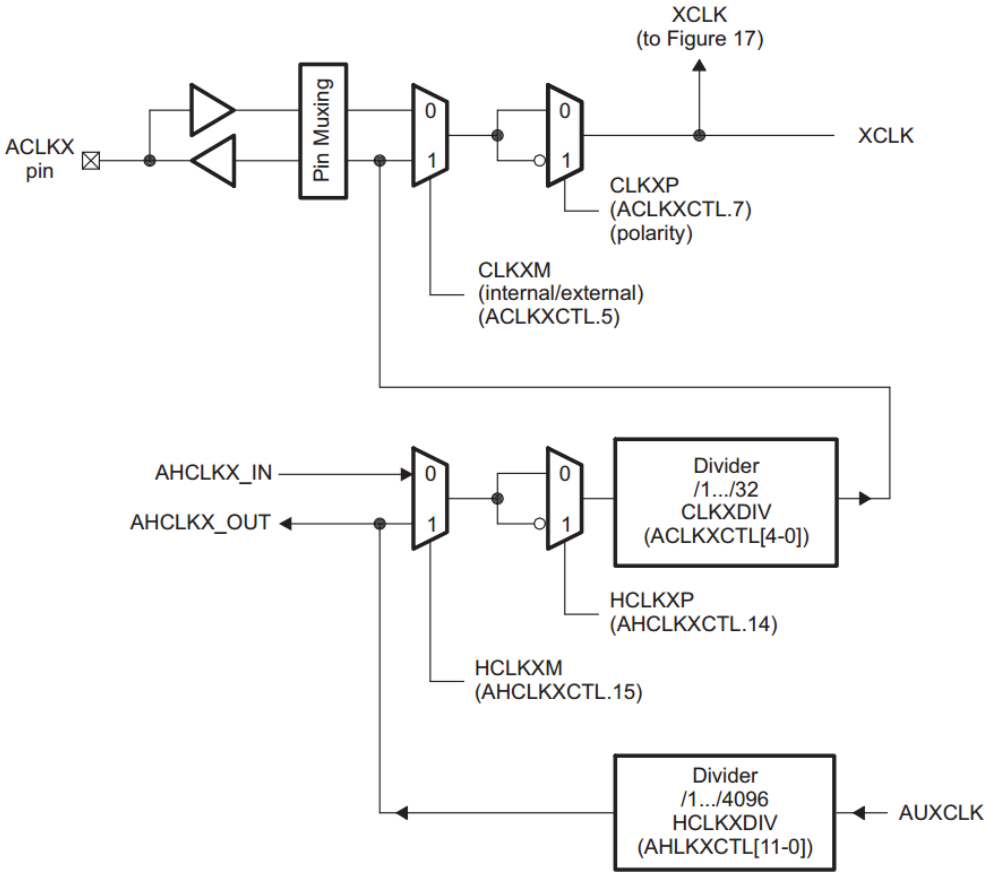

Abbildung 2.8: Blockdiagramm der Taktgenerierung von TI AM335X CPU[3, S. 4557 Figure 22-17.]

Ein weiterer Vorteil besteht darin, dass Texas Instruments einen eigenen Fork des Linux-Kernels 4.1 pflegt, wodurch die aktuelle Hardwareunterstützung sehr ausgereift ist. Der modifizierte Linux-Kernel wird aufgrund dessen auch für die BeagleBone Modelle, welche auf der AM335X CPU basieren, zur Verfügung gestellt[6].
Für die BeagleBone Reihe wird eine angepasste Debian Distribution empfohlen, welche allerdings zurzeit in der stabilen Version noch den relativ alten Linux-Kernel 3.8.13 nutzt. Dieser bietet noch keine ausgereifte Unterstützung für Device Trees (2.8.3), was eine aufwendigere Entwicklung der Gerätetreiber zur Folge hat[7]. Während der Bachelorarbeit wurde daher der noch von Texas Instruments als unstabil eingestufte Linux-Kernel 4.1 vom offiziellen BeagleBone Linux-Repository auf GitHub.com geforked und verwendet[15].

[6] `http://elinux.org/Beagleboard:BeagleBoneBlack_Debian`
[7] `http://elinux.org/BeagleBone_and_the_3.8_Kernel`

2.7 Audio Codecs

2.7.1 Recherche und Auswahl

Als gute Quelle zur Auswahl hat sich das Audiocodec Treiber-Verzeichnis[8] vom ALSA ASoC Layer erwiesen. Alle dort aufgelisteten Audiocodecs unterstützen bereits den ASoC Layer. Aufgrund des günstigen Preises und der verhältnismäßig guten Klangqualität, sind die beiden Audiocodecs CS4272 von Cirrus Logic[16] und AD1938 von Analog Devices[5] in die engere Auswahl für eine neue Platine gekommen. Der AD1938 Audiocodec verfügt laut Datenblatt zwar über eine geringere technische Klangqualität als der CS4272, bietet jedoch im Gegensatz dazu 2 Stereo-Eingänge und 4 Stereo-Ausgänge an (siehe 2.7.2 und 2.7.3). Zufälligerweise wurde zu diesem Zeitpunkt das SuperAudioBoard[1] von RF William Hollender veröffentlicht, welches auch einen CS4272 Audiocodec nutzt. Damit war eine gute Grundlage zur Evaluierung des CS4272 Audiocodecs gegeben. Da es bis zu diesem Zeitpunkt noch keine PCM Soundkarten mit mehr als 2 Kanälen für den Raspberry Pi 2 Model B und dessen Vorgängermodelle gab, wurde für eine eigens entwickelte Platine der AD1938 Audiocodec ausgewählt.

2.7.2 Cirrus Logic CS4272

Folgend eine Zusammenfassung einiger Kennwerte und Features aus dem CS4272 Datenblatt[16]:

- Allgemein
  - bis zu 192 kHz Abtastrate
  - SPI- und I^2C-Schnittstelle
  - On-Chip Oszillator (kein Quarz nötig)
  - interner digitaler Loopback
  - I^2S / Left-Justfied / Right-Justified

[8] `https://git.kernel.org/cgit/linux/kernel/git/stable/linux-stable.git/tree/sound/soc/codecs`

- 2 x 24 Bit Analog-Digital-Wandler
  - -100 dB THD+N (Total Harmonic Distortion + Noise)
  - 114 dB DNR (Dynamic Range)
  - differentieller Eingang
  - unabhängige Hochpassfilter für linken und rechten Kanal
  - automatisches Dithering für 16 Bit Daten
- 2 x 24 Bit Digital-Analog-Wandler
  - -100 dB THD+N
  - 114 dB DNR
  - differentieller Ausgang
  - logarithmische Lautstärkenregelung

2.7.3 Analog Devices AD1938

Folgend eine Zusammenfassung einiger Kennwerte und Features aus dem AD1938 Datenblatt[5]:

- Allgemein
  - bis zu 192 kHz Abtastrate
  - SPI-Schnittstelle
  - PLL generierte oder direkte Masterclock (Quarz nötig)
  - I^2S / Left-Justfied / Right-Justified / TDM
- 4 x 24 Bit Analog-Digital-Wandler
  - 107 dB DNR / SNR

  - -96 dB THD+N
  - differentieller Eingang
- 8 x 24 Bit Digital-Analog-Wandler
  - 108 dB DNR / SNR
  - -92 dB THD+N (2 aktive Kanäle)
  - -86 dB THD+N (8 aktive Kanäle)
  - Single-Ended Ausgang
  - logarithmische Lautstärkenregelung

2.8 GNU/Linux

2.8.1 Advanced Linux Sound Architecture (ALSA)

Die Advanced Linux Sound Architecture[17] bietet Audio und MIDI Funktionalität für Linux und ersetzt das mittlerweile veraltete Open Sound System (OSS). Zu den Zielen von ALSA zählt die automatische Konfiguration der Soundhardware und die Unterstützung von mehreren Soundkarten in einem System. Besondere Features sind unter anderem

- Support für diverse Audio Interfaces im Heim- und professionellen Bereich
- voll modularisierte Audio-Treiber
- SMP (Symmetrisches Multiprozessorsystem) und threadsicheres Design
- Userspace Bibliothek, zur Erleichterung der Anwendungsprogrammierung
- Support für alte OSS API und binärkompatibel zu den meisten OSS Programmen

ALSA besteht aus mehreren Schichten. Auf der oberen Schicht befindet sich der Soundserver, über den ALSA-Anwendungen mithilfe der ALSA Bibliothek mit Mixer, Se-

quencer und anderen hardwareabhängigen Schnittstellen auf der mittleren Ebene kommuniziert. Auf der unteren Ebene befindet sich der hardwarespezifische Treiber-Code. Folgendes Diagramm bietet eine gute Übersicht von ALSA:

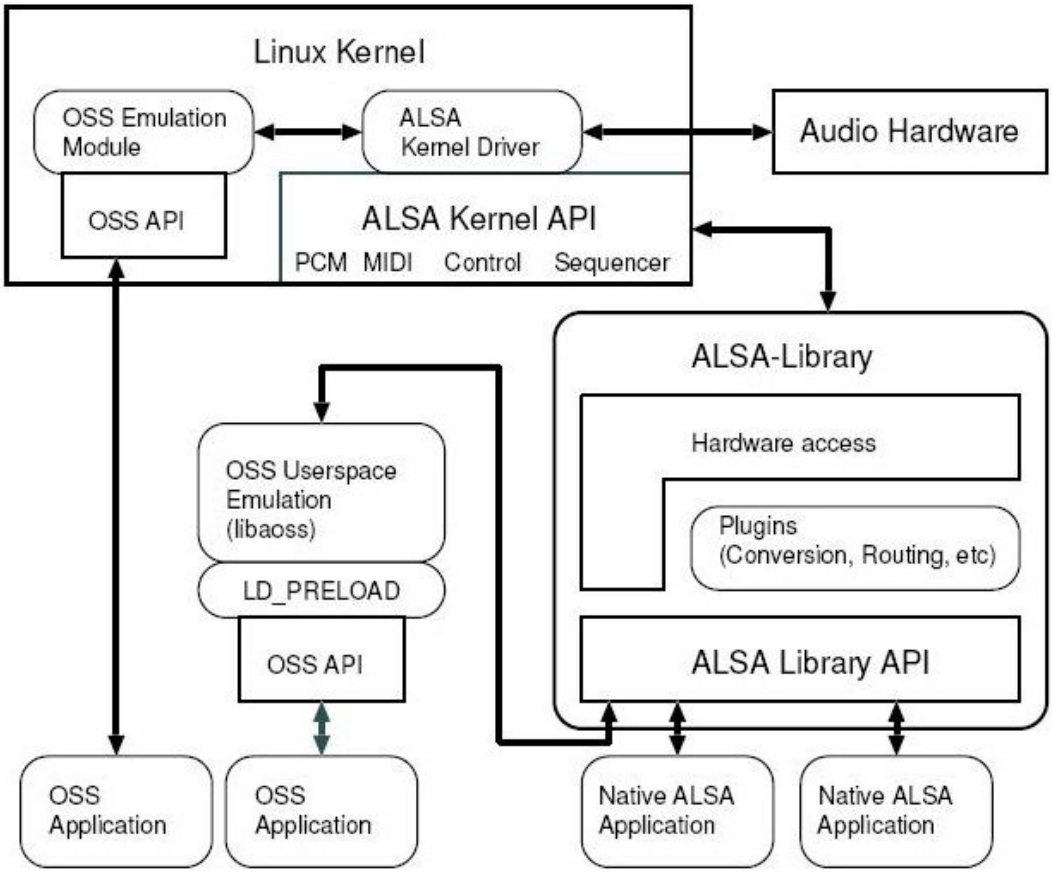

Abbildung 2.9: Blockdiagramm von ALSA System[18, Figure 1.]

Abbildung 2.9 stellt die Architektur von ALSA dar. Der Abstraktionsgrad steigt mit der nach unten gehenden Richtung. Dementsprechend sind die hardwarenahen Komponenten weiter oben dargestellt. Die ALSA-Treiber werden innerhalb des Kernel-Kontextes ausgeführt und implementieren verschiedene Funktionen, wie PCM, auf die über eine fest definierte API über die ALSA-Bibliothek im Userspace zugegriffen werden kann. Detaillierte Informationen über die Geschichte und Funktionen von ALSA können im Dokument „Sound Systems on Linux: From the Past To the Future“ von Takashi Iwai nachgelesen werden[18].
ALSA bietet weiterhin die Möglichkeit, die Puffergrößen und den Zeitpunkt, wann ein Interrupt ausgelöst wird, zu konfigurieren. Aufgrund dessen sollen in diesem Zusam-

menhang die Begriffe „Frame“ und „Period“ hier noch einmal klar definiert werden. Ein *Frame* repräsentiert ein analoges Audiosample, abhängig von dem PCM-Format und der Anzahl der Audiokanäle. Wird beispielsweise ein Audiosample mit 24 Bit Samplingtiefe und 2 Kanälen abgespielt, hat ein einzelner *Frame* eine Größe von $2 \cdot 24Bit = 48Bit = 6Byte$.

> Eine *Periode* repräsentiert die Anzahl der Frames innerhalb eines Hardwareinterrupts. Der Puffer ist ein Ringpuffer und ist im Regelfall doppelt so groß wie die Periodengröße[19].

Mittels des Ringpuffers wird bei der Wiedergabe von Audio innerhalb eines Interrupts zunächst der halbe Puffer mit Frames gefüllt. Beim nächsten Interrupt wird die nächste Hälfte gefüllt und die vorige weiter übertragen. Dies wiederholt sich solange, bis keine Audiodaten zum Übertragen mehr verfügbar sind. Sollte sich die Übertragung der Daten im Puffer so lange verzögern, dass der Puffer überläuft bzw. bei der Aufnahme der Puffer unterläuft, wird dies im Kontext von ALSA als „XRUN“ bezeichnet.

ALSA System on Chip (ASoC)

Da ALSA Treiber stark von der CPU Architektur und dem Audiocodec abhängig sind, entstand insbesondere bei eingebetteten Systemen das Problem, bei kleinen Änderungen der Hardware wiederholt einen komplett neuen Treiber entwickeln zu müssen. Hat sich ein Hersteller beispielsweise in einem neuen Produkt für einen anderen Audiocodec entschieden, musste der komplette ALSA Treiber neu entwickelt werden. Das gleiche Problem entsteht insbesondere bei einer Änderung der CPU Architektur bzw. der PCM-Schnittstelle der CPU. Aus diesem Grund entstand die Idee, eine zusätzliche Schicht, den sogenannten ASoC (ALSA System on Chip) Layer[20], einzuführen, der den CPU Plattform-Treiber von den Audiocodec-Treibern entkoppelt und damit die einzelnen Treiber besser wiederverwendbar macht. ALSA Treiber, die den ASoC Layer nutzen, werden in 3 Hauptkomponenten unterteilt, die in der folgenden Tabelle beschrieben sind:

Komponente	Beschreibung
Plattform-Treiber	Implementiert alle Funktionen, die das digitale Audio Interface der CPU betreffen. Da der Treiber wiederverwendbar sein soll, darf hier nichts implementiert werden, was einen spezifischen Audiocodec betrifft.
Codec-Treiber	Implementiert alle Funktionen, die den Audiocodec betreffen. Ist im Gegensatz zum Plattform-Treiber unabhängig von dem digitalen Audiointerface der CPU.
Maschinen-Treiber	Verbindet Plattform- und Codec-Treiber. Ist nicht wiederverwendbar, da er sowohl CPU-Plattform, wie auch Codec-Treiber spezifischen Quellcode enthält und somit von beiden Komponenten abhängig ist.

Tabelle 2.10: ASoC Komponenten

Eine sehr gute Übersicht bietet das ASoC-Architektur Diagramm der AM335X CPU von Texas Instruments, welche auch die Basis für den BeagleBone Black[14] und den BeagleBone Green[21] ist:

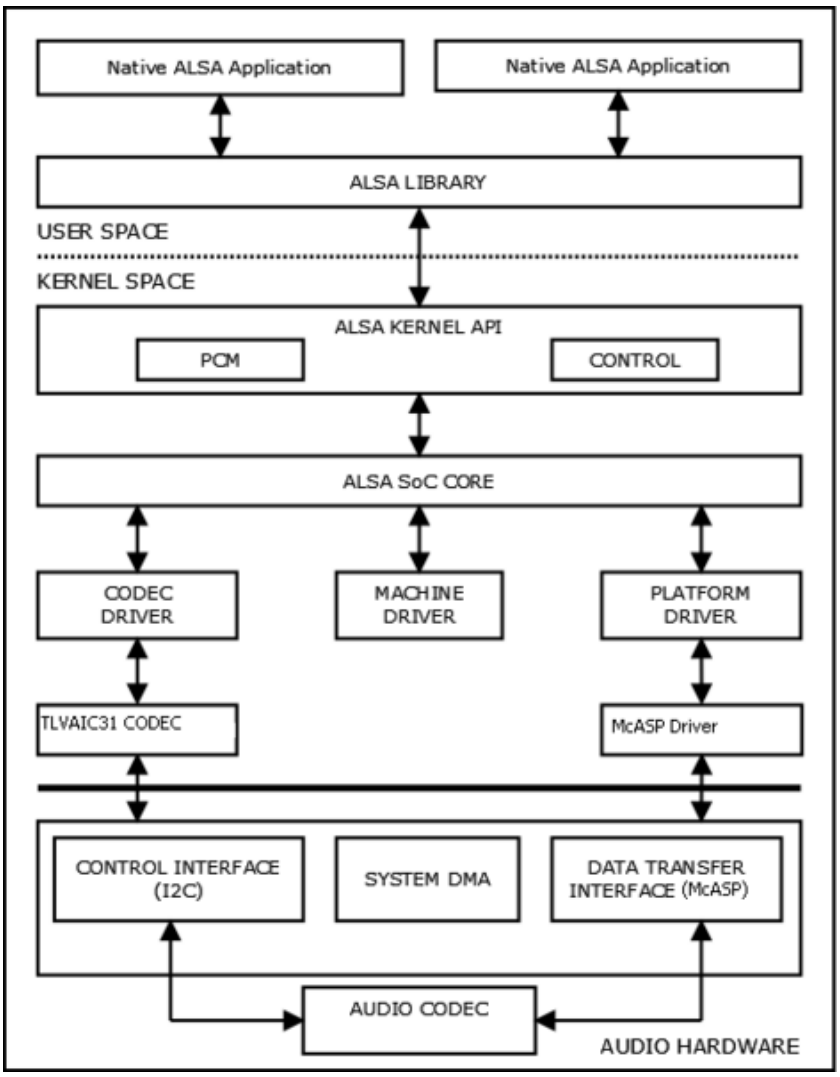

Abbildung 2.11: Texas Instruments Davinci ASoC Architektur mit TLVAIC31 Audiocodec[22]

Aus der Abbildung 2.11 erkennt man die Trennung von Plattform-, Codec- und Maschinentreiber und deren Kommunikation über den ASoC Layer. Der Abstraktionsgrad wird in Richtung nach unten immer geringer.

2.8.2 Soundserver

Für Linux existieren diverse Soundserver, welche in der Regel ALSA als Backend nutzen und verschiedene Funktionen, wie z.B. Audiorouting, vereinfachen[23]. Hervorzuheben

sind hier insbesondere die Soundserver JACK Audio Connection Kit (JACK)[24] und Pulseaudio[25], welche in der Regel über den Paketmanager der entsprechenden Linux-Distribution (im Falle von Debian ist dies der Paketmanager apt) mühelos nachinstalliert werden können. Nachteilig ist, dass immer ein gewisser Overhead entsteht, da nach wie vor ALSA als Backend fungiert. Damit würde sich die Rechenlast, wenn auch nur minimal, erhöhen, was jedoch gerade bei eingebetteten Systemen mit vergleichsweise schwacher Rechenperformance eine größere Auswirkung auf die Latenz haben kann. Aus diesem Grund wurde in der Bachelorarbeit immer direkt die ALSA-Hardwareschnittstelle bzw. ein natives ALSA Plugin genutzt, um die Audio-Schnittstelle anzusprechen.

2.8.3 Device Tree

Ein Device Tree ist eine baumartige Datenstruktur, die eine Hardwareplattform beschreibt. Die Knoten können dabei unterschiedliche Eigenschaften und Kindknoten enthalten. Die Syntax von Device Trees ist an die Programmiersprache C angelehnt und muss ebenso mithilfe eines Device Tree Compilers (DTC) kompiliert werden. Ein kompilierter Device Tree wird als Device Tree Blob (DTB) bezeichnet. Ein DTB wird dem Bootloader, in diesem Fall U-Boot, beim Bootvorgang übergeben, anhand dessen die entsprechenden Treiber bzw. Kernelmodule und deren Abhängigkeiten automatisch in der richtigen Reihenfolge geladen werden. Weiterhin kann innerhalb des Kernel-Kontextes auf die Eigenschaften, wie Handles oder Argumente eines anderen Device Tree Knotens, zugegriffen werden. Einer der Vorteile, insbesondere bei eingebetteten Systemen, besteht darin, dass Änderungen von Parametern der Peripherie ohne Änderung und damit Neukompilierung des Kernelmoduls im Device Tree abgebildet werden können. Weiterhin benötigt man dadurch keine Blacklist[9] mehr, um Linux-Kernel-Module am automatischen Laden zu hindern bzw. zu deaktivieren, da nur noch Linux-Kernel-Module geladen werden, die explizit im Device Tree referenziert sind. Da ein Device Tree sehr umfangreich und damit unübersichtlich sein kann, werden häufig hardwareabhängige Teile, wie die SoC-Plattform, in einzelne .dtsi Dateien aufgeteilt. Mittels eines Präprozessors werden diese vor Beginn der Kompilierung zusammengefügt. Device Trees befinden sich üblicherweise im Verzeichnis arch/arm/boot/dts[10] der Linux-Kernel-Quellen.

[9] Datei, in der Linux-Kernel-Module eingetragen werden, um deren Ausführung zu unterbinden

[10] `https://git.kernel.org/cgit/linux/kernel/git/stable/linux-stable.git/tree/arch/arm/boot/dts`

2.8.4 Device Tree Overlays und Capes

Device Tree Overlays werden genutzt, um Teile des zu ladenden Device Trees zu überdecken bzw. zu modifizieren. So kann beispielsweise eine SPI-Schnittstelle (2.4) aktiviert, konfiguriert und anschließend deren Handle einem Audiocodec-Treiber übergeben werden. Weiterhin können Gerätetreiber so exklusiven Anspruch auf einzelne Hardwarekomponenten, wie GPIOs, anfordern, die dadurch reserviert werden.

Raspberry Pi 2 Model B

Beim Raspberry Pi 2 Model B ist die Schnittstelle zum Konfigurieren des Bootloaders (U-Boot) die Konfigurationsdatei /boot/config.txt[11]. Durch Hinzufügen von „dtoverlay=“ und dem Namen des kompilierten Device Tree Overlays, lädt der Bootloader nach einem Neustart den kompletten DTB, überschreibt dessen im kompilierten Device Tree Overlay definierten Teile und übergibt anschließend dem Kernel den kompletten, modifizierten DTB. Weiterhin kann man dem Bootloader mit dem Befehl „dtparam=“ einzelne Device Tree Parameter übergeben. Anwendung findet dies beispielsweise, um die I^2S-Schnittstelle zu aktiveren. Die kompilierten Device Trees befinden sich im Verzeichnis /boot und die kompilierten Overlays im Verzeichnis /boot/overlays.

BeagleBone Green

Beim BeagleBone Green ist die Schnittstelle, um DTBs zu laden, ähnlich wie beim Raspberry Pi 2 Model B, die Datei /boot/uEnv.txt. Durch Hinzufügen von „dtb=“ und dem Namen des DTBs, wird dem Bootloader (U-Boot) beim Bootvorgang der entsprechende DTB übergeben. Die kompilierten DTBs befinden sich im Verzeichnis /boot/dtbs/<KERNEL_VERSION>/. Weiterhin besteht die Möglichkeit Device Tree Overlays zur Laufzeit über den extra dafür entwickelten Bone-Cape-Manager zu laden oder zu entladen. Device Tree Overlays werden in diesem Zusammenhang als Capes bezeichnet und haben zusätzliche Eigenschaften wie Versionsnummern. Die kompilierten Capes befinden sich im Verzeichnis /lib/firmware und haben eine .dtbo Endung, werden jedoch genauso kompiliert wie Device Trees. Detaillierte Informationen hierüber werden in den anschließenden Kapiteln zur Treiberentwicklung erläutert.

[11] `https://www.raspberrypi.org/documentation/configuration/device-tree.md#part3`

2.8.5 Regmap

Um die Kommunikation mit Audiocodecs und anderer Peripherie zu erleichtern, wurde mit ASoC das Tool Regmap eingeführt. Regmap stellt eine einheitliche, abstrahierte Programmierschnittstelle zur Verfügung, um Registereinträge zu lesen oder zu schreiben. Dadurch müssen in Treibern, die mit externer Peripherie über I^2C (2.5) oder SPI (2.4) kommunizieren, keinen hardwarespezifischen Schnittstellen-Code enthalten und sind damit besser wiederverwendbar. Mittlerweile wird Regmap auch in diversen anderen Bereichen eingesetzt[12]. Ein konkretes Code-Beispiel, um Registereinträge vom Audiocodec auszulesen und auszugeben, befindet sich im Abschnitt 3.4.2.

2.8.6 Realtime Patch

Ein Echtzeitbetriebssystem kann maximale Antwortzeiten von Anwendungen kalkulieren und dadurch anderen Anwendungen nach einer maximal festgelegten Verzögerungszeit Rechenzeit garantieren[26]. Mit dem Realtime Patch kann der Linux-Kernel, insbesondere der Prozess-Scheduler, um den CONFIG_PREEMPT_RT-Modus erweitert werden, welcher die Echtzeitfähigkeit durch Prioritäten von Tasks realisiert. Einem Task mit der höchsten Priorität wird innerhalb einer festen Zeitspanne Rechenzeit gewährt. Dadurch ist die maximale Verzögerungszeit nur abhängig von Tasks mit der gleichen oder höherer Priorität und kann damit leichter und präziser berechnet werden.
Im Gegensatz dazu ist die maximale Antwortzeit beim normalen Prozess-Scheduler von Linux abhängig von allen Tasks, was eine Berechnung der maximalen Antwortzeit wiederum erschwert. Weiterhin garantiert der Prozess-Scheduler nicht, einen Task nach einer maximal festgelegten Rechenzeit zu unterbrechen. Standardmäßig ist im Linux Kernel der CONFIG_PREEMPT_RT-Modus nicht verfügbar, kann jedoch durch den entsprechenden Realtime-Patch erweitert werden (siehe 3.3.3). Die Version des Patches muss mit der des Linux-Kernels übereinstimmen. Zu erwähnen ist noch, dass es sich hierbei um ein weiches Echtzeitsystem handelt, da die Software zwar versucht eine Antwort innerhalb einer festgelegten Zeitspanne zu geben, dies aufgrund der Hardware jedoch nicht zu 100% garantiert werden kann. Ein hartes Echtzeitsystem garantiert im Gegensatz dazu in jedem Fall eine maximale Antwortzeit, da das Echtzeitverhalten durch Kombination von extra dafür vorgesehener Hard- und Software realisiert wird.

[12] `http://elinux.org/images/a/a3/Regmap-_The_Power_of_Subsystems_and_Abstractions.pdf`

3 Linux Treiberentwicklung

3.1 Quellen

Für den allgemeinen Einstieg in die Gerätetreiber-Entwicklung unter Linux gibt es bereits zahlreiche Literatur und Online-Tutorien. Besonders empfehlenswert ist das Buch „Linux Device Drivers“[27], welches auch kostenlos im Internet verfügbar ist[1]. Dieses bietet insbesondere eine aufschlussreiche Einleitung zur Funktionsweise von Linux im Zusammenhang mit Kernel-Modulen.

3.1.1 Kernel- und Treiber-Kompilierung

Da Gerätetreiber bzw. Linux-Kernelmodule nur mit der entsprechenden Kernelversion kompatibel sind, gegen die sie kompiliert wurden, ist es sinnvoll, während des Entwickelns eine bestimmte Kernelversion festzulegen. Im Rahmen dieser Bachelorarbeit wurden die entsprechend angepassten Linux-Kernelquellen der aktuellsten, stabilen Version genutzt[2]. Dies ist sowohl beim Raspberry Pi[3], wie auch beim BeagleBone [4] die Kernelversion 4.1. Als Hilfestellung zum Kompilieren des Linux-Kernels, einschließlich der Gerätetreiber, bietet sich beim Raspberry Pi die Anleitung von elinux.org[28] und beim BeagleBone Green von beyondlogic.org[29] an.

[1] `https://lwn.net/Kernel/LDD3/`
[2] Beim BeagleBone Green musste aufgrund des mangelnden Device Tree Supports, die noch als unstabil eingestufte Version 4.1 verwendet werden
[3] `https://github.com/raspberrypi/linux`
[4] `https://github.com/beagleboard/linux`

3.1.2 Device Tree

Detaillierte Informationen über Device Trees stehen auf der elinux[30] und der Raspberry Pi Foundation Webseite[31] zur Verfügung. Komplette Device Trees für die ARM Architektur befinden sich im Verzeichnis arch/arm/boot/dts[5]. Die Namen entsprechen meistens einem Teil der CPU Plattform und einem Anhängsel, das die Plattform genauer spezifiziert. Beim Raspberry Pi 2 Model B ist dies die Datei bcm2709-rpi-2-b.dts[6] und beim BeagleBone Green am335x-bonegreen.dts[7].

Device Tree Overlays und Capes

Da Device Tree Overlays beim Raspberry Pi und beim BeagleBone unterschiedlich geladen werden, gibt es auch hier unterschiedliche Ressourcen. Beim Raspberry Pi befinden sich die Device Tree Overlays im Verzeichnis arch/arm/boot/dts/overlays[8]. Beim BeagleBone befinden sich diese ebenfalls im Verzeichnis arch/arm/boot/dts[9], wie die Basis Device Trees. Weiterhin gibt es beim BeagleBone zusätzlich den Bone-Cape-Manger (2.8.4), um Device Tree Overlays zu laden. In diesem Kontext werden die kompilierten Device Tree Overlays als Capes bezeichnet. Für diese steht ein extra Repository[32] inklusive Anleitung zum Kompilieren und Laden bzw. Entladen zur Verfügung.

3.1.3 Datenblätter

Teensy 3.1

Detaillierte Hardwareinformationen sind im MK20DX256VLH7 Datenblatt[9] verfügbar.

[5] https://git.kernel.org/cgit/linux/kernel/git/stable/linux-stable.git/tree/arch/arm/boot/dts
[6] https://github.com/raspberrypi/linux/blob/rpi-4.1.y/arch/arm/boot/dts/bcm2709-rpi-2-b.dts
[7] https://github.com/beagleboard/linux/blob/4.1/arch/arm/boot/dts/am335x-bonegreen.dts
[8] https://github.com/raspberrypi/linux/tree/rpi-4.1.y/arch/arm/boot/dts/overlays
[9] https://github.com/beagleboard/linux/tree/4.1/arch/arm/boot/dts

Raspberry Pi

Weitergehende Informationen über das digitale Audio Interface und sonstige Peripherie können im BCM2835 Datenblatt[6] recherchiert werden.

BeagleBone Green

Eine Übersicht über die Peripherie vom BeagleBone Green können im entsprechenden Datenblatt[21] und im ausführlicherem BeagleBone Black Datenblatt[14] nachgeschlagen werden. Detaillierte Hardwareinformationen, wie Funktionsweise und Register des McASP, sind im Datenblatt des AM335X SoC[3] verfügbar, welches auch eine ausführliche Beschreibung über PCM-Audioschnittstellen enthält.

3.1.4 Advanced Linux Sound Architecture (ALSA)

Für ALSA und insbesondere den ASoC Layer ist leider nur sehr wenig Literatur verfügbar, wodurch der Einstieg anfangs recht aufwendig erscheinen kann. Es existieren zwar einige Dokumente, die eine Übersicht über den ASoC Layer geben, allerdings hat sich schnell herausgestellt, dass zur aktiven Entwicklung diverse, konkrete Treiber-Implementierungen benötigt werden. Insbesondere der Vergleich von verschiedenen ASoC-Plattform- und ASoC-Codec-Treibern ist zum Verständnis der Architektur und Funktionsweise von ALSA sehr hilfreich.

ALSA Dokumentation aus dem Linux-Archiv

Eine gute Übersicht über den ASoC Layer, unabhängig von konkreten Quellcodes, bietet die Dokumentation aus den Linux Kernelquellen im Verzeichnis Documentation/sound/alsa/soc[10]. Dort sind die unterschiedlichen Komponenten des ASoC-Layers in einzelne Textdateien aufgeteilt und beschrieben. Als Einstiegsdokument ist hier das Dokument overview.txt[11] zu empfehlen. Im übergeordneten Verzeichnis befinden sich weiterhin Dokumente, die auf ALSA im Allgemeinen eingehen.

[10] `https://www.kernel.org/doc/Documentation/sound/alsa/soc/`
[11] `https://www.kernel.org/doc/Documentation/sound/alsa/soc/overview.txt`

ALSA Treiber API

Die ALSA Treiber API[33] fasst alle vorhanden Funktionen innerhalb des ALSA-Treiber-Kontextes zusammen. Sie untergliedert die Funktionen in die einzelnen Komponenten innerhalb von ALSA, wie z.B. den ASoC-Layer. Entstehen beispielsweise Unklarheiten bei Funktionsparametern, erhält man hier schnell die benötigten Informationen. In Kombination mit konkreten Implementierungen kann so schnell die Funktionsweise von ALSA Treibern nachvollzogen werden. Da in der ALSA-Treiber API keine Beschreibung von Konfigurationskonstanten usw. vorgesehen ist, können diese bei Unklarheiten in der Referenz der ALSA C Bibliothek nachgelesen werden[34].

Konkrete Treiber-Implementierungen

Im Unterverzeichnis sound/soc/[12] der Linux-Kernelquellen befinden sich alle ALSA Treiber, die den ASoC Layer nutzen. Das Unterverzeichnis ist in ASoC-Plattform-Treiber bzw. ASoC-Maschinen-Treiber und ASoC-Codec-Treiber unterteilt. Die ASoC-Codec-Treiber befinden sich im Verzeichnis sound/soc/codecs[13]. Die ASoC-Plattform-/ und ASoC-Maschinen-Treiber befinden sich im jeweiligen Verzeichnis, dass der SoC Architektur entspricht. Im Fall vom Raspberry Pi 2 Model B ist dies das Verzeichnis sound/-soc/bcm[14] und im Fall des BeagleBone Green das Verzeichnis sound/soc/davinci[15].

Anleitung „Writing an ALSA Driver“ von Takashi Iwai

Im Web wird häufig auf die Anleitung „Writing an ALSA Driver“[35] von Takashi Iwai zur Einführung in die ALSA-Treiberprogrammierung verwiesen. Leider ist diese unübersichtlich und in manchen Dingen so detailliert, dass sie zur Einarbeitung nicht empfehlenswert ist. Dennoch bietet sie während der Entwicklung in speziellen Fällen, wie z.B. der PCM-Schnittstelle, nützliche Informationen zum Nachlesen. Ein weiterer Nachteil ist, dass die Anleitung keinerlei Auskunft über den ASoC Layer gibt.

[12] `https://git.kernel.org/cgit/linux/kernel/git/stable/linux-stable.git/tree/sound/soc`
[13] `https://git.kernel.org/cgit/linux/kernel/git/stable/linux-stable.git/tree/sound/soc/codecs`
[14] `https://github.com/raspberrypi/linux/tree/rpi-4.1.y/sound/soc/bcm`
[15] `https://github.com/beagleboard/linux/tree/4.1/sound/soc/davinci`

3.2 Planung

Um ALSA-Treiber (2.8.1), die den ASoC (2.8.1) Layer nutzen, zu entwickeln, sollte immer die Aufteilung in Plattform-Treiber, Maschinen-Treiber und Codec-Treiber berücksichtigt werden. Außerdem sollte man beachten, dass die Treiber zwar in der imperativen Programmiersprache C geschrieben werden, jedoch nach dem objektorientierten Programmierparadigma funktionieren. In diesem Fall gibt es zwar keine Vererbung oder Polymorphie, dennoch können Objekte über Strukturen mithilfe von Zeigern abgebildet werden. Da die Audiocodecs zu Beginn der Bachelorarbeit festgelegt worden sind, ist es ein guter Einstieg den konkreten Codec-Treiber[16] mit dem entsprechenden Datenblatt[5] zu vergleichen und zu verstehen. Einen weiteren guten Einstieg bietet der ASoC-Maschinen-Treiber[17] vom SuperAudioBoard von RF William Hollender[1], da dieser recht kompakt und damit übersichtlich ist. Es folgt eine Auflistung mit Funktionsbeschreibung der einzelnen ASoC-Treiber-Komponenten, um die Grundkonzepte verständlich darzustellen:

[16] `https://github.com/raspberrypi/linux/blob/rpi-4.1.y/sound/soc/codecs/ad193x.c`
[17] `https://github.com/whollender/linux/blob/rpi-4.0.y/sound/soc/bcm/superaudioboard.c`

Komponente	gehört zu	Beschreibung
bcm2708-i2s.h	Plattform-Treiber	Enthält Pin Konfigurationen und exportierte Funktionsdeklarationen.
bcm2708-i2s.c	Plattform-Treiber	Konkrete Implementierung von Plattform-Treiber für den Raspberry Pi 2 Model B und Vorgängermodelle.
cs4271.h	Codec-Treiber	Enthält Regmap- und Device-Tree-Strukturdeklaration und exportierte Treiber-Initialisierungsfunktion (probe()).
cs4271-i2c.c	Codec-Treiber	Enthält Device Tree Identifikatoren und Regmap-Konfiguration für die I^2C-Schnittstelle.
cs4271-spi.c	Codec-Treiber	Enthält Device Tree Identifikatoren und Regmap-Konfiguration für SPI-Schnittstelle.
cs4271.c	Codec-Treiber	Enthält konkrete Treiber-Implementierung von CS4271 Audiocodec. Der CS4271 ist kompatibel zu dem CS4272, weshalb auch der gleiche Codec-Treiber verwendet werden kann.
superaudioboard.c	Maschinen-Treiber	Konkrete Treiber-Implementierung, die Plattform- und Codec-Treiber verbindet.

Tabelle 3.1: SuperAudioBoard ALSA Treiber-Komponenten

Insbesondere der Zusammenhang von ASoC-Plattform- und ASoC-Codec-Treiber und wie die Verbindung dieser, sollte nachvollzogen werden können. Zu beobachten ist, dass sowohl Plattform wie auch Codec-Treiber jeweils die zwei folgenden Strukturen aus include/sound/soc-dai.h[18] verwenden, um das digitale Audio Interface zu definieren:

```
static const struct snd_soc_dai_ops = {
  ...
  /*
    Konfigurtiert Hardwareparameter, wie z.B. Abtastrate,
    vor Beginn von Audiostream
```

[18] https://git.kernel.org/cgit/linux/kernel/git/stable/linux-stable.git/tree/include/sound/soc-dai.h

```
  */
  int (*hw_params)(struct snd_pcm_substream *,
      struct snd_pcm_hw_params *, struct snd_soc_dai *);
  /*
    Konfiguriert gleichbleibende Parameter, wie z.B.
    Taktpolaritäten, Taktmaster, usw.
  */
  int (*set_fmt)(struct snd_soc_dai *dai, unsigned int fmt);
  /*
    Konfiguriert Taktfrequenz der Masterclock
  */
  int (*set_sysclk)(struct snd_soc_dai *dai,
      int clk_id, unsigned int freq, int dir);
  ...
};
static struct snd_soc_dai_driver = {
  ...
  /*
    Name von entsprechender Komponente (zb. cs4271-hifi)
  */
  const char *name;
  /*
    Definiert mögliche Hardwareparameter, wie z.B. Abtastraten,
    PCM-Formate, Anzahl Kanäle, usw. von digitalem Audiointerface
  */
  struct snd_soc_pcm_stream capture;
  struct snd_soc_pcm_stream playback;
  /*
    siehe oben
  */
  const struct snd_soc_dai_ops *ops;
  ...
};
```

Die Strukturen setzen sich aus anderen Datentypen und Void-Zeigern zusammen, welche auf die entsprechende Funktion in der jeweiligen Treiber-Komponente zeigen. Im Maschinen-Treiber werden diese Strukturen anschließend genutzt, um Plattform- und Codec-Treiber miteinander zu verbinden. Dies erkennt man insbesondere an folgender Struktur aus include/sound/soc.h[19]:

[19] https://git.kernel.org/cgit/linux/kernel/git/stable/linux-stable.git/tree/include/sound/soc.h

```
static struct snd_soc_dai_link = {
  ...
  /*
    Frei wählbarer Soundkartenname (z.B. AudioCard)
  */
  const char *name;
  /*
    Frei wählbarer PCM-Streamname (z.B. AudioCard TDM)
  */
  const char *stream_name;
  /*
    Wird mit DAI Knoten von Device Tree befüllt
  */
  struct device_node *cpu_of_node;
  /*
    Name von CPU DAI (ist in snd_soc_dai_driver-Struktur von jeweiligem
    Plattform-Treiber definiert)
  */
  const char *cpu_dai_name;
  /*
    Wird von Device Tree befüllt (z.B. CS4271-Knoten)
  */
  struct device_node *codec_of_node;
  /*
    Name von Codec DAI (ist in snd_soc_dai_driver-Struktur von jeweiligen
    Audiocodec-Treiber definiert)
  */
  const char *codec_dai_name;
  /*
    Definiert DAI-Format, wie z.B. Bitpolarität, über
    Initialisierungsfunktion in der jeweiligen Treiberkomponente
  */
  unsigned int dai_fmt;
  /*
    Enthält Zeiger auf Funktionen von Maschinen-Treiber,
    um vor Beginn eines PCM-Streams beispielsweise Hardwareparameter,
    wie z.B. Abtastrate zu konfigurieren.
    Innerhalb der Funktion werden die entsprechenden Funktionen
    des Plattform-Treibers und Audiocodec-Treibers angesprochen
    (siehe oben).
  */
  const struct snd_soc_ops *ops;
```

```
    /*
      Zeiger auf Initialisierungsfunktion in Maschinen-Treiber,
      welche wiederrum die entsprechende Funktion von
      Audiocodec-Treiber und Plattform-Treiber ausführt.
    */
    int (*init)(struct snd_soc_pcm_runtime *rtd);
    ...

};
```

Um das SuperAudioBoard mit dem Raspberry Pi 2 Model B bzw. Raspbian zu nutzen, ist ein von Raspberry geforkter Linux-Kernel[20] verfügbar, der die entsprechend benötigten ALSA-Treiber und Device Tree Overlays enthält. Die Kompilierung wurde nach der unten beschriebenen Anleitung für den Raspberry Pi (3.1.1) durchgeführt.

Informationen über Device Tree Parameter sind in der jeweiligen Datei im Verzeichnis Documentation/devicetree/bindings[21] verfügbar.

3.3 Implementierung

Zur Entwicklung wurden sowohl der Raspberry Pi Kernel[36], wie auch der BeagleBone Kernel[15] auf Github.com geforked und modifiziert. Da die komplette Beschreibung der Treiber so umfangreich ist, dass sie den Rahmen dieses Thesisdokuments sprengen würde, wird in diesem Kapitel nur grundlegend auf die einzelnen Komponenten der Treiber eingegangen. Um die komplette Funktionsweise der Treiber im Detail zu verstehen, sollten die konkreten Implementierungen mithilfe der oben genannten Quellen verwendet werden.

Als erstes wurde mithilfe des ASoC-Maschinen-Treibers vom SuperAudioBoard eine Vorlage für einen eigenen Treiber erstellt. Damit sind in der Treiber-Vorlage folgende Funktionen und Strukturen bzw. Objekte vorhanden.

[20] https://github.com/whollender/linux
[21] https://git.kernel.org/cgit/linux/kernel/git/stable/linux-stable.git/tree/Documentation/devicetree/bindings

Funktionen:

```
/*
  Wird einmalig nach Laden von Kernelmodul zum Initalisieren von Audio
  Interface aufgerufen, um beispielsweise Taktfrequenzen zu konfigurieren.
*/
static int snd_audiocard_init(struct snd_soc_pcm_runtime *rtd) {...}
/*
  Kann zu jedem Zeitpunkt aufgerufen werden, um beispielsweise
  PCM-Format und Abtastrate zu konfigurieren.
*/
static int snd_audiocard_hw_params(
  struct snd_pcm_substream *substream, struct snd_pcm_hw_params *params) {
    ...
  }
/*
  Wird einmalig vor Beginn von PCM-Stream ausgeführt,
  um PCM Einstellungen vorzunehmen.
*/
static int snd_audiocard_startup(struct snd_pcm_substream *substream){
  ...
}
/*
  Wird einmalig nach Ende von PCM-Stream ausgeführt,
  um PCM Einstellungen zurückzunehmen.
*/
static void snd_audiocard_shutdown(struct snd_pcm_substream *substream) {
  ...
}
/*
  Wird einmalig beim Laden von Kernel-Modul aufgerufen.
*/
static int snd_audiocard_probe(struct platform_device *pdev) {
  ...
}
/*
  Wird einmalig beim Entladen von Kernel-Modul aufgerufen.
*/
static int snd_audiocard_remove(struct platform_device *pdev) {
  ...
}
```

Listing 3.1: Callback Funktionen im AD1938 AudioCard Maschinen-Treiber

Strukturen bzw. Objekte:

```
/*
  Enthält Funktionen, die zu Beginn oder Ende von PCM-Stream ausgeführt
  werden, um PCM-Einstellungen vorzunehmen.
*/
static struct snd_soc_ops snd_audiocard_ops = {...};
/*
  Verbindet CPU DAI mit Audiocodec DAI und
  konfiguriert Übertragungsprotokoll.
*/
static struct snd_soc_dai_link snd_audiocard_dai = {...};
/*
  Repräsentiert den ASoC-Maschinen-Treiber und
  enthält Plattform <--> Audiocodec Interface.
*/
static struct snd_soc_card snd_audiocard = {...};
/*
  Legt Device Tree Identifikatoren fest, um Kernel-Modul
  in Device Tree zu referenzieren.
*/
static const struct of_device_id snd_audiocard_of_match[] = {...};
/*
  Wird genutzt um ASoC-Maschinen-Treiber im Linux-Kernel als
  Plattform-Geräte-Treiber zu registrieren.
*/
static struct platform_driver snd_audiocard_driver = {...};
```

Listing 3.2: Objekte im AD1938 AudioCard Maschinen-Treiber

Da der AD193X ASoC-Audiocodec-Treiber ad193x-spi.c[22] seit 2010 nicht mehr überarbeitet wurde und zu diesem Zeitpunkt Device Trees noch nicht vollständig im Linux-Kernel implementiert waren, musste der SPI-Codec-Treiber so modifiziert werden, dass er entsprechende Device Tree Identifikatoren exportiert. Wäre dies nicht der Fall, könnte der Codec-Treiber später nicht automatisch über einen DTB (2.8.4) geladen und konfiguriert werden.

```
/*
  SPI Device Tree Identifikatoren
*/
static const struct spi_device_id ad193x_spi_id[] = {
```

[22] https://github.com/henrix/rpi-linux/blob/rpi-4.1.y/sound/soc/codecs/ad193x-spi.c

```
  { "ad1938", },
  { "ad1939", },
  { },
};
MODULE_DEVICE_TABLE(spi, ad193x_spi_id);

/*
  Audiocodec Device Tree Identifikatoren
*/
static const struct of_device_id ad193x_of_match[] = {
  { .compatible = "analog,ad1938", },
  { .compatible = "analog,ad1939", },
  { }
};
MODULE_DEVICE_TABLE(of, ad193x_of_match);

/*
  Exportierte Struktur, um gesamten Treiber zu Laden bzw. zu Entladen.
*/
static struct spi_driver ad193x_spi_driver = {
  .driver = {
    .name = "ad193x",
    .owner  = THIS_MODULE,
    .of_match_table = ad193x_of_match,
  },
  .probe    = ad193x_spi_probe,
  .remove   = ad193x_spi_remove,
  .id_table = ad193x_spi_id
};
module_spi_driver(ad193x_spi_driver);
```

Listing 3.3: AD193X ASoC-Audiocodec-Treiber Device Tree Modifizierungen

3.3.1 Raspberry Pi 2

Es folgt eine kurze Beschreibung der snd_soc_dai_link-Struktur aus dem ASoC-Maschinen-Treiber[23], welche das Plattform und Audiocodec-Interface verbindet:

```
static struct snd_soc_dai_link snd_rpi_audiocard_dai = {
```

[23] https://github.com/henrix/rpi-linux/blob/rpi-4.1.y/sound/soc/bcm/audiocard.c

```
/*
  Name der Soundkarte (frei wählbar)
*/
.name = "AudioCard",
/*
  Name von PCM-Stream (frei wählbar)
*/
.stream_name = "AudioCard HiFi",
/*
  Name von Plattform-Interface (befindet sich in snd_soc_dai_driver
  Struktur von ASoC-Plattform-Treiber (bcm2708-i2s.c))
*/
.cpu_dai_name = "bcm2708-i2s.0",
/*
  Name von Audiocodec-Interface (befindet sich in snd_soc_dai_driver
  Struktur von ASoC-Audiocodec-Treiber (ad193x.c))
*/
.codec_dai_name = "ad193x-hifi",
/*
  Name von Plattform-Treiber (befindet sich in platform_driver Struktur
  von ASoC-Plattform-Treiber (bcm2708-i2s.c))
*/
.platform_name = "bcm2708-i2s.0",
/*
  Name von Audiocodec-Konfigurations-Interface.
  Entspricht der SPI-Schnittstelle und Chip-Select.
*/
.codec_name = "spi0.0",
/*
  Konfiguriert Übertragungsprotokoll (I2S), Bitclock- und
  Frameclock-Polarität (normale Bitclock und invertierte Frameclock) und
  Codec-Clock (Codec ist Bitclock- und Frameclock-Master)
*/
.dai_fmt = SND_SOC_DAIFMT_I2S | SND_SOC_DAIFMT_NB_IF |
 SND_SOC_DAIFMT_CBM_CFM,
/*
  Enthält PCM-Stream Operationen (startup(), shutdown(), hw_params())
*/
.ops = &snd_rpi_audiocard_ops,
/*
  Zeigt auf Initalisierungsfunktion
*/
```

```
  .init = snd_rpi_audiocard_init,
};
```

Listing 3.4: Verbindung von BCM2708 CPU Interface und AD1938 Audiocodec Interface auf dem Raspberry Pi 2 Model B

Der Raspberry Pi 2 Model B wird mit folgenden Pins der AD1938 AudioCard verbunden:

AD1938 AudioCard Pin Name	**Raspberry Pi pin #**	**Raspberry Pi Pin name**
I2S_MCLK	Nicht benutzt	Nicht benutzt
I2S_DAC_LRCLK	35	GPIO19
I2S_DAC_BCLK	12	GPIO18
I2S_ADC_LRCLK	35	GPIO19
I2S_ADC_BCLK	12	GPIO18
I2S_DAC_D1	40	GPIO21
12S_ADC_D1	38	GPIO20
SPI_L	24	GPIO8 (SPI_CE0_N)
SPI_C	23	GPIO11 (SPI_CLK)
SPI_O	21	GPIO9 (SPI_MISO)
SPI_I	19	GPIO10 (SPI_MOSI)
PWRIN_G	39	GND
PWRIN_50	2	+5V

Tabelle 3.2: Pin-Verbindungen von AD1938 AudioCard und Raspberry Pi 2 Model B

Es folgt der Device Tree Overlay[24] für die AD1938 AudioCard mit Beschreibung, welcher auch auf Basis des SuperAudioBoards[25] erstellt wurde:

```
/ {
  compatible = "brcm,bcm2708";

  /*
    Aktiviert AD1938 AudioCard und übergibt
```

[24] https://github.com/henrix/rpi-linux/blob/rpi-4.1.y/arch/arm/boot/dts/overlays/audiocard-overlay.dts

[25] https://github.com/whollender/linux/blob/rpi-4.0.y/arch/arm/boot/dts/overlays/superaudioboard-overlay.dts

```
  ASoC-Maschinen-Treiber I2S-Schnittstelle
*/
fragment@0 {
  target = <&sound>;
  __overlay__ {
    compatible = "audiocard,audiocard";
    i2s-controller = <&i2s>;
    status = "okay";
  };
};

/*
  Aktiviert die I2S-Schnittstelle von CPU (ASoC-Plattform-Treiber)
*/
fragment@1 {
  target = <&i2s>;
  __overlay__ {
    status = "okay";
  };
};

/*
  Aktiviert und konfiguiert Audiocodec (ASoC-Audiocodec-Treiber)
*/
fragment@2 {
  target = <&spi0>;
  __overlay__ {
    #address-cells = <1>; // Unterdrückt Compilerwarnungen
    #size-cells = <0>;
    status = "okay";

    /*
      Legt Chip-Select (CS_0) für Audiocodec fest
    */
    ad193x: ad193x@0{
      compatible = "analog,ad1938";
      reg = <0>;
      status = "okay";

      spi-max-frequency = <10000000>;
    };
  };
```

```
    };
};
```

Listing 3.5: Device Tree Overlay für AD1938 AudioCard auf dem Raspberry Pi 2 Model B

Im Laufe der Treiber-Entwicklung wurde festgestellt, dass es auf Basis des Raspberry Pi 2 Model B keine Möglichkeit gibt, den TDM Modus und damit mehr als 2 Audiokanäle softwaretechnisch zu unterstützen. Daher wurde nach einer Alternativplattform gesucht.

3.3.2 BeagleBone Green

Aufgrund der guten Beschreibung der Funktionsweise von ALSA und des ASoC-Layers [22] im Zusammenhang mit dem Multichannel Audio Serial Port (McASP), hat ergab sich die AM335X CPU von Texas Instruments[3] als passende Alternative. Weiterhin waren die Einplatinencomputer BeagleBone Black und BeagleBone Green bereits verfügbar, welche eine AM335X CPU nutzen. Da der BeagleBone Green etwas kostengünstiger als der BeagleBone Black ist und für diesen Einsatzzweck vorerst kein HDMI bzw. keine grafische Oberfläche benötigt wird, schien der BeagleBone Green als geeignet. Beim BeagleBone Green wird die Verbindung von CPU und Audiocodec-Interface teilweise über den Device Tree realisiert. Die snd_soc_dai_link-Struktur ist daher eher als ein Platzhalter zu betrachten, welche über den Device Tree befüllt wird. Im Vergleich zum Raspberry Pi 2 Model B, ändern sich lediglich die Argumente, welche die CPU Plattform betreffen. Der BeagleBone Green bietet weiterhin die Möglichkeit über Pin-Multiplexing verschiedene Signale an den externen Pins herauszuführen. Jeder Pin verfügt über 8 verschiedene Modi, welche über den Device Tree konfiguriert werden können (siehe Listing 3.6). Der BeagleBone Green wird mit folgenden Pins der AD1938 AudioCard verbunden:

AD1938 AudioCard Pin Name	**BBG pin #**	**BBG mode name**	**BBG Pin name**
MCLKXO	P9.25 Mode 0	mcasp0_ahclkx	GPIO3_21
I2S_DAC_LRCLK	P9.29 Mode 0	mcasp0_fsx	SPI1_D0
I2S_DAC_BCLK	P9.31 Mode 0	mcasp0_aclkx	SPI1_SCLK
I2S_ADC_LRCLK	P9.27 Mode 0	mcasp0_fsr	GPIO3_19
I2S_ADC_BCLK	P9.12 Mode 6	mcasp0_aclkr	GPIO1_28
I2S_DAC_D1	P9.28 Mode 2	mcasp0_axr2	SPI1_CS0
12S_ADC_D1	P9.30 Mode 0	mcasp0_axr0	SPI1_D1
SPI_L	P9.17 Mode 0	spi0_cs0	I2C1_SCL
SPI_C	P9.22 Mode 0	spi0_sclk	UART2_RXD
SPI_O	P9.21 Mode 0	spi0_d0	UART2_TXD
SPI_I	P9.18 Mode 0	spi0_d1	I2C1_SDA
RESET_IN	P9.15 Mode 7	gpio1[16]	GPIO1_16
PWRIN_G	P9.1/P9.2	GND	GND
PWRIN_50	P9.7/P9.8	SYS_5V	SYS_5V

Tabelle 3.3: Pin Verbindungen von AD1938 AudioCard und BeagleBone Black/Green

Als Vorlage für die eigenen BeagleBone Capes diente der offizelle BeagleBone Black Audio Cape RevB[37] und wurde nach dem selben Verfahren kompiliert. Details der Pinkonfigurationen befinden sich in der BeagleBone Black System Reference Manual[14, S. 101 ff]. Detaillierte Informationen über Device Tree Parameter für den McASP können der Datei davinci-mcasp-audio.txt[26] aus den Linux-Kernen-Quellen entnommen werden. Um asynchron Audiodaten abspielen bzw. aufnehmen zu können, wurden zusätzlich 2 unterschiedliche Taktzonen für das digitale Audiointerface des Audiocodecs und der CPU konfiguriert. Standardmäßig ist dies im McASP-Plattform-Treiber[27] im Zusammenhang mit I^2S deaktiviert, konnte jedoch durch Ersetzen der Zeile 911 mit folgender Zeile aktiviert werden:

```
mcasp_set_bits(mcasp, DAVINCI_MCASP_ACLKXCTL_REG, TX_ASYNC);
```

[26] https://github.com/henrix/beagle-linux/blob/4.1/Documentation/devicetree/bindings/sound/davinci-mcasp-audio.txt

[27] https://github.com/henrix/beagle-linux/blob/4.1/sound/soc/davinci/davinci-mcasp.c

Es folgt der BeagleBone Cape für 8 TDM-Slots[28] für die AD1938 AudioCard mit Erläuterung:

```
/ {
  compatible = "ti,beaglebone", "ti,beaglebone-black", "ti,beaglebone-
    green";

  /* Cape Idenfikatoren und Version */
  part-number = "BBB-AD193X-8TDM";
  version = "00A0", "A0";

  /*
    Reserviert einzelne Pins und Hardwarekomponenten
  */
  exclusive-use =
    /* audio */
    "P9.30",  /* mcasp0_axr0    audio in */
    "P9.28",  /* mcasp0_axr2    audio out */
    "P9.31",  /* mcasp0_ahclkx  transmit bitclock */
    "P9.29",  /* mcasp0_fsx     transmit frameclock */
    "P9.12",  /* mcasp0_aclkr   receive bitclock */
    "P9.27",  /* mcasp0_fsr     receive frameclock */
    "P9.25",  /* mcasp0_ahclkx  masterclock */
    /* spi */
    "P9.22",  /* spi0_sclk */
    "P9.21",  /* spi0_d0 */
    "P9.18",  /* spi0_d1 */
    "P9.17",  /* spi0_cs0 */
    /* the hardware ip uses */
    "mcasp0", "spi0";

  /*
    Konfiguriert die benötigten Pins
  */
  fragment@0 {
    target = <&am33xx_pinmux>;
    __overlay__ {
      mcasp0_pins: pinmix_mcasp0_pins {
      pinctrl-single,pins = <
        0x1ac 0x20    /* mcasp0_ahclkx, MODE0 | INPUT_PULLDOWN | P9_25 */
        0x19c 0x02    /* mcasp0_axr2,   MODE2 | OUTPUT_PULLDOWN | P9_28 */
        0x194 0x20    /* mcasp0_fsx,    MODE0 | INPUT_PULLDOWN | P9_29 */
        0x190 0x20    /* mcasp0_aclkx,  MODE0 | INPUT_PULLDOWN | P9_31 */
```

[28] https://github.com/henrix/beagle-linux/blob/4.1/BBB-AD193X-8TDM-00A0.dts

```
      0x1a4 0x20    /* mcasp0_fsr,    MODE0 | INPUT_PULLDOWN | P9_27 */
      0x078 0x26    /* mcasp0_aclkr,  MODE6 | INPUT_PULLDOWN | P9_12 */
      0x198 0x20    /* mcasp0_axr0,   MODE0 | INPUT_PULLDOWN | P9_30 */
    >;
    };

    audiocard_spi0_pins: pinmux_audiocard_spi0_pins {
    pinctrl-single,pins = <
      0x150 0x30    /* spi0_sclk, MODE0 | INPUT_PULLUP | P9_22 */
      0x154 0x30    /* spi0_d0,   MODE0 | INPUT_PULLUP | P9_21 */
      0x158 0x10    /* spi0_d1,   MODE0 | OUTPUT_PULLUP | P9_18 */
      0x15c 0x10    /* spi0_cs0,  MODE0 | OUTPUT_PULLUP | P9_17 */
    >;
    };
  };
};

/*
  Aktiviert und Konfiguriert die SPI-Schnittstelle mit
  AD1938 Audiocodec
*/
fragment@1 {
  target = <&spi0>; //Legt die SPI-Schnittstelle fest
  __overlay__ {
    /* avoid dtc warnings */
    #address-cells = <1>;
    #size-cells = <0>;
    pinctrl-names = "default";
    pinctrl-0 = <&audiocard_spi0_pins>;
    status = "okay";

    /*
      Legt Chip-Select (CS_0) für Audiocodec fest
    */
    ad193x: ad193x@0{
      spi-max-frequency = <10000000>;
      reg = <0>;
      /*
        Referernziert entsprechenden Audiocodec
      */
      compatible = "analog,ad1938";
    };
```

```
    };
};

/*
  Aktiviert und Konfiguriert Multichannel Audio Serial Port
*/
fragment@2 {
  target = <&mcasp0>;
  __overlay__ {
    pinctrl-names = "default";
    pinctrl-0 = <&mcasp0_pins>;
    status = "okay";

    op-mode = <0>; //McASP in I2S-Modus
    tdm-slots = <8>;
    num-serializer = <16>;
    /*
      Nutze mcasp_axr0 als Eingang und mcasp_axr2 als Ausgang
    */
    serial-dir = <  /* 0: INACTIVE, 1: TX, 2: RX */
      2 0 1 0
      0 0 0 0
      0 0 0 0
      0 0 0 0
    >;
    tx-num-evt = <1>;
    rx-num-evt = <1>;
  };
};

/*
  Aktiviert und konfiguriert AD1938 AudioCard.
  Übergibt ASoC-Maschinen-Treiber McASP-Interface
  und Audiocodec-Interface.
*/
fragment@3 {
  target = <&ocp>;
  __overlay__ {
    sound {
      compatible = "audiocard,audiocard";
      model = "AD193X AudioCard 8TDM";
      audio-codec = <&ad193x>;
```

```
        mcasp-controller = <&mcasp0>;
        audiocard-tdm-slots = <8>;
        codec-clock-rate = <12288000>;
        cpu-clock-rate = <24576000>;
        audio-routing =
          "Line Out",              "DAC1OUT",
          "Line Out",              "DAC2OUT",
          "Line Out",              "DAC3OUT",
          "Line Out",              "DAC4OUT",
          "ADC1IN",                "Line In",
          "ADC2IN",                "Line In";
      };
    };
  };
};
```

Listing 3.6: BeagleBone Cape (Device Tree Overlay) für AD1938 AudioCard mit 8 TDM-Slots

3.3.3 Kernel-/Treiber-Kompilierung und Installation

Um den Einstieg in die Kernel- und Treiber-Kompilierung zu erleichtern, folgt jeweils eine kurze Schritt-für-Schritt-Anleitung für den Raspberry Pi und den BeagleBone für die Nutzung der AD1938 AudioCard. Die Kompilierung wurde unter der Linux Distribution Ubuntu 14.04[29] ausgehend vom Heimverzeichnis durchgeführt.

Raspberry Pi 2 Model B

```
# Installiere Build-Tools
sudo apt-get install gcc-arm-linux-gnueabi git lzop libssl-dev \
libncurses5-dev wget u-boot-tools
# Lade geforktes Raspberry Pi Linux-Repository herunter und wechsle in
# dieses
git clone https://github.com/henrix/rpi-linux.git && cd rpi-linux
# Räume Build-Verzeichnis auf und erstelle ein temporäres Verzeichnis
# für Module
make mrproper && mkdir modules_tmp
```

[29] http://www.ubuntu.com/desktop

```
# Generiere Linux-Kernel-Konfigurationsdatei für Raspberry Pi 2
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- bcm2709_defconfig
# (Optional) Lade Realtime-Patch herunter
wget https://www.kernel.org/pub/linux/kernel/projects/rt/4.1/patch-4.1.10-
    rt11.patch.xz
# (Optional) Patche Kernel mit Realtime-Patch
xzcat ../patch-4.1.10-rt11.patch.xz | patch -p1
# (Optional) Konfigurationsdatei anpassen
# Kernel Features -> Preemption Model -> Fully Preemptible Kernel (RT)
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- menuconfig
# Kompiliere Kernel
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- -j5
# Kompiliere Kernelmodule
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- modules -j5
# Installiere Kernelmodule in temporäres Verzeichnis
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- \
INSTALL_MOD_PATH=modules_tmp modules_install
# Hänge Kernel Flag für Device Tree Nutzung an (dadurch erkennt U-Boot,
# dass es sich um einen Kernel, der Device Trees unterstützt, handelt
# und übergibt dem Kernel den entsprechenden Device Tree Blob)
./scripts/mkknlimg arch/arm/boot/Image arch/arm/boot/rpi-4.1.img
# Kopiere Kernel auf Raspberry Pi (SD-Karte in ~/mnt eingehängt)
cp arch/arm/boot/rpi-4.1.img ~/mnt/boot/
# Kopiere Kernelmodule auf Raspberry Pi
cp -r modules_tmp/lib ~/mnt/
# Kopiere Device Tree Overlays auf Raspberry Pi
cp -r arch/arm/boot/dts/overlays/*.dtb ~/mnt/boot/overlays
# (Die nächsten Schritt werden auf dem Raspberry Pi ausgeführt)
# Aktualisiere Raspberry Pi Firmware und starte anschließend neu
sudo rpi-update
# Aktiviere neuen Kernel in Konfigurationsdatei /boot/config.txt
# durch Hinzufügen von
kernel=rpi-4.1.img
# Aktiviere I2S-, SPI-Schnittestelle und Device Tree Overlay für
# AD1938 AudioCard durch Hinzufügen von
dtparam=i2s=on,spi=on
dtoverlay=audiocard
# Kontrolliere nach einem Neustart mit aplay -l, ob
# AD1938 AudioCard erkannt worden ist
```

Listing 3.7: Anleitung zur Kompilierung und Konfiguration von eigenem Raspberry Pi Kernel mit AD1938 AudioCard

BeagleBone Green

```
# Installiere Build-Tools
sudo apt-get install gcc-arm-linux-gnueabi git lzop libssl-dev \
libncurses5-dev wget u-boot-tools
# Lade geforktes BeagleBone Green Linux-Repository herunter und wechsle in
# dieses
git clone https://github.com/henrix/beagle-linux.git && cd beagle-linux
# Räume Build-Verzeichnis auf und erstelle ein temporäres Verzeichnis
# für Module
make mrproper && mkdir modules_tmp
# Generiere Linux-Kernel-Konfigurationsdatei für BeagleBone
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- bb.org_defconfig
# (Optional) Lade Realtime-Patch herunter
wget https://www.kernel.org/pub/linux/kernel/projects/rt/4.1/patch-4.1.10-
    rt11.patch.xz
# (Optional) Patche Kernel mit Realtime-Patch
xzcat ../patch-4.1.10-rt11.patch.xz | patch -p1
# (Optional) Konfigurationsdatei anpassen
# Kernel Features -> Preemption Model -> Fully Preemptible Kernel (RT)
# Block Layer -> I/O scheduler -> Deadline
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- menuconfig
# Kompiliere Kernel
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- -j5
# Kompiliere U-Boot-Image und Device Tree Blobs
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- uImage dtbs \
LOADADDR=0x80008000 -j8
# Kompiliere Kernelmodule
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- modules -j5
# Installiere Kernelmodule in temporäres Verzeichnis
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- \
INSTALL_MOD_PATH=modules_tmp modules_install
# Kopiere Kernel auf BeagleBone Green (SD-Karte in ~/mnt eingehängt)
cp arch/arm/boot/zImage ~/rootfs/boot/vmlinuz-<KERNEL_VERSION>
# Kopiere Kernelmodule auf BeagleBone Green
sudo cp -r modules_tmp/lib/ ~/mnt/rootfs/
# Kopiere Device Tree Blobs auf Beaglebone Green
cp -r arch/arm/boot/dts/*.dtb \
~/mnt/rootfs/boot/dtbs/<KERNEL_VERSION>/
# Kopiere Kernel-Konfigurationsdateia auf BeagleBone Green
sudo cp .config ~/mnt/rootfs/boot/config-<KERNEL_VERSION>
# Kopiere AD1938 AudioCard Cape auf BeagleBone Green
cp BBB-AD193X-8TDM-00A0.dts ~/mnt/rootfs/root
```

```
# (Die nächsten Schritt werden auf dem BeagleBone Green ausgeführt)
# Aktiviere neuen Kernel in Konfigurationsdatei /boot/uEnv.txt
# durch Hinzufügen von
uname_r=<KERNEL_VERSION>
# Konfiguriere U-Boot für Device Tree
# Hinweis: Während der Thesis wurde anfangs immer der Device Tree für den
# BeagleBone Green genutzt. Nach sehr langer und aufwendiger Fehlersuche
# hat sich allerdings herausgestellt, dass der Multichannel Audio Serial
# Port nicht fehlerfrei mit dem Beaglebone Green Device Tree funktioniert.
# Durch erstellen eines neuen Device Trees auf Basis des BeagleBone Black
# Device Trees konnte das Problem behoben werden (siehe 5.2)
dtb=am335x-bonegreen-audiocard.dtb
# Kompiliere und installiere Device Tree Compiler
git clone https://github.com/beagleboard/bb.org-overlays &&
cd ./bb.org-overlays
./dtc-overlay.sh
./install.sh
# Starte BeagleBone Green neu
reboot
# Kompiliere AD1938 AudioCard Cape (in diesem Fall mit 8 TDM Slots)
dtc -O dtb -o BBB-AD193X-8TDM-00A0.dtbo -b 0 -@ BBB-AD193X-8TDM-00A0.dts
mv BBB-AD193X-8TDM-00A0.dtbo /lib/firmware
# Lade AD1938 AudioCard Cape mithilfe von Bone-Cape-Manager
sh -c "echo 'BBB-AD193X-8TDM' > /sys/devices/platform/bone_capemgr/slots"
# Kontrolliere mit aplay -l, ob AD1938 AudioCard Treiber erfolgreich
# geladen worden ist.
# (Optional) Um die Audiotreiber beim Systemstart automatisch zu Laden,
# kann der obige Befehl in die Datei /etc/rc.local hinzugefügt werden.
```

Listing 3.8: Anleitung zur Kompilierung und Konfiguration von eigenem BeagleBone Kernel mit AD1938 AudioCard

3.3.4 ALSA Soundkartenkonfiguration

Standardmäßig bietet ALSA nur die Hardwareschnittstelle hw:x,y ohne jegliche Vorkonfiguration an, was bei falscher Anwendung schlechte Klangqualität oder große Latenzen zur Folge haben kann. Weiterhin müssen beim Ansprechen der Hardwareschnittstelle alle Parameter, wie z.B. Abtastrate, PCM-Format, Anzahl der Kanäle, usw. von der Hardware unterstützt werden, wodurch lange Befehle entstehen. Um dies zu vermeiden und Endanwendern die Nutzung der Soundschnittstellen zu erleichtern, bietet ALSA die

systemweite Konfigurationsdatei asound.conf bzw. die benutzerbezogene Konfigurationsdatei .asoundrc an[38]. In der Konfigurationsdatei können virtuelle PCM-Schnittstellen definiert und diverse Parameter vorgegeben werden, wodurch das Ansprechen der Soundschnittstelle vereinfacht wird. Um virtuelle PCM-Schnittstellen zu definieren, bietet ALSA standardmäßig viele verschiedene Plugins[39] an, welche in der Referenz der ALSA C Bibliothek[34] erläutert werden. Insbesondere das Plugin „dmix“[30], welches verschiedene Audiostreams vor der Ausgabe auf einer Hardwareschnittstelle mischt, und “dsnoop“[31], welches das Gleiche für die Aufnahme übernimmt, sind besonders hilfreich. Weiterhin können der Hardware-PCM-Schnittstelle dadurch ALSA-Puffergrößen und Periodengrößen vorgegeben werden, was insbesondere die Latenzen beeinflusst. Da der AD1938 Audiocodec 2 Stereo Eingänge und 4 Stereo Ausgänge besitzt, diese jedoch nicht einzeln über die Hardware-PCM-Schnittstelle angesprochen werden können, wurde mithilfe des Plugins „route“ jeweils eine virtuelle Schnittstelle eingerichtet. So können die Eingänge über die Schnittstellennamen „ADC1“ bis „ADC2“ und die Ausgänge über die Schnittstellennamen „DAC1“ bis „DAC4“ angesprochen werden. Hardwareparameter werden hierbei nicht vorgegeben.

3.3.5 Funktionstest

Für ALSA sind verschiedene Testtools verfügbar, welche bis auf „pcm“ im Debian-Paket „alsa-utils “ enthalten sind.

pcm

Das ALSA-Tool pcm dient zum minimalen Funktionstest einer ALSA-PCM-Schnittstelle, indem es einen einfachen Sinus, dessen Frequenz bestimmt werden kann, generiert. Weiterhin können mithilfe von Übergabeparametern verschiedene Einstellungen, wie Abtastrate und PCM-Format, getestet werden. Allerdings ist das Tool in keinem Paket für Debian enthalten und muss daher selbst kompiliert werden. Es befindet sich in der ALSA-Bibliothek im Unterverzeichnis test/ zusammen mit anderen Testtools. Zu diesem Zeitpunkt wird unter Debian standardmäßig die ALSA-Bibliothek 1.0.25[32] verwendet,

[30] http://www.alsa-project.org/main/index.php/Asoundrc#dmix
[31] http://www.alsa-project.org/main/index.php/Asoundrc#dsnoop
[32] ftp://ftp.alsa-project.org/pub/lib/

weshalb auch die gleiche Version genutzt werden sollte. Es folgt ein kurzes Beispiel zur Ausgabe eines 1 kHz Sinus auf allen 8 Ausgängen:

```
# Ausgabe von 1 kHz Sinus mit S16_LE (Signed 16 Bit Little-Endian)
# PCM-Format und 48 kHz Abtastrate auf allen 8 Ausgängen
./pcm -D hw:0,0 -c 8 -r 48000 -f 1000 -o S16_LE
```

aplay

Das ALSA-Tool aplay dient zur einfachen Wiedergabe von WAV-Dateien. Es bietet weiterhin die Möglichkeit mithilfe von Übergabeparametern verschiedene Stream-Parameter zu konfigurieren. Sollte die als Argument übergebende Abtastrate von der Abtastrate der Audiodatei abweichen, kann das ALSA-Plugin „plug“ genutzt werden, welches sich automatisch um das Resampling kümmert (siehe 6.1.1).

```
# Ausgabe einer WAV-Audiodatei
aplay -D hw:0,0 <WAVFILE>
```

arecord

Das ALSA-Tool arecord ist das Gegenstück zu aplay und dient zur Aufname von Audio. Mit folgendem Befehl, werden alle 4 Audioeingänge aufgenommen und in einer WAV-Datei gespeichert:

```
# Aufnahme aller 4 Eingänge mit S16_LE (Signed 16 Bit Little-Endian)
# PCM-Format und 48 kHz Abtastrate
arecord -D hw:0,0 -c 4 -f S16_LE -r 48000 audioinput.wav
```

speaker-test

Das Tool speaker-test dient dazu einzelne Audioausgänge nacheinander zu testen. Weiterhin kann speaker-test nicht nur einen Sinus generieren, sondern auch Rosa Rauschen[33] und WAV-Dateien abspielen. Mit folgendem Befehl wird nacheinander ein 1 kHz Sinus auf jedem Audioausgang ausgegeben:

[33] Im Gegensatz zu weißem Rauschen, welches alle Frequenzbereiche gleichmäßig abdeckt, wird rosa Rauschen mit steigender Frequenz geringer. Da das menschliche Gehör mit steigender Frequenz sensibler wird, empfinden wie rosa Rauschen als gleich laut.

```
# Aufeinanderfolgende Ausgabe von 1 kHz Sinus mit S16_LE
# (Signed 16 Bit Little-Endian) PCM-Format und
# 48 kHz Abtastrate auf allen 8 Ausgängen
speaker-test -D hw:0,0 -r 48000 -c 8 -f 1000 -F S16_LE -t sine
```

alsamixer

Das ALSA-Tool alsamixer bietet eine intuitive Oberfläche, um verschiedene Hardwareparameter, wie z.B. Lautstärke, Hochpassfilter, De-emphase[34], usw. des Audiocodecs zu konfigurieren.

amixer

Das ALSA-Tool amixer dient, ähnlich wie das Tool alsamixer, der Konfiguration der Soundkarte, bietet jedoch keine Oberfläche. Dennoch offeriert es mehr Möglichkeiten, wie z.B. die Lautstärke in Dezibel einzustellen.

3.4 Debugging

3.4.1 Kernel Logs

Da die Gerätetreiber bzw. Kernel-Module unter Linux innerhalb des Kernel-Kontextes ausgeführt werden, entfallen übliche Debug-Tools. Es ist beispielsweise nicht möglich Breakpoints zu setzen und Variablenwerte auszulesen, da dadurch der Kernel angehalten werden würde und dementsprechend nicht mehr reagiert. Innerhalb dieser Bachelorarbeit wurde die Fehlersuche daher ausschließlich mithilfe von Kernel-Debug-Ausgaben (dev_dbg()) durchgeführt, welche mithilfe von „dmesq“ abgerufen werden können. Weitere Informationen können der Anleitung dynamic-debug-howto.txt[35] entnommen werden. Standardmäßig sind die Debug-Ausgaben unter Linux nicht aktiviert, können jedoch durch Hinzufügen von #define DEBUG 1 in einer Treiber-Quelldatei aktiviert werden.

[34] Absenkung der hohen Frequenzen, um Rauschen zu unterdrücken
[35] `https://www.kernel.org/doc/Documentation/dynamic-debug-howto.txt`

3.4.2 Verifizierung von Registereinträgen

Um Fehler der Konfiguration der PCM-Schnittstelle der CPU oder des Audiocodecs auszuschließen, ist es sinnvoll zur Laufzeit einzelne Registerwerte auszulesen und diese mit dem entsprechenden Datenblatt zu vergleichen. Mithilfe der Debug-Ausgabefunktion dev_dbg() und regmap können innerhalb beliebiger Treiberfunktionen die aktuellen Registereinträge ausgelesen oder auch gesetzt werden. Es folgt ein kurzes Listing, welches demonstrieren soll, wie alle 16 Registereinträge nach dem Setzen von Hardwareparametern des AD1938 Audiocodecs in die Kernel-Logs ausgegeben werden:

```
static int ad193x_hw_params(
  struct snd_pcm_substream *substream,
  struct snd_pcm_hw_params *params,
  struct snd_soc_dai *dai) {
    ...
    for (i=0; i<=16; i++){
      regmap_read(ad193x->regmap, i, &ret);
      dev_dbg(codec->dev, "AD193X register %d:\t0x%x", i, ret);
    }
    ...
  }
```

Listing 3.9: Debug-Ausgabe der Registereinträge von AD1938 Audiocodec mithilfe von regmap

Mit der Eingabe des Befehls „dmesq“ im Terminal können anschließend die Ausgaben angezeigt werden.

3.4.3 Serielle Dekodierung mit Oszilloskop

Das Picoscope 3204D MSO USB-Oszilloskop stellt eine Funktion zur seriellen Dekodierung diverser Protokolle zur Verfügung. Dazu zählen unter anderem SPI, I^2C und I^2S, welche von den Audiocodecs CS4272 und AD1938 genutzt werden. Dies macht insbesondere Sinn, wenn die Software anscheinend funktioniert, jedoch die Hardware nicht das erwartete Verhalten zeigt (siehe z.B. Problem der Taktgenerierung auf BeagleBone Green 5.2).

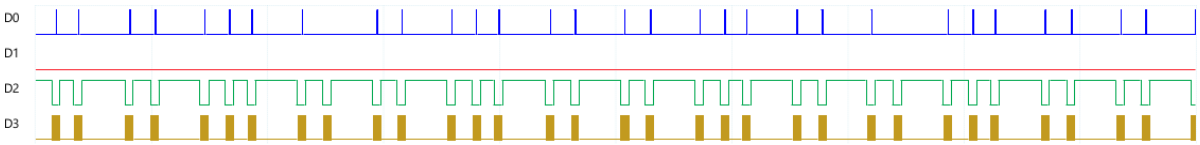

Abbildung 3.4: Serielle Dekodierung von SPI mithilfe von Picoscope 3204D MSO Oszilloskop

Abbildung 3.4 zeigt die Dekodierung der SPI-Übertragung zwischen dem BeagleBone Green und der AD1938 AudioCard, wobei der BeagleBone Green als Busmaster konfiguriert ist. D0 entspricht dem MOSI Signal, D1 dem MISO Signal, D2 dem Chip-Select-Signal und D3 dem Taktsignal (siehe 2.4).

4 Evaluierung vorhandener Platinen

Alle in diesem Kapitel beschriebenen Messprogramme befinden sich in einem Git-Repository der Fachhochschule Kiel.

4.1 Kennwerte und Messverfahren

Die folgenden charakteristischen Kennwerte des Audiosystems wurden auf Basis des Handbuchs „The Data Conversion Handbook“[40, S. 8.60 ff] von Analog Devices recherchiert und berechnet.

4.1.1 Total Harmonic Distortion (THD)

Die Total Harmonic Distortion ist das Verhältnis des quadratischen Mittels der im allgemeinen 5 harmonischen Oberschwingungen zur Grundschwingung im Frequenzband von 20 Hz bis 20 kHz und wird in Dezibel folgendermaßen berechnet:

$THD_{dB} = 20 \cdot \log_{10}(\frac{\sqrt{V_2^2+V_3^2+V_4^2+...+V_n^2}}{V_s})$[41]

Weiterhin gibt es noch die Total Harmonic Distortion + Noise (THD+N), welche zusätzlich zum Verhältnis des quadratischen Mittels der harmonischen Oberschwingungen, das gesamte restliche Signal und damit zusätzlich das Rauschen miteinbezieht. Hieraus ergibt sich folgende Formel:

$THD + N_{dB} = 20 \cdot \log_{10}(\frac{\sqrt{V_2^2+V_3^2+V_4^2+...+V_n^2+V_{noise}^2}}{V_s})$[41]

Messdurchführung

Zunächst müssen die entsprechenden Digital-Analog-Wandler mit den Analog-Digital-Wandlern verbunden werden. Anschließend wird über den Digital-Analog-Wandler für eine Sekunde ein -1dBFS[1] 1 kHz Sinus ausgegeben, mit dem Analog-Digital-Wandler wieder aufgenommen und in einer WAV-Datei abgespeichert. Bei einer Referenzamplitude von 1 ergibt sich beim -1dBFS Signal eine Amplitude von

$A_2 = A_1 \cdot 10^{\frac{G_{dB}}{20}} = 10^{-\frac{1}{20}} = 0,8912509381.$

Um ein möglichst aussagekräftiges Ergebnis zu erhalten, wurde jede Messung 5 mal durchgeführt.

Auswertung

GNU Octave bietet die Funktion wavread() an, welche direkt eine WAV-Datei lesen und als Zahlen darstellen kann. Anschließend wurde mithilfe einer schnellen Fourier-Transformation (FFT) jeweils das Spektrum der entsprechenden WAV-Datei erstellt und anhand der harmonischen Oberschwingungen die THD berechnet. Zuletzt wird von den 5 ermittelten Werten der Mittelwert und die Standardabweichung gebildet.

4.1.2 Dynamic Range (DNR)

Der Dynamikumfang eines Audiosystems gibt den nutzbaren Bereich an, in dem sich der Pegel des Audiosignals bewegen kann. Er wird durch das Verhältnis vom maximalen Pegel zum quadratischen Mittel des restlichen Signals und Anwendung eines Logarithmus berechnet. Im Gegensatz zum Signal-Rausch-Verhältnis, welches durch die absoluten Werte berechnet wird, wird der Dynamikumfang durch das Verhältnis vom maximalen Pegel und dem quadratischen Mittel berechnet.

$DNR_{dB} = 20 \cdot \log_{10}(\frac{V_S}{V_{noise,rms}})$

[1] entspricht -1 dB von vollausgesteuertem Signal

Messdurchführung

Der Dynamikumfang wird nach dem gleichen Verfahren gemessen wie die Total Harmonic Distortion.

Auswertung

Die Auswertung der aufgenommenen WAV-Dateien wurde ebenso durch Erstellen des Spektrums mithilfe einer FFT und dem Verhältnis des maximalen Pegels und des quadratischen Mittels des restlichen Signals durchgeführt. Anschließend wird anhand der 5 ermittelten Werte der Mittelwert und die Standardabweichung berechnet.

4.1.3 Übersprechen (Crosstalk)

Ein großes Problem der Signal- bzw. Datenübertragung mithilfe von elektrischen Signalen besteht darin, dass elektrische Wechselströme elektromagnetische Felder erzeugen, die in benachbarten Bauteilen oder Signalleitungen elektrische Energie induzieren. Im Bereich der Audioübertragung macht sich dies durch unerwünschte Geräusche bemerkbar. Im Idealfall sollten sich die unterschiedlichen Signale gegenseitig nicht beeinflussen und voneinander isoliert sein. Das Übersprechen gibt das Verhältnis des eigentlichen Signals zum induzierten Signal in der benachbarten Signalleitung an und ist daher möglichst stark zu minimieren. Entscheidend ist hier ein sorgfältiges Design der Schaltung des Audio-Interfaces und des Layouts der gedruckten Schaltung.

Messdurchführung

Das Übersprechen wurde gemessen indem zuerst die Digital-Analog-Wandler mit den Analog-Digital-Wandlern verbunden wurden. Anschließend wurde für eine Sekunde ein -1dBFS 1 kHz Sinus auf dem linken Kanal des Digital-Analog-Wandlers ausgegeben und über den Analog-Digital-Wandler beide Kanäle wieder aufgenommen. Aufgrund mangelnder Zeit wurde die Messung jeweils nur einmal durchgeführt.

Auswertung

Die Auswertung der Audiosignale wurde mithilfe von GNU Octave durchgeführt. Durch eine FFT wurden die Spektren von dem Ursprungssignal und dem benachbartem Signal berechnet. Der Abstand der Amplituden der Grundschwingung ergibt das Übersprechen in dB.

4.1.4 Frequenzgang

Der Frequenzgang sollte bei Audiosystemen in einem Bereich von 20 Hz bis 20 kHz möglichst linear verlaufen und damit alle Frequenzen gleichstark aussteuern. Das Nyquist-Shannon-Abtasttheorem besagt, dass ein Signal mit mindestens doppelt so hoher Frequenz abgetastet werden muss, um es wieder rekonstruieren zu können. Daher kann abhängig von der Abtastrate nur eine Nutzsignalfrequenz bis f/2 gemessen werden.

Messdurchführung

Die Frequenzgänge wurden abhängig von der Abtastrate bis zur maximal möglichen Frequenz - 1000 Hz gemessen, da durch einen Sinus mit der halben Frequenz der Abtastrate ein Nullsignal entstehen würde. Die Ermittlung des Frequenzgangs wurde durch die aufeinander folgende Ausgabe eines vollausgesteuertem Sinus über die DACs und wieder Aufnahme über die ADCs realisiert. Aufgrund mangelnder Zeit wurde die Messung jeweils nur einmal durchgeführt.

Auswertung

Die Auswertung wurde auf Basis des MATLAB-Projekts „Measurement of Loudspeaker Frequency Response with Matlab Implementation“[42] durchgeführt. Dieses erstellt durch eine FFT und dem Verhältnis der Ausgangs- und Eingangsamplitude das Frequenzspektrum.

4.1.5 Latenz

Die Latenz ist im Gegensatz zu den oben genannten Kenngrößen nicht nur von der Audioplatine, sondern auch von der verwendeten Hardwareplattform abhängig. Die Latenztests müssen daher für jede einzelne Hardwareplattform durchgeführt werden. Weiterhin ist zu beachten, dass die aktuelle Rechenlast, insbesondere das Abarbeiten von Interrupts, einen großen Einfluss auf die Latenz hat.

Round Trip Time (RTT)

Die Paketumlaufzeit oder auch RTT gibt die Zeitdifferenz an, die ein Paket vom Start zum Ziel in einem digitalen System benötigt und ist somit eine gute Messgröße, um Latenzen zu analysieren und zu vergleichen.

Messdurchführung

Das Messverfahren wurde in diesem Zusammenhang mithilfe eines USB-Oszilloskops (Picoscope 3204D MSO) mit integriertem Funktionsgenerator durchgeführt. Ein Kanal des Oszilloskops wird hierbei mit dem Eingang des Analog-Digital-Wandlers und der andere Kanal mit dem Ausgang des Digital-Analog-Wandlers verbunden. Der Trigger-Modus des Oszilloskops muss hierbei auf einzeln und den Kanal am Analog-Digital-Wandler eingestellt werden. Die Auflösung der Zeitachse des Oszilloskops sollte anfangs im unteren Millisekunden Bereich, wie z.B. 5 ms, liegen. Wichtig ist, dass man die Änderung am Ausgang des Digital-Analog-Wandlers sieht. Über das ALSA-Tool „alsaloop“[2], welches sich im Debian Paket „alsa-utils“ befindet, wird ein Soundkarteneingang mit einem Soundkartenausgang verbunden. Der Datentransfer wird hierbei über die CPU bzw. DMA (Direct Memory Access) und nicht etwa im Audio Codec intern geregelt, wodurch sich ein recht realistisches Szenario ergibt, da im Anwendungsfall die Audiodaten vor der Ausgabe erst über die CPU verarbeitet werden. Zu Beginn des Tests generiert der Funktionsgenerator einen einzelnen Rechteckimpuls am Eingang des Analog-Digital-Wandlers. Sobald dieser das Oszilloskop erreicht, wird die Messung gestartet bzw. getriggert. Anschließend kann über die Werteskala die Zeitdifferenz zwischen Eingangs- und Ausgangsimpuls abgelesen werden, welche der Round-Trip-Time entspricht.

[2] `http://manpages.ubuntu.com/manpages/precise/man1/alsaloop.1.html`

ALSA Latenz-Testtool

Das Testtool „latency“ befindet sich in der ALSA Bibliothek. Das Tool dient dazu, die minimale Latenz bzw. die optimale Puffergröße und Periodengröße, ohne Auftreten eines Pufferüberlaufs oder Pufferunterlaufs (XRUN), zu ermitteln. Über die Übergabeparameter, wie z.B. Abtastrate, Samplingtiefe, Audiokanäle, usw. können so unterschiedliche Anwendungsfälle getestet werden. Das Tool fängt hierbei mit einer minimal vorgegebenen Latenz oder Puffergröße an und vergrößert diese in regelmäßigen Schritten, bis der Test für eine bestimmte Zeit ohne Pufferüberlauf oder Pufferunterlauf erfolgreich durchgeführt wurde.

4.1.6 Automatisierter Test

Um die Evaluierung zu erleichtern und Zeit zu sparen, wurde ein automatisierter Test (automated-test.sh) mithilfe von Bash erstellt, welcher die oben genannten Messungen nacheinander durchführt. Anfangs wurden die Berechnungen mithilfe von MATLAB durchgeführt, da es hierfür extra Funktionen zur Berechnung der Total Harmonic Distortion und Signal-to-Noise-Ratio gibt. Da MATLAB allerdings eine properitäre Software und sehr teuer ist, wurden die Rechnungen manuell mithilfe von GNU Octave implementiert. Dadurch bleiben alle verwendeten Werkzeuge in der Thesis Open Source, so dass jeder die Berechnungen kostenlos durchführen kann. Als Informationsquelle für die Messung der Total Harmonic Distortion und des Dynamikumfangs diente das MATLAB Skript von RF William Hollender.

Ablauf

Der Automatisierte Test setzt sich aus folgenden Einzeltests zusammen:

1. Latenztest mithilfe des ALSA Testtools „latency“
2. THD, THD+N und DNR Berechnung mithilfe von GNU Octave
3. Crosstalk Berechnung mithilfe von GNU Octave
4. Frequenzgang mithilfe von GNU Octave

Bei den Punkten 2. bis 4. müssen, wie oben bei den Messverfahren beschrieben, die entsprechenden DACs mit den ADCs verbunden werden. Um die Vorbereitung des Tests zu erleichtern, wurde weiterhin das Bash-Skript „automated-test-preparation.sh“ erstellt, welches alle benötigten Softwarepakete, insbesondere GNU Octave in der Version 3.8 und die dazugehörigen Pakete, und deren Abhängigkeiten installiert. Da für die Berechnungen mindestens die Version 3.8.x von GNU Octave benötigt wird, welche in den stabilen Paketquellen von Debian nicht enthalten ist, muss in der Datei /etc/apt/sources.list die Zeile
„deb http://ftp.debian.org/debian wheezy-backports main contrib non-free“
auskommentiert werden. Der automatisierte Test selbst wird durch Ausführen des Bash Skripts „automated-test.sh“ gestartet. Als Parameter werden jeweils eine PCM-Schnittstelle zur Ein- und Ausgabe, der Soundkartenname für Plots und, falls ein einzelner Test ausgeführt werden soll, der Name des entsprechenden Tests (latency, thd, crosstalk oder frequency-response) erwartet. Die Hardware-PCM-Schnittstelle wird automatisch von ALSA mit dem Anfangskürzel „hw:“ plus dem Soundkartenindex und dem Geräteindex benannt. Der Soundkartenindex und Geräteindex kann durch Ausführen des Befehls „aplay -l“ ermittelt werden. Es folgt eine kurze Beispielausgabe auf dem BeagleBone Green:

```
# Ermittelung des Soundkarten- und Geräteindexes
debian@beaglebone:~$ aplay -l
**** List of PLAYBACK Hardware Devices ****
card 1: Card [AD193X Card], device 0: AudioCard TDM ad193x-hifi-0 []
Subdevices: 1/1
Subdevice #0: subdevice #0
```

Listing 4.1: Ermittlung des Index der Soundkarte

Somit kann die Soundkarte bzw. die PCM-Schnittstelle über „hw:1,0“ angesprochen werden. Zu beachten ist, dass es sich hierbei um die richtige Hardwareschnittstelle handelt und somit nur Parameter, wie z.B. Abtastrate, funktionieren, die die Soundkarte wirklich unterstützt.

4.2 Evaluierung von SuperAudioBoard, AD1938 AudioCard Rev.0 und AD1938 AudioCard Rev.1

4.2.1 THD, THD+N und DNR

Abtastrate / PCM-Format	**SuperAudioBoard**	**AD1938 AudioCard Rev.0**
44,1 kHz S16_LE	N.A.	THD: -74.8 dB, σ : 4.55 THD+N: -25.3 dB, σ : 0.08 DNR: 68.3 dB, σ : 0.08
44,1 kHz S32_LE	N.A.	THD: -79.2 dB, σ : 5.73 THD+N: -25.3 dB, σ : 0.07 DNR: 68.3 dB, σ : 0.07
48 kHz S16_LE	THD: -83.7 dB, σ : 0.36 THD+N: -82.8 dB, σ : 0.32 DNR: 133.3 dB, σ : 0.10	THD: -91.4 dB, σ : 0.55 THD+N: -81.3 dB, σ : 0.36 DNR: 124.8 dB, σ : 0.44
48 kHz S32_LE	THD: -83.7 dB, σ : 0.23 THD+N: -83.4 dB, σ : 0.22 DNR: 137.2 dB, σ : 0.19	THD: -91.5 dB, σ : 0.41 THD+N: -85.1 dB, σ : 1.14 DNR: 129.2 dB, σ : 1.47
96 kHz S16_LE	THD: -83.9 dB, σ : 0.16 THD+N: -83.4 dB, σ : 0.15 DNR: 135.8 dB, σ : 0.09	THD: -91.0 dB, σ : 0.49 THD+N: -84.6 dB, σ : 0.70 DNR: 128.8 dB, σ : 0.82
96 kHz S32_LE	THD: -83.9 dB, σ : 0.17 THD+N: -83.5 dB, σ : 0.17 DNR: 136.9 dB, σ : 0.10	THD: -90.7 dB, σ : 0.38 THD+N: -85.9 dB, σ : 0.51 DNR: 130.7 dB, σ : 0.80
192 kHz S16_LE	THD: -83.9 dB, σ : 0.31 THD+N: -83.4 dB, σ : 0.32 DNR: 136.7 dB, σ : 0.39	THD: -91.2 dB, σ : 0.29 THD+N: -84.4 dB, σ : 0.49 DNR: 128.4 dB, σ : 0.63
192 kHz S32_LE	THD: -83.9 dB, σ : 0.16 THD+N: -83.5 dB, σ : 0.16 DNR: 136.6 dB, σ : 0.13	THD: -91.4 dB, σ : 0.44 THD+N: -84.1 dB, σ : 0.92 DNR: 128.2 dB, σ : 1.22

Tabelle 4.1: Vergleich THD, THD+N und DNR von SuperAudioBoard und AD1938 AudioCard Rev.0

Da die AD1938 AudioCard Rev.0 einen Designfehler im Schaltplan hat, wodurch das -1dBFS Signal gedämpfter ausgegeben wird, ergeben sich bei der THD weitaus bessere Messwerte als beim SuperAudioBoard. Am gemessenen Frequenzgang (Abbildung 4.9) kann man die stärkere Dämpfung gut erkennen. Dies wurde in der nächsten Revision der AD1938 AudioCard entsprechend geändert (siehe Tabelle 4.2).

Abtastrate / PCM-Format	**AD1938 AudioCard Rev.1 (DAC1-ADC1)**	**AD1938 AudioCard Rev.1 (DAC2-ADC2)**
44,1 kHz S16_LE	THD: -73.3 dB, σ : 2.07 THD+N: -25.4 dB, σ : 0.06 DNR: 68.4 dB, σ : 0.06	THD: -72.5 dB, σ : 2.68 THD+N: -25.3 dB, σ : 0.07 DNR: 68.4 dB, σ : 0.07
44,1 kHz S32_LE	THD: -75.5 dB, σ : 3.75 THD+N: -25.3 dB, σ : 0.08 DNR: 68.3 dB, σ : 0.08	THD: -71.1 dB, σ : 2.03 THD+N: -25.3 dB, σ : 0.06 DNR: 68.3 dB, σ : 0.06
48 kHz S16_LE	THD: -83.6 dB, σ : 0.07 THD+N: -80.1 dB, σ : 0.40 DNR: 125.6 dB, σ : 0.66	THD: -74.8 dB, σ : 0.004 THD+N: -73.8 dB, σ : 0.12 DNR: 123.5 dB, σ : 0.54
48 kHz S32_LE	THD: -83.5 dB, σ : 0.11 THD+N: -81.6 dB, σ : 0.28 DNR: 129.3 dB, σ : 0.75	THD: -74.8 dB, σ : 0.004 THD+N: -74.1 dB, σ : 0.1 DNR: 125.2 dB, σ : 0.62
96 kHz S16_LE	THD: -83.4 dB, σ : 0.09 THD+N: -80.8 dB, σ : 0.38 DNR: 127.4 dB, σ : 0.88	THD: -74.8 dB, σ : 0.008 THD+N: -73.8 dB, σ : 0.15 DNR: 123.9 dB, σ : 0.73
96 kHz S32_LE	THD: -83.4 dB, σ : 0.08 THD+N: -81.1 dB, σ : 0.19 DNR: 127.9 dB, σ : 0.51	THD: -74.8 dB, σ : 0.008 THD+N: -73.8 dB, σ : 0.17 DNR: 123.9 dB, σ : 0.75
192 kHz S16_LE	THD: -82.8 dB, σ : 0.05 THD+N: 80.6 dB, σ : 0.84 DNR: 128.3 dB, σ : 3.28	N.A.
192 kHz S32_LE	THD: -83.0 dB, σ : 0.09 THD+N: -80.2 dB, σ : 0.32 DNR: 126.4 dB, σ : 0, 74	N.A.

Tabelle 4.2: THD, THD+N und DNR von AD1938 AudioCard Rev.1

Da die Weiterentwicklung bzw. Verbesserung der AD1938 AudioCard unter anderem die Selektion geeigneter Bauteile beinhaltet, wurden zu Evaluationszwecken bei den ADCs bzw. DACs der AD1938 AudioCard Rev.1 unterschiedliche Operationsverstärker verwendet. ADC2 und DAC2 nutzen jeweils einen kostengünstigeren, wodurch die schlechteren THD und THD+N Messwerte aus Tabelle 4.2 entstehen. In der nächsten Revision der AD1938 AudioCard wird dies entsprechend geändert.

Fazit

Laut dem „Data Conversion Handbook“[40, S. 8.62, Figure. 8.64] sollte die THD+N für Computer-Audio bei einer Abtastrate von 48 kHz im Bereich von -90 dB bis -80 dB liegen, was bei beiden Versionen der AD1938 AudioCard erreicht wurde. Allerdings weicht die minimal ermittelte THD+N mit -81.6 dB der AD1938 AudioCard Rev.1 noch stark von dem angegebenen Wert (-94 dB) aus dem Datenblatt vom AD1938 Audiocodec ab[5]. Die THD+N und das Signal-Rausch-Verhältnis, welches ungefähr dem Dynamikumfang entspricht, sollte im professionellen Studiobereich sogar geringer als -100 dB bzw. größer als 100 dB sein[40, S. 8.66]. Da bei der AD1938 AudioCard Rev.0 jeweils nur 2 Operationsverstärker verwendet wurden, wodurch weniger elektrische Störungen in benachbarten Bauteile induziert wird, ist die THD+N um ca. 3 dB geringer. Die geringe Verschlechterung der THD+N gegenüber der THD lässt sich mit Wahrscheinlichkeit auf noch nicht verwendete Isolatorschaltungen zur Trennung der elektrischen Signale des BeagleBone Green und der AD1938 AudioCard Rev.1 zurückführen.

4.2.2 Crosstalk

Abtastrate / PCM-Format	**SuperAudio-Board**	**AD1938 AudioCard Rev.0**	**AD1938 AudioCard Rev.1**
44,1 kHz S32_LE	N.A.	76.099425 dB	100.861060 dB
48 kHz S32_LE	112.701884 dB	88.730533 dB	103.539178 dB
96 kHz S32_LE	113.184898 dB	88.180910 dB	103.332601 dB
192 kHz S32_LE	112.711722 dB	88.427720 dB	103.373644 dB

Tabelle 4.3: Vergleich Crosstalk von SuperAudioBoard, AD1938 AudioCard Rev.0 und AD1937 AudioCard Rev.1

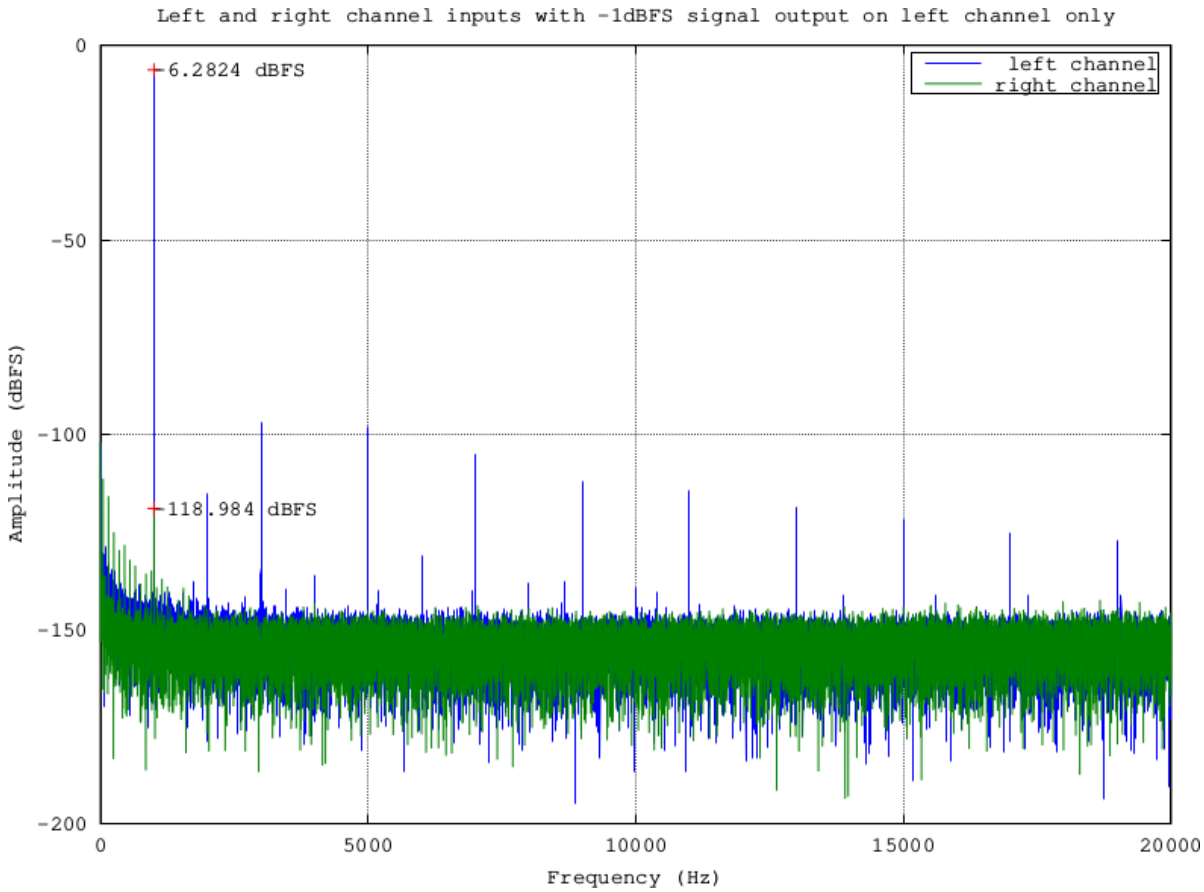

Abbildung 4.4: SuperAudioBoard Crosstalk mit 48 kHz Abtastrate

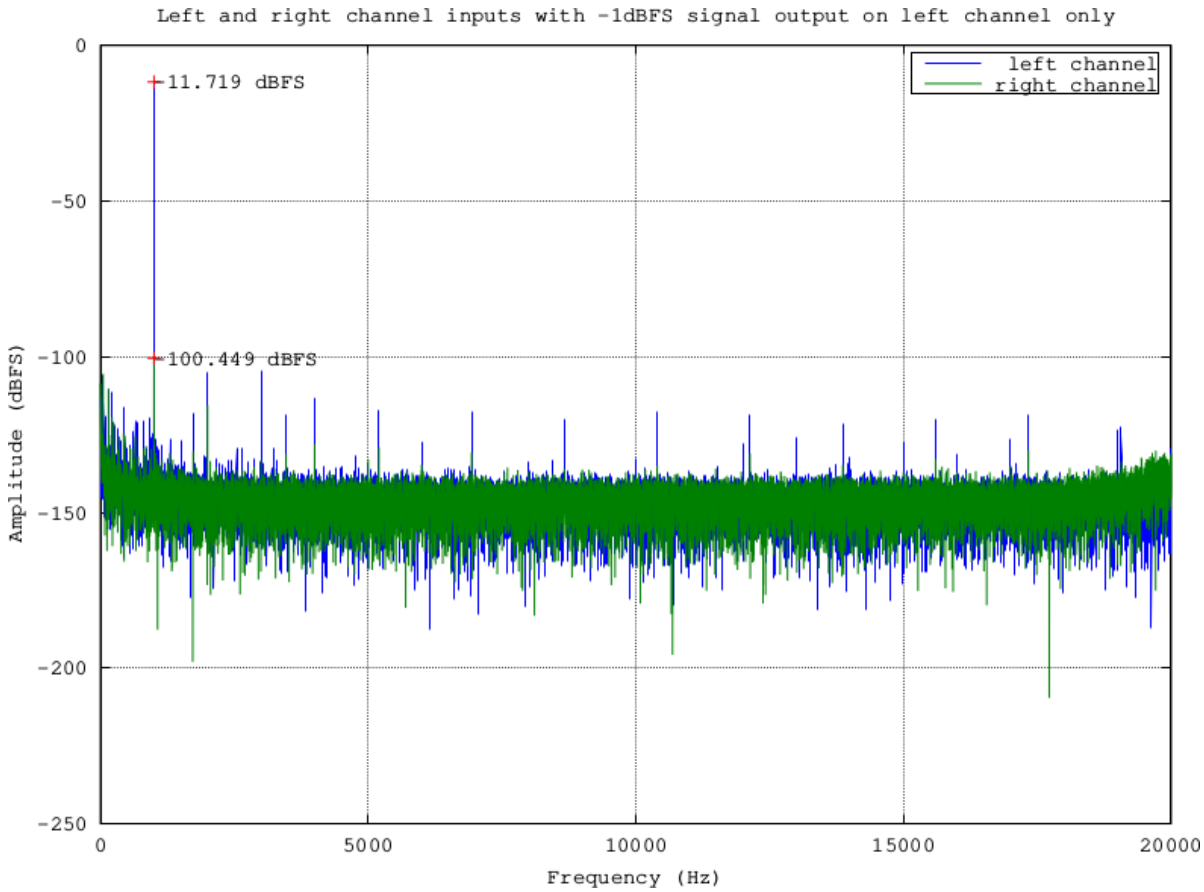

Abbildung 4.5: AD1938 AudioCard Rev.0 Crosstalk mit 48 kHz Abtastrate

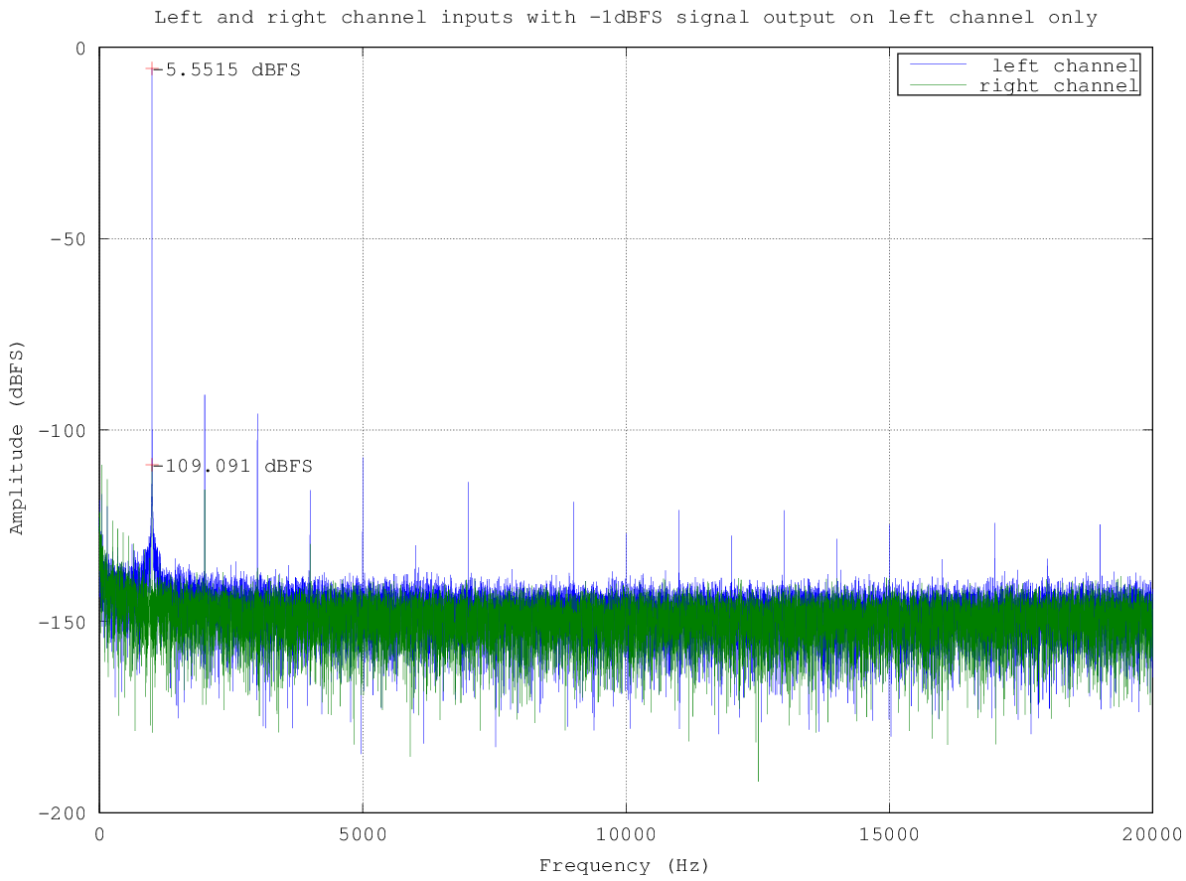

Abbildung 4.6: AD1938 AudioCard Rev.1 Crosstalk mit 48 kHz Abtastrate

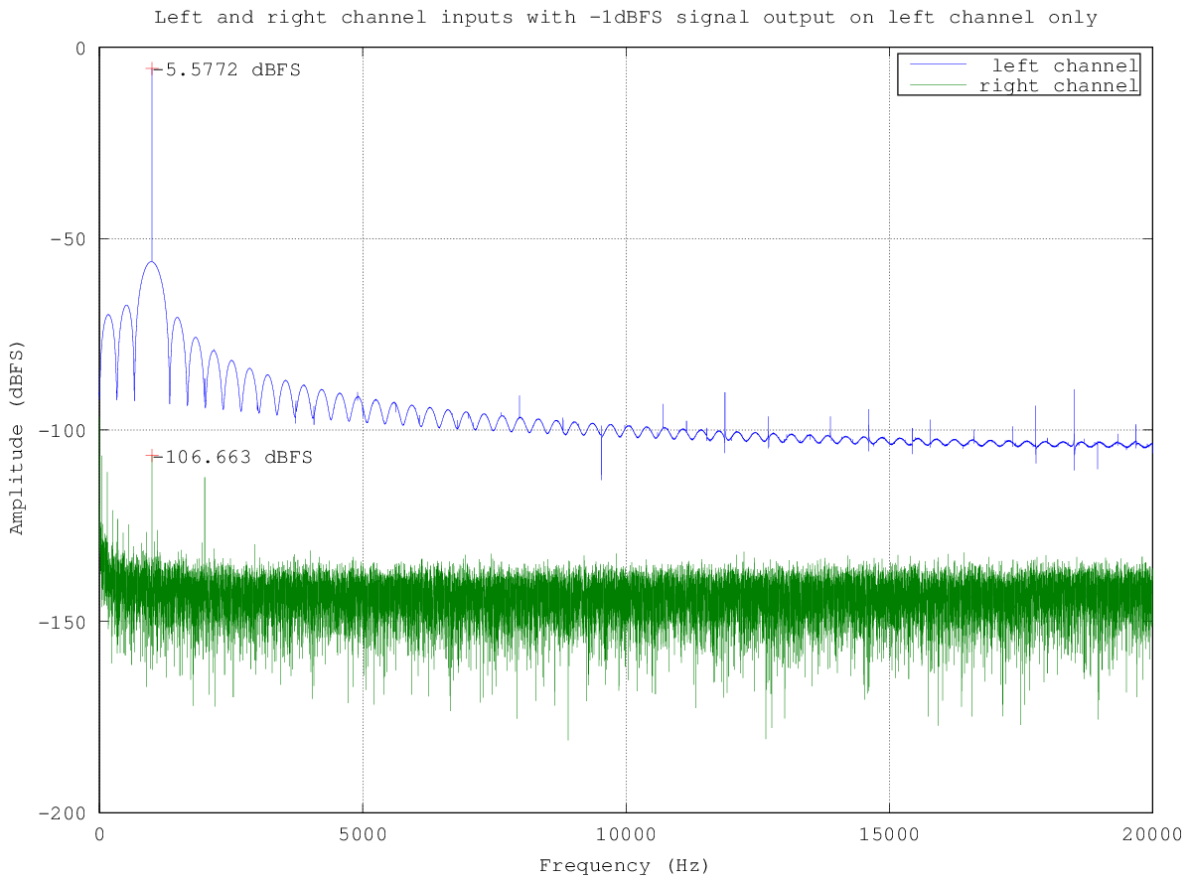

Abbildung 4.7: AD1938 AudioCard Rev.1 Crosstalk mit 44,1 kHz Resampling

Fazit

Im Vergleich zum SuperAudioBoard haben sich bei der AD1938 AudioCard Rev.0 mit einer Differenz von 23.971311 dB weitaus schlechtere Resultate beim Übersprechen ergeben. Die Revision 1 der AD1938 AudioCard wurde entsprechend optimiert und erreicht ca. 15 dB mehr.

4.2.3 Frequenzgang

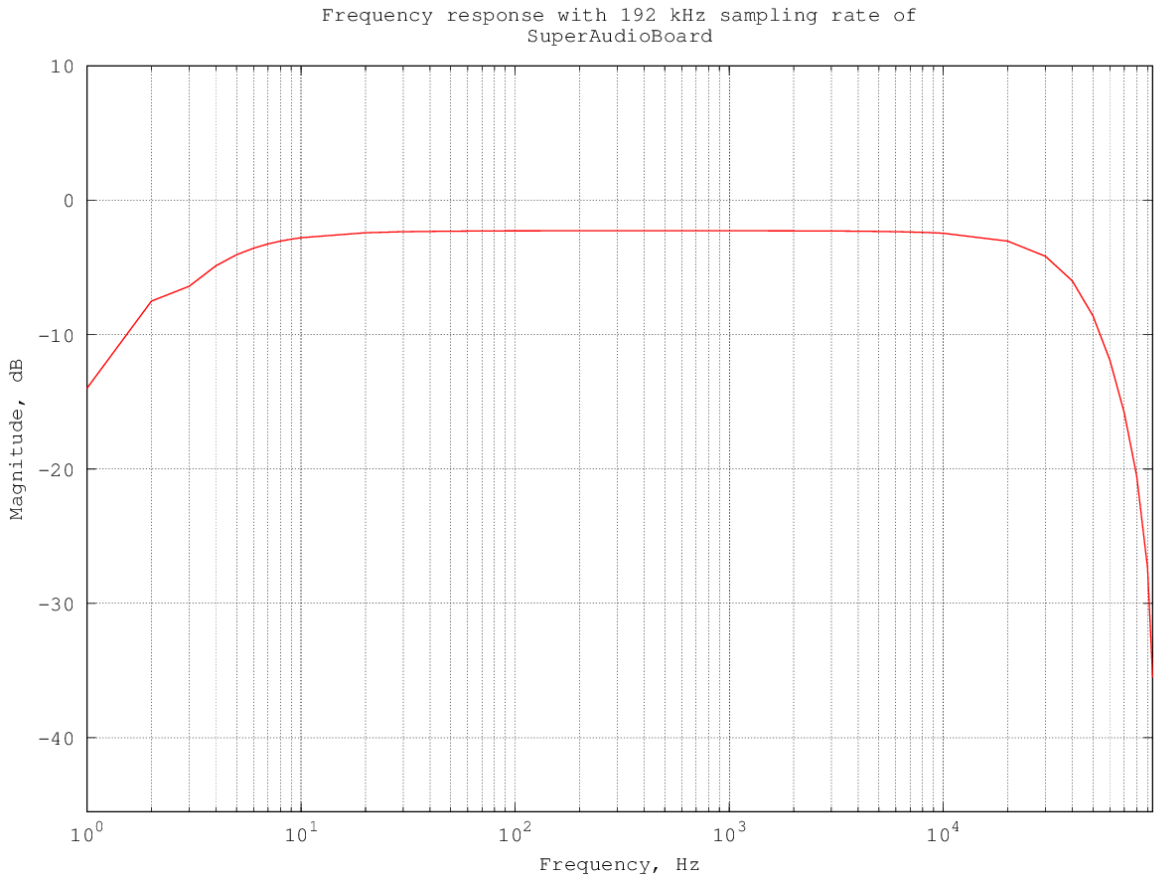

Abbildung 4.8: Frequenzgang von SuperAudioBoard mit 192 kHz Abtastrate (ohne De-emphasis)

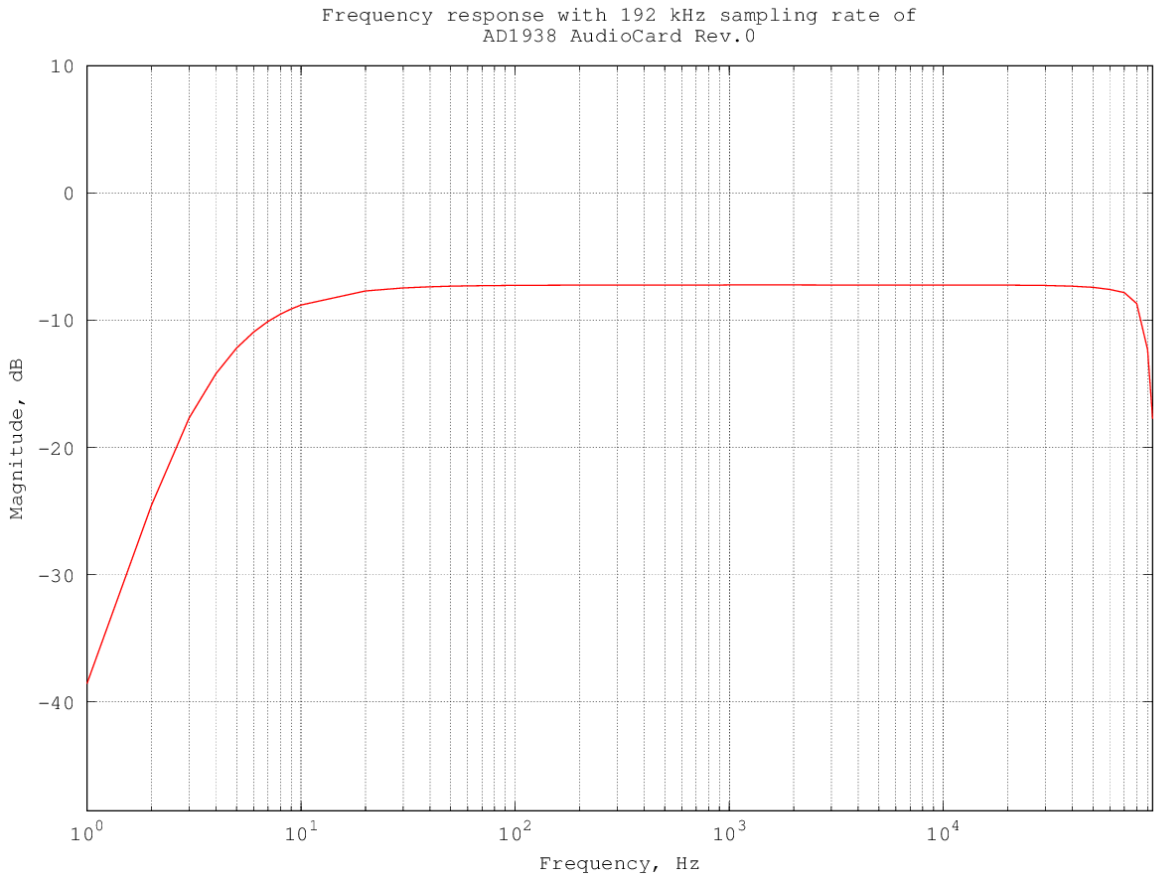

Abbildung 4.9: Frequenzgang von AD1938 AudioCard Rev.0 mit 192 kHz Abtastrate (ohne De-emphasis)

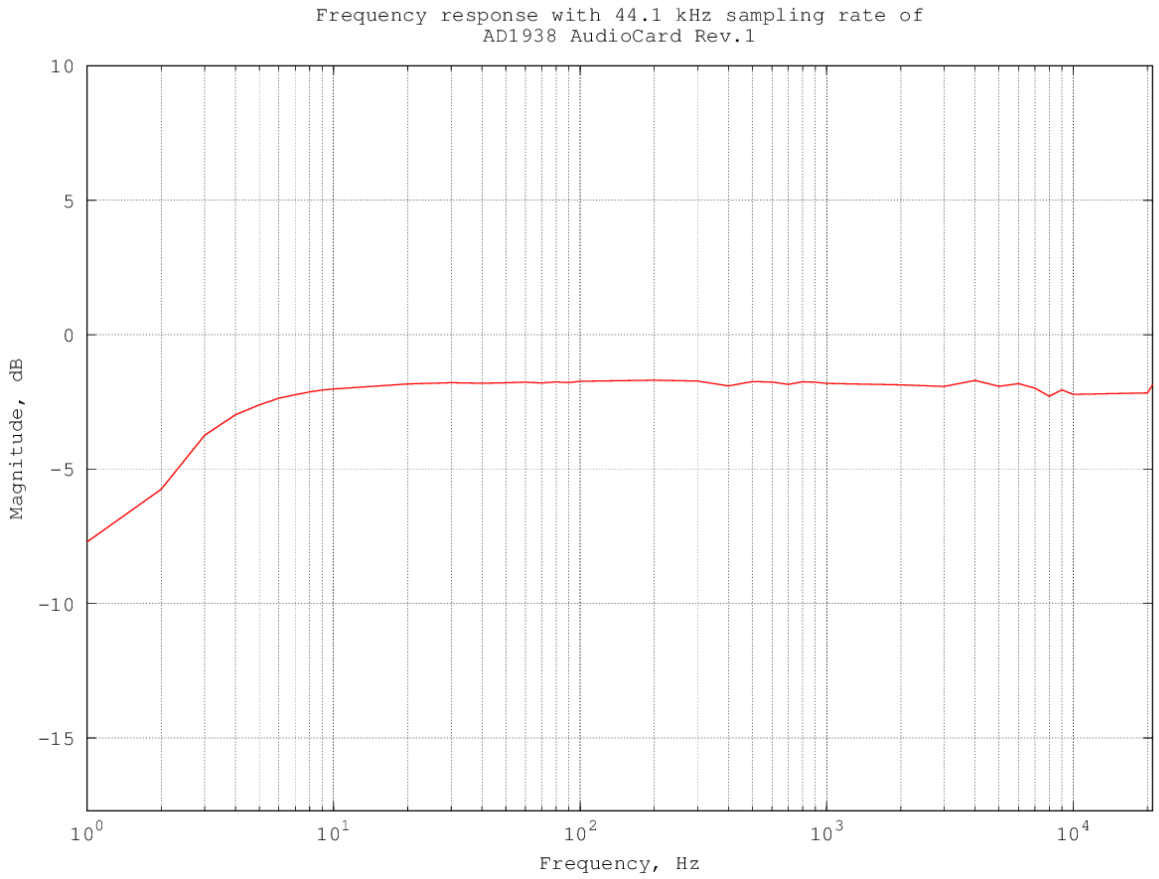

Abbildung 4.10: Frequenzgang von AD1938 AudioCard Rev.1 mit 44,1 kHz Abtastrate (ohne De-emphasis)

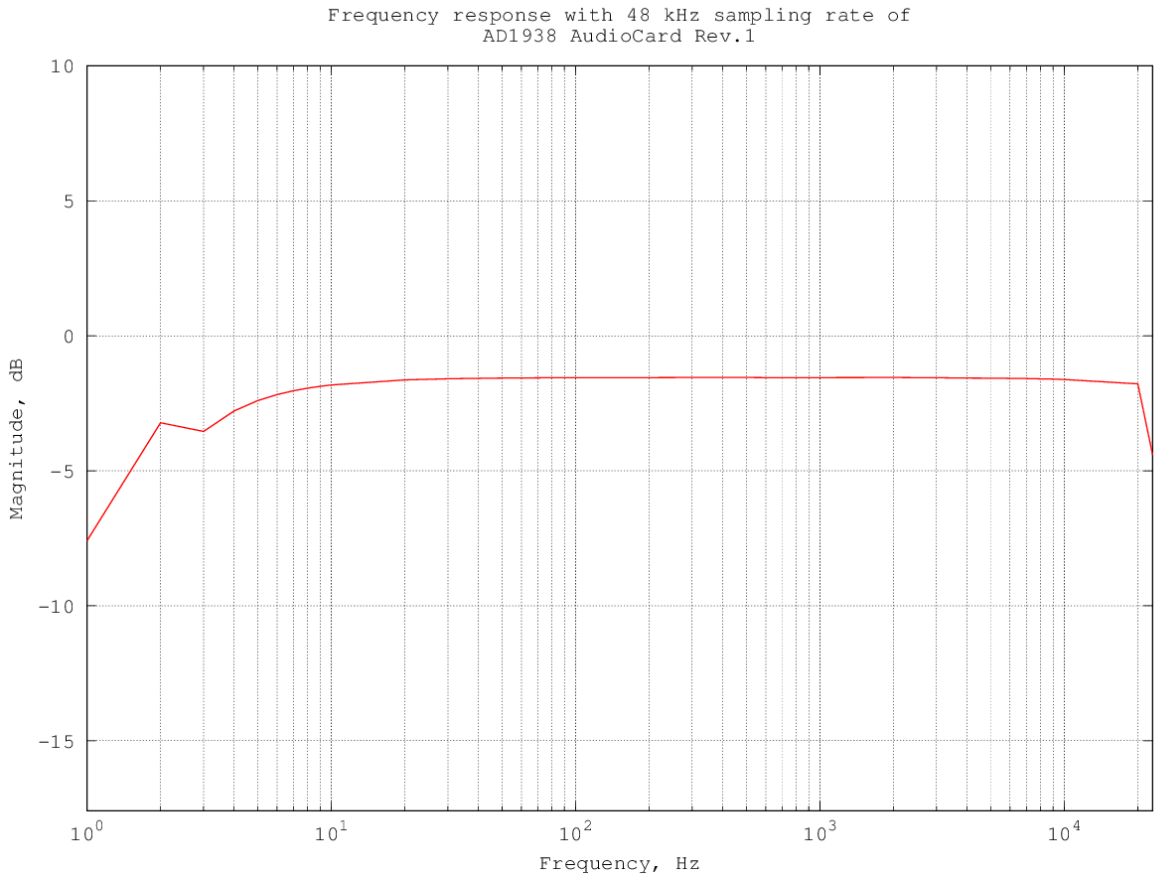

Abbildung 4.11: Frequenzgang von AD1938 AudioCard Rev.1 mit 48 kHz Abtastrate (ohne De-emphasis)

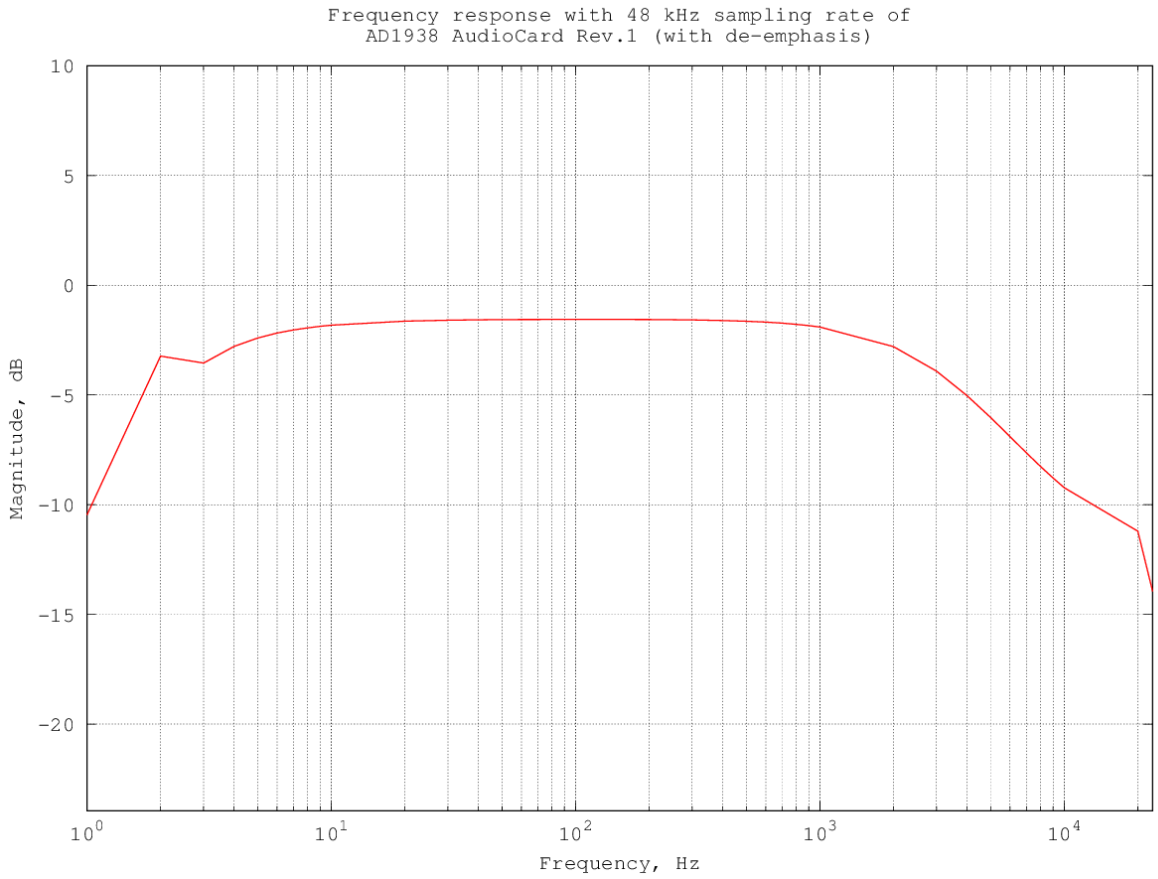

Abbildung 4.12: Frequenzgang von AD1938 AudioCard Rev.1 mit 48 kHz Abtastrate (mit De-emphasis)

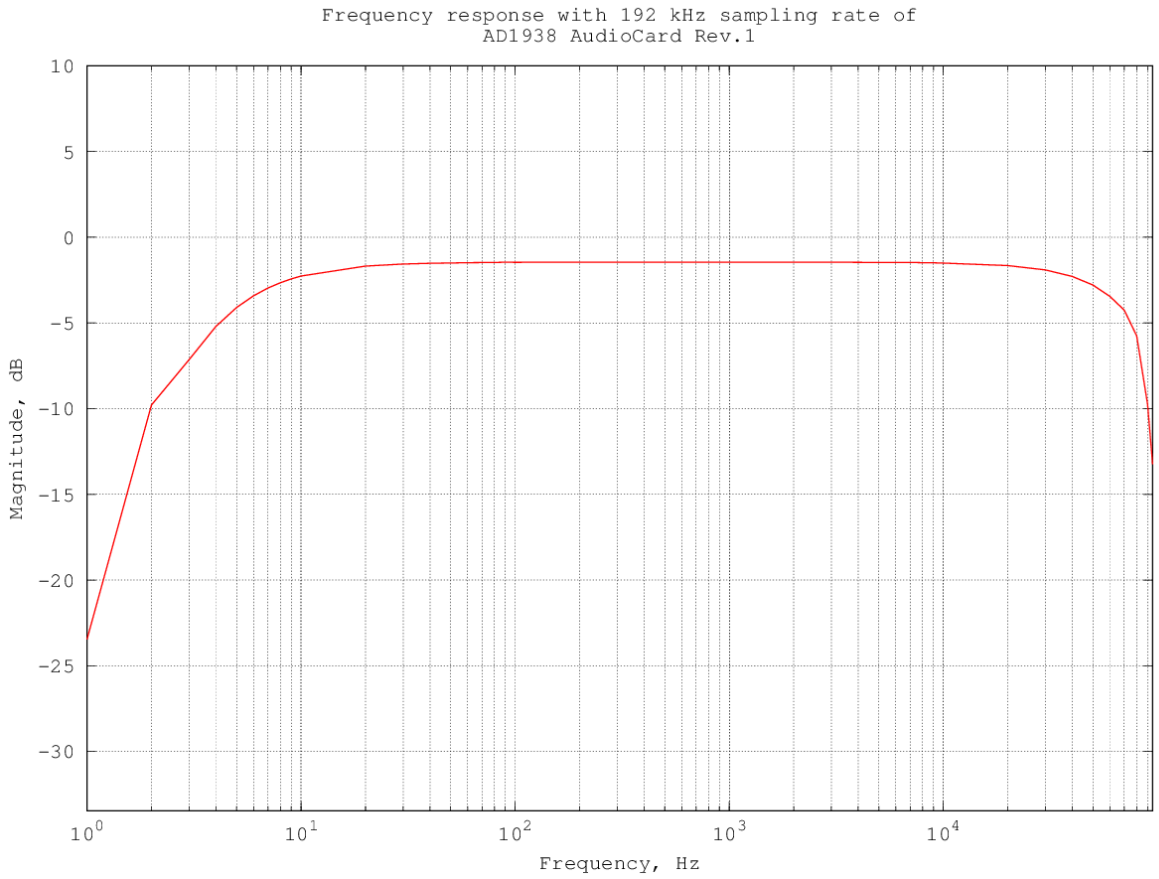

Abbildung 4.13: Frequenzgang von AD1938 AudioCard Rev.1 mit 192 kHz Abtastrate (ohne De-emphasis)

Fazit

Aus den obigen Abbildungen der AD1938 AudioCard Rev.1 kann man entnehmen, dass der Frequenzgang, wie erhofft, sehr linear verläuft. Im Vergleich zur älteren Revision 0 ist der Pegel des vollausgesteuertem Signals durch die Optimierung um ca. 6 dB gestiegen und mit ca. -2 dB vergleichbar mit dem SuperAudioBoard. Weiterhin wird das Signal der AD1938 AudioCard Rev.1, ähnlich dem SuperAudioBoard, bei einer Frequenz über 20 kHz zunehmend gedämpft.

4.2.4 Latenz

Um einen aussagekräftigen Vergleich zwischen dem Raspberry Pi 2 Model B und dem BeagleBone Green zu ermitteln und der Raspberry Pi maximal nur 2 Audiokanäle unterstützt, wurde die Latenz zuerst nur mit 2 Audiokanälen getestet. Um die Rechenlast zu simulieren wurde das Tool „stress“[43] genutzt, welches in den offiziellen Debian Paketen enthalten ist. Das Tool wurde mit folgenden Argumenten ausgeführt:

```
$ stress --cpu 4 --io 2 --vm 2 --vm-bytes 128M --hdd 4 --hdd-bytes 64M
```

Folgend eine kurze Beschreibung der Argumente:

Argument	**Beschreibung**
–cpu 4	4 Prozesse, die in einer Dauerschleife mit Wurzel ziehen (sqrt()) beschäftigt sind.
–io 2	2 Prozesse, die in einer Dauerschleife gepufferte Daten in Speicher mit SD-Karte synchronisieren (sync()).
–vm 2 –vm-bytes 128M	2 Prozesse, die in einer Dauerschleife Speicher allozieren (malloc()) und wieder freigeben (free()).
–hdd 4 –hdd-bytes 64M	4 Prozesse, die in einer Dauerschleife Testdaten auf SD-Karte schreiben (write()) und wieder löschen (unlink()).

Tabelle 4.14: Simulation von Rechenlast mithilfe des Tools „stress“

Die durch das ALSA-Latenz-Testtool ermittelte Latenz entspricht der Round-Trip-Time und wird folgendermaßen berechnet:

$$T_{RTT} = T_{Abtastrate} \cdot N_{Puffergröße} = \frac{1}{f_s} \cdot N_{Puffergröße}$$

Testbedingungen	**Raspberry Pi 2**	**BeagleBone Green**
Abtastrate: 48 kHz Kanäle: 2 PCM-Format: S32_LE Linux-Kernel: 4.1 Ohne Stress	Periode: 28 Frames Puffergröße: 56 Frames Latenz: 1.166667 ms	Periode: 68 Frames Puffergröße: 136 Frames Latenz: 2.833333 ms
Abtastrate: 48 kHz Kanäle: 2 PCM-Format: S32_LE Linux-Kernel: 4.1 Mit Stress	Periode: 92 Frames Puffergröße: 184 Frames Latenz: 3,833333 ms	Periode: 2720 Frames Puffergröße: 5440 Frames Latenz: 113.333333 ms
Abtastrate: 48 kHz Kanäle: 2 PCM-Format: S32_LE Linux-Kernel: 4.1-RT Ohne Stress	Periode: 60 Frames Puffergröße: 120 Frames Latenz: 2.5 ms	Periode: 104 Frames Puffergröße: 208 Frames Latenz: 4.333333 ms
Abtastrate: 48 kHz Kanäle: 2 PCM-Format: S32_LE Linux-Kernel: 4.1-RT Mit Stress	Periode: 144 Frames Puffergröße: 288 Frames Latenz: 6 ms	Periode: 2760 Frames Puffergröße: 5520 Frames Latenz: 115 ms
Abtastrate: 192 kHz Kanäle: 2 PCM-Format: S32_LE Linux-Kernel: 4.1 Ohne Stress	Periode: 64 Frames Puffergröße: 128 Frames Latenz: 0,6666666 ms	Periode: 213 Frames Puffergröße: 426 Frames Latenz: 2.21875 ms
Abtastrate: 192 kHz Kanäle: 2 PCM-Format: S32_LE Linux-Kernel: 4.1-RT Ohne Stress	Periode: 132 Frames Puffergröße: 264 Frames Latenz: 1.375000 ms	Periode: 364 Frames Puffergröße: 728 Frames Latenz: 3.791667 ms

Tabelle 4.15: Latenzvergleich von Raspberry Pi 2 und BeagleBone Green

An den automatisch ermittelten, minimalen Latenzen (siehe 4.1.5) kann man klar ent-

nehmen, dass der Raspberry Pi 2 Model B, insbesondere unter einer hohen Rechenlast, eine weitaus bessere Latenz aufweist. Der BeagleBone Green schneidet mit einer Latenz von maximal 113 Millisekunden bei weitem schlechter ab, was zum größten Teil an dem Einkernprozessor liegt. Interessanterweise hat der Linux-Kernel mit Realtime-Patch in beiden Fällen auch unter hoher Rechenlast eine schlechtere Latenz zur Folge.

Um die Korrektheit der Resultate zu verifizieren, wurde mithilfe des ALSA-Tools „alsaloop“ und dem Oszilloskop die Round-Trip-Time mit der minimal ermittelten Periodengröße von 68 Frames auf dem BeagleBone Green gemessen.

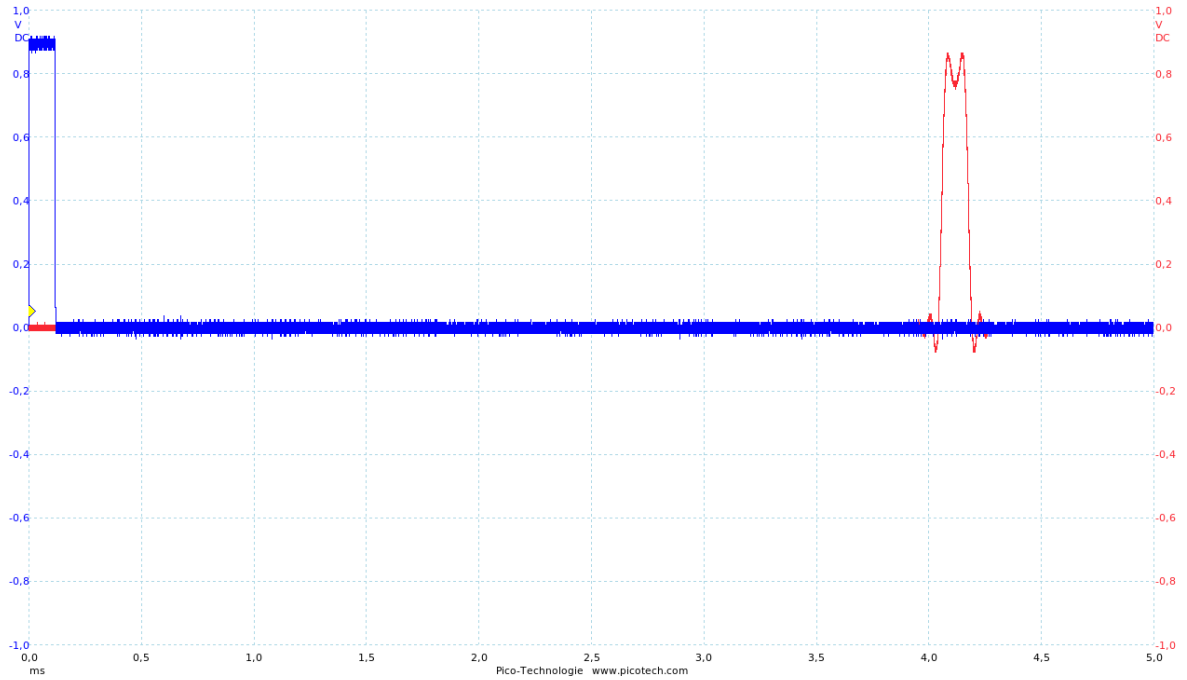

Abbildung 4.16: Verifizierung von Round-Trip-Time mit AD1938 AudioCard Rev.1 und einer Periode von 68 Frames auf BeagleBone Green

Dem Oszilloskop-Plot kann man eine Round-Trip-Time von ca. 4 Millisekunden entnehmen. Dies weicht um ca. 1,2 Millisekunden von der durch das Tool ermittelten Latenz ab. Um zu überprüfen, ob es sich um eine zeitliche Konstante handelt, wurde die Round-Trip-Time nochmals mit einer doppelt so großen Periode (136 Frames) gemessen, was einer Latenz von 5.666667 Millisekunden entspricht.

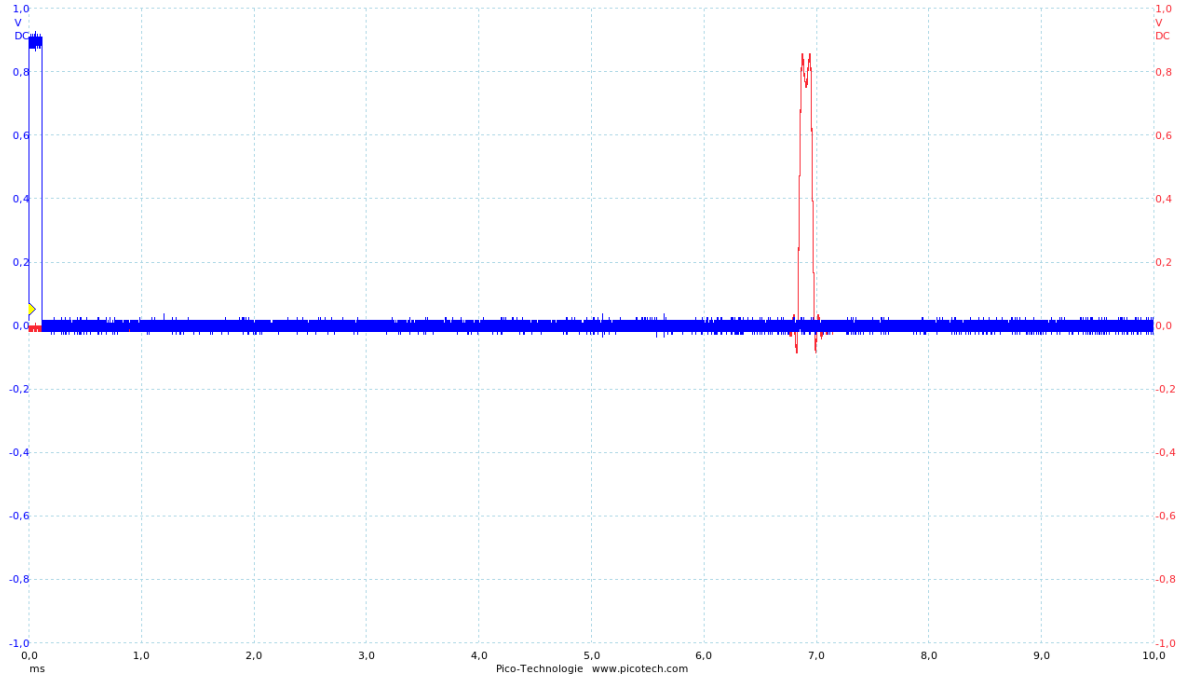

Abbildung 4.17: Round-Trip-Time mit AD1938 AudioCard Rev.1 und einer Periode von 136 Frames auf BeagleBone Green

Aus Abbildung 4.17 ist eine Round-Trip-Time von ca. 6,8 Millisekunden zu erkennen. Es handelt sich hier um eine zeitliche Konstante von ca. 1,2 Millisekunden, die durch die Signallaufzeit vom Audiocodec entsteht.

Um zu ermitteln, ob beim SuperAudioBoard mit dem Raspberry Pi 2 Model B ebenfalls eine ähnlich große zeitliche Konstante in der Latenz vorhanden ist, wurde ebenfalls die Round-Trip-Time mit der minimalen Periodengröße von 28 Frames (1,166667 Millisekunden) gemessen.

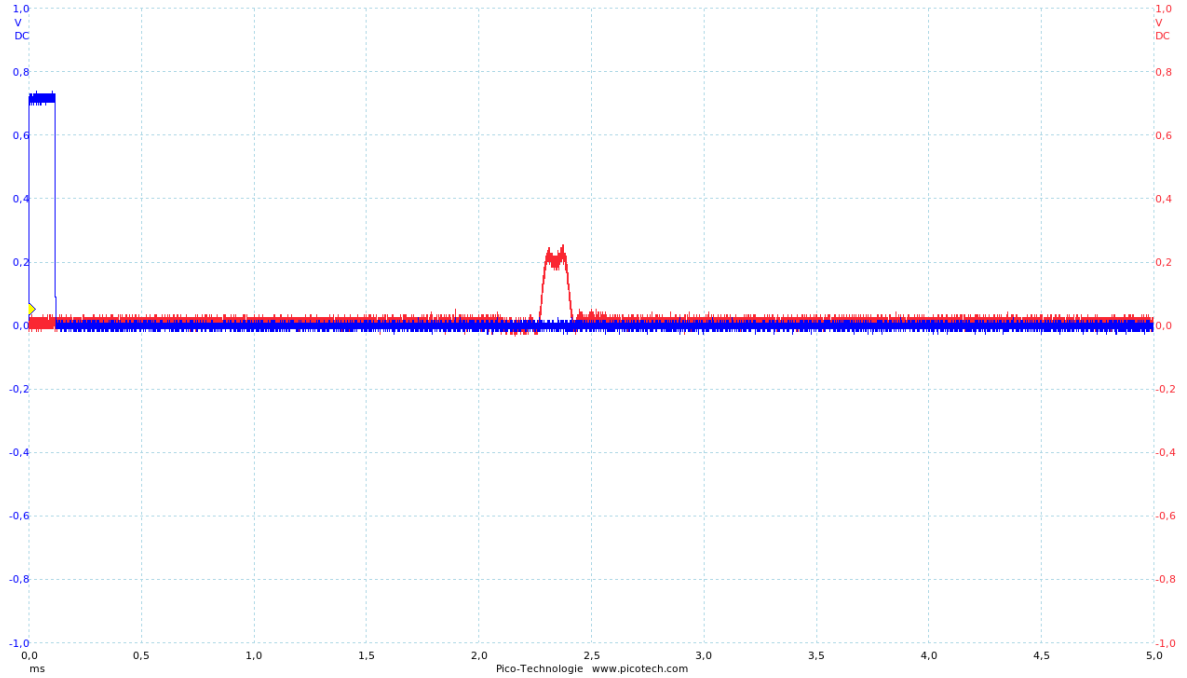

Abbildung 4.18: Round-Trip-Time mit SuperAudioBoard und einer Periode von 28 Frames auf Raspberry Pi 2 Model B

Laut des Oszilloskop-Plots entspricht die Round-Trip-Time ca. 2,3 Millisekunden. Hieraus ergibt sich also eine ähnlich große Zeitkonstante von 1,1 Millisekunden.

Da das SuperAudioBoard nicht mehr als 2 Audiokanäle unterstützt und das ALSA-Latenz-Testtool nur eine gleiche Anzahl von Audiokanälen der Ein- und Ausgabe unterstützt, folgen nun die Resultate des ALSA-Latenz-Testtools mit 4 Audiokanälen.

Testbedingungen	**BeagleBone Green**
Abtastrate: 48 kHz Kanäle: 4 PCM-Format: S32_LE Linux-Kernel: 4.1 Ohne Stress	Periode: 68 Frames Puffergröße: 136 Frames Latenz: 2.8333333 ms
Abtastrate: 48 kHz Kanäle: 4 PCM-Format: S32_LE Linux-Kernel: 4.1 Mit Stress	Ab einer Puffergröße von 4128 Frames tritt ein Speicherzugriffsfehler auf.
Abtastrate: 48 kHz Kanäle: 4 PCM-Format: S32_LE Linux-Kernel: 4.1-RT Ohne Stress	Periode: 104 Frames Puffergröße: 208 Frames Latenz: 4.166667 ms
Abtastrate: 48 kHz Kanäle: 4 PCM-Format: S32_LE Linux-Kernel: 4.1-RT Mit Stress	Ab einer Puffergröße von 4128 Frames tritt ein Speicherzugriffsfehler auf.

Tabelle 4.19: Latenztest mit 4 Kanälen

Interessanterweise ergeben sich beim Latenz-Test mit 4 Audiokanälen genau die gleichen Resultate, wie mit 2 Audiokanälen. Allerdings tritt mit dem Realtime-Kernel unter hoher Rechenlast ein Speicherzugriffsfehler auf. Auch hier wurden die Resultate mithilfe des Oszilloskops verifiziert.

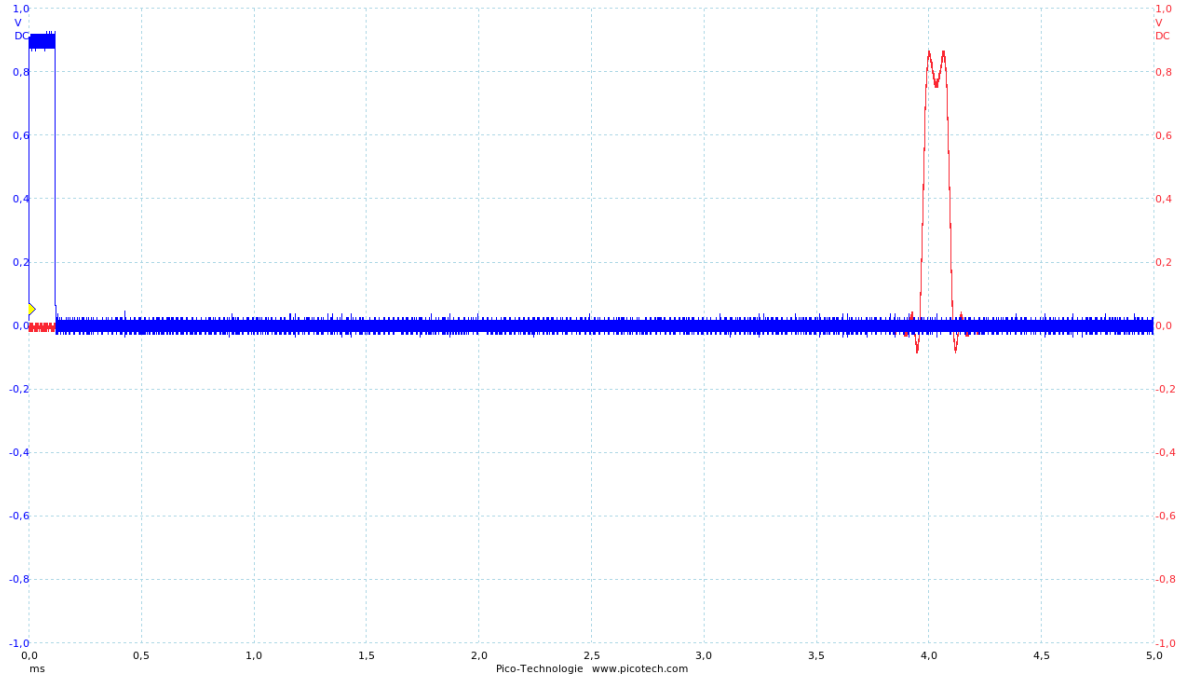

Abbildung 4.20: Round-Trip-Time von AD1938 AudioCard Rev.1 mit 4 Audiokanälen und einer Periodengröße von 68 Frames auf BeagleBone Green

Wie erwartet ergibt sich bei einer Periodengröße von 68 Frames genau die gleiche Round-Trip-Time von ca. 4 Millisekunden.

Fazit

Leider unterstützt die PCM-Schnittstelle vom Raspberry Pi 2 Model B nur 2 Audiokanäle, aufgrund dessen dieser für den AD1938 Audiocodec nicht geeignet ist. Allerdings weißt der BeagleBone Green mit einer effektiven Round-Trip-Time von 4 Millisekunden auch zufriedenstellende Resultate auf. Zu beachten ist jedoch, dass sich diese unter starker Rechenlast verschlechtert (bis zu 113 Millisekunden). Da sich mit dem Realtime-Linux-Kernel auch unter hoher Rechenlast schlechtere Resultate und sogar Speicherzugriffsfehler ergaben, sollte der unveränderte Linux-Kernel verwendet werden. Durch Hinzufügen eines Benutzers zur Gruppe „audio“ und Modifizierung der Datei /etc/security/limits.conf nach den offiziellen ALSA Empfehlungen für eine niedrige Latenz[44], ist es dem Benutzer erlaubt, Prozesse, die ALSA betreffen, höher zu Priorisieren, wodurch auch kürzere Perioden möglich sind. Dies soll durch folgenden Oszilloskop-Plot belegt werden, indem mit einer Periodengröße von 34 Frames die geringste Round-Trip-Time

von ca. 3,2 Millisekunden erreicht wurde:

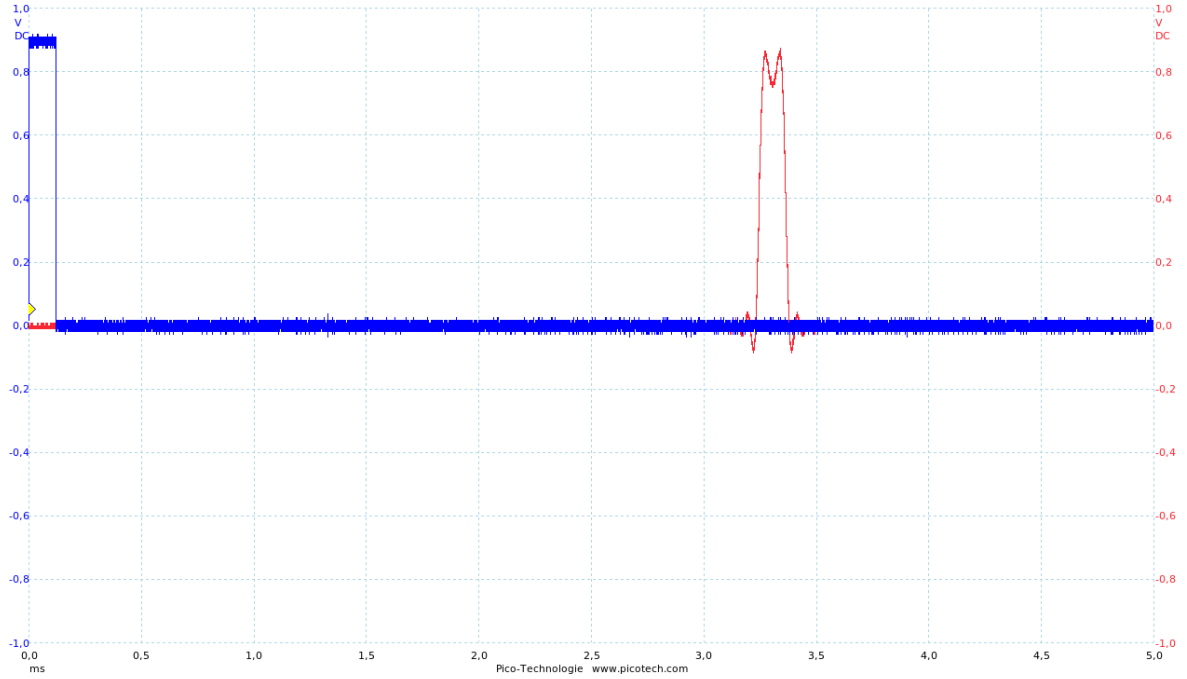

Abbildung 4.21: Minimal erreichte Round-Trip-Time von AD1938 AudioCard Rev.1 mit BeagleBone Green und einer Periodengröße von 34 Frames

5 Probleme und Lösungsansätze

Während der Bachelorarbeit ergaben sich zwei besondere Probleme, deren Lösung sich als äußerst zeitaufwendig erwies. Daher sollen diese hier kurz erläutert werden.

5.1 Digitales Audio Interface Raspberry Pi 2 Model B - TDM Slots

Im Laufe der Bachelorarbeit stellte sich heraus, dass der ASoC-Plattform-Treiber[1] des Raspberry Pi 2 Model B standardmäßig leider nicht mehr als 2 Audiokanäle unterstützt. Dem BCM2835 Datenblatt ist zwar zu entnehmen, dass die Länge und Position des Frameclock-Signals der PCM-Schnittstelle frei programmierbar sind, jedoch hardwaremäßig nur 2 Audiokanäle zulässt[6, S. 119.]. Während der Recherche nach einer Alternativplattform erwies sich der BeagleBone Green aufgrund des McASP, welcher bis zu 32 TDM-Slots unterstützt[3, S. 4542.], als geeignete Wahl.

5.2 BeagleBone Green - Taktgenerierung

Ursprünglich wurde die PCM-Schnittstelle des Raspberry Pi 2 Model B durch den AD1938 Audiocodec getaktet und fungierte somit als Slave, was auch auf Anhieb problemlos funktionierte. Jedoch ergab sich beim BeagleBone Green, eine fehlerhafte Taktung des McASP. Durch die serielle Dekodierung mit dem Oszilloskop bestätigte sich dies ebenfalls, da die Bit- und Frameclock korrekt dargestellt wurden. Um zu verifizieren, ob die Eingänge bzw. Ausgänge des McASP tatsächlich korrekt funktionieren, wurde der

[1] `https://github.com/raspberrypi/linux/blob/rpi-4.1.y/sound/soc/bcm/bcm2835-i2s.c`

Audiocodec als Slave konfiguriert und die Taktgenerierung dementsprechend durch den McASP des BeagleBone Green durchgeführt. Mithilfe des Oszilloskops zeigte sich, dass die Frameclock unerwarteterweise der Frequenz der Bitclock entsprach (bei einer Abtastrate von 48 kHz und 8 TDM Slots, sind dies $f_{CLK} = 8 \cdot 32 \cdot 48kHz = 12,288Mhz$) und die Bitclock selbst gar keine Änderung aufwies. Da der McASP des BeagleBone Green äußerst viele Konfigurationsmöglichkeiten und damit Konfigurationsregister zur Verfügung stellt (81 x 32 Bit Register[3, S. 4599]), wurden die Register, welche die Taktgenerierung betreffen, durch Ausgabe von Kernel-Logs verifiziert. Allerdings ergab sich hierbei, dass alle betreffenden Register korrekte Werte aufwiesen, wodurch die Vermutung entstand, dass am BeagleBone Green tatsächlich etwas defekt war. Nach weiteren Tests entstand die Idee, den Device Tree vom BeagleBone Black statt vom BeagleBone Green zu nutzen, da dieser den McASP zur Ausgabe von Audio über HDMI einsetzt (der BeagleBone Green besitzt keinen HDMI-Port und hat somit standardmäßig keine Möglichkeit zur Ausgabe von Audio). Tatsächlich ergab sich, dass die Linux-Kernelmodule für den McASP mit dem BeagleBone Green Device Tree beim Systemstart nicht korrekt geladen wurden, wodurch die oben genannten Probleme entstanden. Da beim Device Tree vom BeagleBone Black zusätzliche Treiber für die HDMI-Schnittstelle geladen werden, welche der BeagleBone Green nicht benötigt, wurde auf Basis des BeagleBone Black Device Trees[2] ein neuer Device Tree[3] für den BeagleBone Green entwickelt.

[2] `https://github.com/henrix/beagle-linux/blob/4.1/arch/arm/boot/dts/am335x-boneblack.dts`

[3] `https://github.com/henrix/beagle-linux/blob/4.1/arch/arm/boot/dts/am335x-bonegreen-audiocard.dts`

6 Fazit und Ausblick

6.1 Kennwerte des Mehrkanal-Audiosystems

6.1.1 Abtastraten

Der AD1938 Audiocodec wird mit einem 12,288 MHz Quarz getaktet und die TDM-Slot-Breite beträgt immer 32 Bit. Aufgrund dessen ergeben sich im Zusammenhang mit der Anzahl der TDM-Slots die folgenden Abtastraten:

Anzahl TDM-Slots	**Abtastraten**
2	44,1 kHz, 48 kHz, 96 kHz, 192 kHz
4	44,1 kHz, 48 kHz, 96 kHz
8	44,1 kHz, 48 kHz

Tabelle 6.1: Unterstützte Abtastraten der AD1938 AudioCard

Resampling von 44,1 kHz Audiodaten

ALSA bietet über standardmäßig vorhandene Plugins, wie z.B. „plug“, bereits die Möglichkeit von 44,1 kHz Resampling an, so dass entsprechende Audiodaten problemlos abgespielt werden können. Allerdings entstehen durch die zusätzliche Rechenlast größere Latenzen bzw. XRUNs bei klein gewählten Perioden und die technische Klangqualität verschlechtert sich (siehe 4.2.1). Um das automatische Resampling zu nutzen, kann vor einen beliebigen Soundkartennamen das Kürzel „plug:“ angehängt werden (z.B. plug:DAC1, plug:hw:0, usw.).

6.1.2 PCM-Formate

Die unterstützen PCM-Formate der AD1938 AudioCard ergeben sich aus der Schnittmenge der unterstützten PCM-Formate des digitalen Audiointerfaces der CPU, sowie des Audiocodecs und können mit diversen ALSA-Tools ermittelt werden (siehe 3.3.5).

PCM-Format	**Beschreibung**
S8	Signed 8 Bit
S16_LE	Signed 16 Bit Little-Endian
S16_BE	Signed 16 Bit Big-Endian
FLOAT_LE	Floating-Point Little-Endian
S32_LE	Signed 32 Bit Little-Endian
S32_BE	Signed 32 Bit Big-Endian

Tabelle 6.2: Unterstützte PCM-Formate der AD1938 AudioCard

Bei den 32 Bit PCM-Formaten werden effektiv nur 24 Bit genutzt, da die ADCs bzw. DACs des AD1938 Audiocodecs[5] nur maximal 24 Bit unterstützen. Weiterhin handelt es sich bei dem PCM-Format *FLOAT_LE* um keine echte Gleitkommaarithmetik, da die Gleitkommazahl im Treiber vor der Übertragung zum Audiocodec auf eine Ganzzahl abgebildet wird. Die restlichen PCM-Formate werden hardwaretechnisch sowohl vom Audiointerface der CPU, wie auch vom Audiocodec unterstützt.

6.1.3 Latenz

Die Latenz des Mehrkanal-Audiosystems ist abhängig von der Periodengröße (2.8.1) und ist über die virtuellen ALSA-Schnittstellen auf 128 Frames festgelegt. Da im Normalfall die Puffergröße doppelt so groß wie die Periode ist, beträgt sie 256 Frames. Dies entspricht einer Latenz von 10.666667 Millisekunden (berechnet mit ALSA-Tool „latency“ 4.1.5 und praktisch bestätigt mit Demonstrationsprojekt 6.3).

6.1.4 Technische Klangqualität

Die technische Klangqualität liegt im normalen Bereich für Computer-Audio, weicht jedoch mit mit einer Differenz von ca. 10 dB von den Kennwerten des AD1938 Audiocodecs ab. Im Kapitel „Evaluierung vorhandener Platinen“ (4.2) werden die Kennwerte, welche die technische Klangqualität betreffen, detailliert erläutert und werden daher hier nicht mehr ausgeführt.

6.2 Angepasste Debian Distribution

Da für die Nutzung der AD1938 AudioCard die Kompilierung der ALSA-Treibers mit dem Linux-Kernel 4.1 durchgeführt und diverse Einstellungen optimiert werden mussten, was einen recht hohen zeitlichen Aufwand bedeutet und viele potentielle Nutzer abschrecken könnte, wurde ein eigenes Debian Image erstellt. Standardmäßig werden die Treiber mit 8 TDM-Slots konfiguriert und beim Systemstart automatisch geladen, wodurch, wie oben beschrieben, eine maximale Abtastrate von 48 kHz möglich ist. Sollte der Bedarf für höhere Abtastraten entstehen, wurden 3 separate Capes für 2, 4 und 8 TDM-Slots erstellt. Die Anzahl der TDM-Slots können in der Datei /etc/rc.local konfiguriert werden (z.b. sh -c “echo ’BBB-AD193X-4TDM’ >/sys/devices/platform/bone_capemgr/slots“). Weiterhin wurde mithilfe des ALSA Plugins „route“ in der Datei /etc/asound.conf virtuelle PCM-Schnittstellen so konfiguriert, dass die ADCs bzw. DACs über ihren entsprechenden Namen angesprochen werden können (ADC1 bis ADC2 bzw. DAC1 bis DAC4).

6.3 Demonstrationsprojekt

Zur Belegung der Hypothese und vollständig erreichten Ziele dieser Bachelorarbeit, wurde mithilfe der C++-Bibliothek DSPatch[45] ein Demonstrationsprojekt entwickelt, welches alle 4 Eingänge und 8 Ausgänge der AD1938 AudioCard simultan nutzt. Hierbei durchlaufen alle 4 Audiosignale der analogen Eingänge jeweils ein Verstärkungsmodul mit dem Faktor 1, werden zu einem Audiosignal gemischt, 8 Audio Delayeffekte mit unterschiedlichen Parametern darauf angewendet und anschließend jeweils über einen

analogen Ausgang ausgegeben, wodurch ein räumlicher Surround-Effekt entsteht. Hierbei ist natürlich auch die Latenz maßgebend, welche mit ca. 11 Millisekunden Round-Trip-Time durchaus zufriedenstellend ist. Es folgt ein Oszilloskop Plot von jeweils einem Eingang und Ausgang mit einem 10 Millisekunden Delayeffekt, wobei die rote Kennlinie das Eingangssignal und die blaue Kennlinie das Ausgangssignal darstellt. Die zeitliche Auflösung entspricht 5 Millisekunden pro Div.

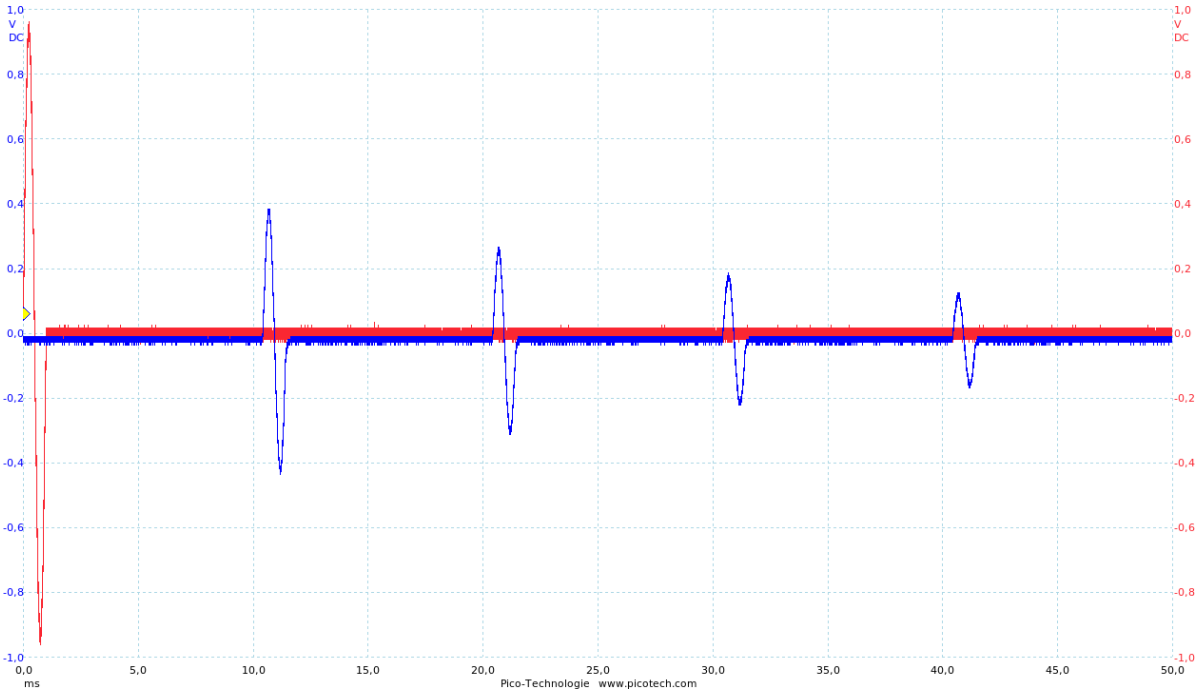

Abbildung 6.3: Demoprojekt Round-Trip-Time Plot mit 10 ms Delayeffekt

Die Round-Trip-Time ist durch den Abstand zwischen dem roten Eingangssignal und dem ersten blauen Ausgangssignal repräsentiert und beträgt ca. 11 Millisekunden. Die Abstände zwischen den einzelnen Sinusperioden der blauen Kennlinie entsprechen der Verzögerungszeit des Delayeffekts (10 Millisekunden). Weiterhin arbeitet der Delayeffekt in diesem Fall mit einem Dämpfungsfaktor von 0,7 (70% Feedback) und einem Mix von 0,5 (sowohl Ursprungssignal, wie auch berechnetes Delaysignal werden nicht verstärkt bzw. gedämpft). Das Demonstrationsprojekt wird mit einer Abtastrate von 48 kHz, dem PCM-Format S32_LE (Signed 32 Bit Little-Endian) und damit in höchstmöglicher Klangqualität mit 8 TDM-Slots ausgeführt. Durch die geringe Periodengröße bzw. Puffergröße wird die CPU allerdings stark belastet (84,2%), was dem folgenden Bildschirm-

foto von htop[1] entnommen werden kann:

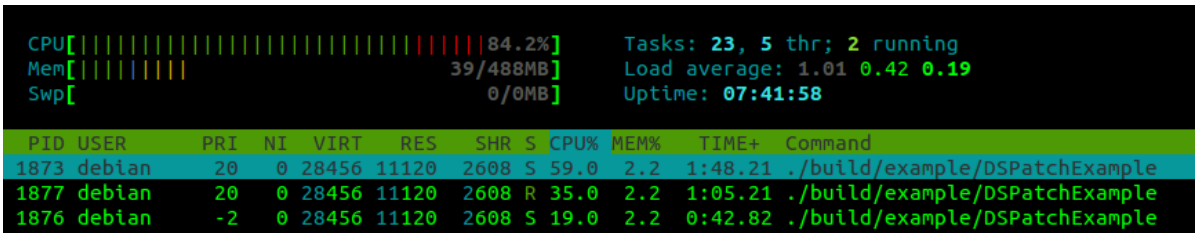

Abbildung 6.4: CPU Last während der Ausführung von Demonstrationsprojekt

Die Puffergröße ist zwar kleiner konfigurierbar, allerdings entstehen dadurch Pufferunterläufe bzw. Pufferüberläufe (XRUNs), welche sich als akustische Störgeräusche bemerkbar machen.

6.4 Fazit

Während dieser Bachelorarbeit konnte erfolgreich die Realisierung eines Mehrkanal-Audiosystems mit niedriger Latenz auf Basis von Linux durchgeführt werden. Der AD1938 Audiocodec kann mit dem BeagleBone Green Einplatinencomputer in vollem Umfang genutzt werden. Durch die Optimierung der Perioden- und Puffergröße konnte eine Latenz bzw. eine Round-Trip-Time von ca. 4 Millisekunden erreicht werden, was durchaus zufriedenstellend ist. Das Demonstrationsprojekt belegt die Sinnhaftigkeit, den BeagleBone Green in Verbindung mit der AD1938 AudioCard in Anwendungsgebieten für Musiker zu nutzen. Es erwies sich zwar, dass es praktisch ohne zusätzliche Hardware nicht möglich ist, mehr als 2 Audiokanäle auf Basis des Raspberry Pi 2 Model B zu unterstützen, da das digitale Audio Interface des BCM2835 dies nicht zulässt, jedoch hat sich der BeagleBone Green als gute Alternative herausgestellt. Ein Nachteil ist, dass durch den Einkernprozessor unter Rechenlast weitaus größere Latenzen entstehen. Der Linux-Kernel mit Realtime-Patch hatte im Vergleich zum normalen Kernel interessanterweise auch unter großer Rechenlast größere Latenzen zur Folge.

[1] dynamischer Prozessmanager für Linux (`http://linux.die.net/man/1/htop`)

6.5 Ausblick

Da aktuell zunehmend verschiedene Einplatinencomputer mit stark ansteigender Rechenperformance, wie z.B. der BeagleBoard-X15[2], verfügbar werden, ist davon auszugehen, dass in Zukunft komplexere Berechnungen bei gleichbleibender oder gar geringerer Latenz realisierbar sind. So könnte man beispielsweise mithilfe der DSPatch-Bibliothek oder anderen Bibliotheken längere und aufwendigere Effekt-Ketten realisieren. Weiterhin kann das Audiosystem auch als externe Surround-Soundkarte über USB mithilfe des Soundservers PulseAudio[3] genutzt werden. Zwar entsteht durch die aufwendige Übertragung per TCP eine größere Latenz, jedoch ist dies bei der einfachen Wiedergabe von Audiodaten nicht relevant.

[2] `http://beagleboard.org/x15`
[3] `http://manurevah.com/blah/en/p/PulseAudio-Sound-over-the-network`

Quellen

[1] SuperAudioBoard Repository. https://github.com/whollender/SuperAudioBoard.

[2] Philips Semiconductors, High Tech Campus 60 5656 AG Eindhoven, Noord-Brabant, The Netherlands. *I^2S bus specification*, june 5, 1996 edition, 1996. https://www.sparkfun.com/datasheets/BreakoutBoards/I2SBUS.pdf.

[3] Texas Instruments, Inc., 12500 TI Boulevard Dallas, Texas 75243 USA. *AM335x Technical Reference Manual*, february 2015 edition, 2015. http://www.ti.com/lit/ug/spruh73l/spruh73l.pdf.

[4] STMicroelectronics N.V., STMicroelectronics N.V., 39, Chemin du Champ des FillesPlan-Les-Ouates, CH1228 Geneva. *ST SPI protocol*, doc id 023176 rev 2 edition, 2013. http://www.st.com/st-web-ui/static/active/en/resource/technical/document/technical_note/DM00054618.pdf.

[5] Analog Devices, Inc., P.O. Box 9106, Norwood, MA 02062-9106, U.S.A. *AD1938 Datenblatt*, rev. e edition, 2013. http://www.analog.com/media/en/technical-documentation/data-sheets/AD1938.pdf.

[6] Broadcom Corporation, Broadcom Europe Ltd. 406 Science Park Milton Road Cambridge CB4 0WW. *BCM2835 ARM Peripherals*, 06 february 2012 edition, 2012. http://www.farnell.com/datasheets/1521578.pdf.

[7] NXP Semiconductors N.V., High Tech Campus 60 5656 AG Eindhoven, Noord-Brabant, The Netherlands. *I^2C-bus specification and user manual*, rev. 6 — 4 april 2014 edition, 2014. http://www.nxp.com/documents/user_manual/UM10204.pdf.

[8] Teensy 3.1 Webseite. https://www.pjrc.com/teensy/teensy31.html.

[9] Freescale Semiconductor, Inc., Corporate Headquarters, 6501 William Cannon Drive West, Austin, Texas 78735, USA. *Teensy 3.1 MK20DX256VLH7 Datenblatt*, rev. 1.1, dec 2012 edition, 2012. https://www.pjrc.com/teensy/K20P64M72SF1RM.pdf.

[10] Raspberry Pi 2 Webseite. https://www.raspberrypi.org/products/raspberry-pi-2-model-b/.

[11] Raspberry Pi 2: Performance-Vergleich und Benchmarks. http://jankarres.de/2015/03/raspberry-pi-2-performance-vergleich-und-benchmarks/.

[12] BeagleBone Green Webseite. http://beagleboard.org/green.

[13] Raspberry Pi 2 Benchmark-Test – große Steigerung gegenüber dem Pi 1. https://www.bitblokes.de/2015/02/raspberry-pi-2-benchmark-test-grosse-steigerung/.

[14] BeagleBoard.org Foundation, BeagleBoard.org Foundation, 4467 Ascot Ct, Oakland Twp, MI 48306. *BeagleBone Black System Reference Manual*, rev c.1 edition, 2014. https://github.com/CircuitCo/BeagleBone-Black/blob/master/BBB_SRM.pdf.

[15] Eigenes BeagleBone Linux Repository. https://github.com/henrix/beagle-linux.

[16] Cirrus Logic, Inc., 800 West 6th Street, Austin, Texas 78701, United States. *CS4272 Datenblatt*, august '05 ds593f1 edition, 2005. https://www.cirrus.com/en/pubs/proDatasheet/CS4272_F1.pdf.

[17] Advanced Linux Sound Architecture (ALSA) project homepage. http://www.alsa-project.org/main/index.php/Main_Page.

[18] Sound Systems on Linux: From the Past To the Future. http://www.alsa-project.org/~tiwai/soundsystems.pdf.

[19] ALSA FramesPeriods. http://www.alsa-project.org/main/index.php/FramesPeriods.

[20] ALSA System on Chip (ASoC). http://www.alsa-project.org/main/index.

php/ASoC.

[21] Seeed Technology Limited., F5, Building 8, Shiling Industrial Park, Xinwei, Number32, Tongsha Road Xili Town, Nanshan District, Shenzhen, China. P.R.C 518055. *BeagleBone Green System Reference Manual*, rev v1a edition, 2015. http://www.seeedstudio.com/wiki/images/0/0f/BBG_SRM_V1a_20151009.pdf.

[22] AM335x Audio Driver's Guide. http://processors.wiki.ti.com/index.php/AM335x_Audio_Driver's_Guide.

[23] How it works: Linux audio explained. http://tuxradar.com/content/how-it-works-linux-audio-explained.

[24] Jack Audio Connection Kit Webseite. http://www.jackaudio.org/.

[25] PulseAudio Webseite. https://wiki.freedesktop.org/www/Software/PulseAudio/.

[26] Real-Time Linux Wiki. https://rt.wiki.kernel.org/index.php/Main_Page.

[27] Alessandro Rubini Jonathan Corbet and Greg Kroah-Hartman. *Linux Device Drivers*. O'Reilly Media, Inc, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, third edition edition, 2005. http://free-electrons.com/doc/books/ldd3.pdf.

[28] Raspberry Pi Kernel Compilation. http://elinux.org/Raspberry_Pi_Kernel_Compilation.

[29] BeagleBoneBlack Building Kernel. http://wiki.beyondlogic.org/index.php/BeagleBoneBlack_Building_Kernel.

[30] Embedded Linux - Device Tree Info. http://elinux.org/Device_Tree.

[31] Device trees, overlays and parameters. https://www.raspberrypi.org/documentation/configuration/device-tree.md.

[32] BeagleBone Device Tree Overlays Repository. https://github.com/beagleboard/bb.org-overlays.

[33] The ALSA Driver API. https://www.kernel.org/doc/htmldocs/alsa-driver-api/index.html.

[34] ALSA project - the C library reference. http://www.alsa-project.org/alsa-doc/alsa-lib/index.html.

[35] Writing an ALSA driver. http://www.alsa-project.org/~tiwai/writing-an-alsa-driver/.

[36] Eigenes Raspberry Pi Linux Repository. https://github.com/henrix/rpi-linux.

[37] BeagleBone Black Audio Cape RevB Gettings Started. http://elinux.org/BBB_Audio_Cape_RevB_Getting_Started.

[38] ALSA Asoundrc. http://www.alsa-project.org/main/index.php/Asoundrc.

[39] ALSA PCM (digital audio) plugins. http://www.alsa-project.org/alsa-doc/alsa-lib/pcm_plugins.html.

[40] Analog Devices, Inc., P.O. Box 9106, Norwood, MA 02062-9106, U.S.A. *The Data Conversion Handbook*, 2004. http://www.analog.com/library/analogDialogue/archives/39-06/data_conversion_handbook.html.

[41] Analog Devices, Inc., P.O. Box 9106, Norwood, MA 02062-9106, U.S.A. *Understand SINAD, ENOB, SNR, THD, THD + N, and SFDR so You Don't Get Lost in the Noise Floor*, rev.a, 10/08, wk edition, 2008. http://www.analog.com/media/en/training-seminars/tutorials/MT-003.pdf.

[42] Measurement of Loudspeaker Frequency Response with Matlab Implementation. .

[43] stress - tool to impose load on and stress test systems. http://linux.die.net/man/1/stress.

[44] Low latency howto. http://www.alsa-project.org/main/index.php/Low_latency_howto.

[45] DSPatch - C++ Cross-Platform, Object-Oriented, Flow-Based Programming Library. http://flowbasedprogramming.com/DSPatch/.