Distributed Audio Processing

Alexander Gustafson
University of Applied Sciences,
Zürich,
Switzerland,
alex_gustafson@yahoo.de

October 20, 2015

Abstract

In modern profesional music studios, the computer has become responsible for tasks that were previously performed by dedicated hardware. Mixing boards, effect processors, dynamic compressors and equalizers, even the instruments themselves, are all available as software. To alleviate the processing load on the CPU there is a growing market for specialized DSP coprocessors which can process mutliple channels of digital audio in realtime. These coprocessors are typically connected via Firewire or PCIe and use multiple DSP chips for the processing. This project will examine an inexpensive alternative based on standard Gigabit Ethernet and higher end Raspberry Pi clones.

Contents

Chapter 1

Introduction

1.1 Ausgangslage

20 years ago the CPU was just one component of a typical music studio. It was generally used to control and synchronize other equipement such as mixing boards, multitrack recorders, synthsizers and effects processors. Today all of the other equipement exists as software, running in realtime on a CPU host. A typical music studio today is comprised of a CPU, multiple analog to digital inputs and outputs, and some DSP equiped audio processing cards.

Simliar to GPU Cards which can accelerate graphics and visualization applications, audio DSP cards can process multiple streams of high qualtiy digital audio, eliviating the load on the CPU Host Computer. Audio DSP cards typically connect to the CPU via PCI, Firewire, or Thunderbolt. Most vendors of DSP cards offer the possibility to connect several cards in parallel to increase the processing capacity.

Unlike GPU processors however, no open standard has evolved to distribute the load across multiple co processors in the way it has for openGL or openCL. 3D graphics applications profit enormously from the interoperability that openGL offers. No such benefit is available for digital audio applications. Also, unlike OpenGL applications, audio software that is developed to run on an audio DSP card cannot be run on the CPU host. This results in vendor lock-in. The consumer that invests in an audio DSP card and software, must continue to buy from the same vendor in order to build on the the initial investment. If another vendor of DSP hardware creates a superior product, a consumer is unlikely to switch platforms if a significant investment has already been made.

10 years ago this was an acceptable compromise because DSP processors connected via PCIe could provide a significant performance increase. Today however, arm based inexpensive CPUs connected via standard gigabit ethernet could offer a competative alternatvie.

1.2 Ziel der Arbeit

Das Ziel dieser Studienarbeit ist es, die Möglichkeiten für die Verteilung von Audio Verarbeitungsaufgaben für einen Software-basierten Musik Synthesizer über ein Netzwerk von SBC-Geräten (Single Board Computer) zu untersuchen. Einschränkungen in der Polyphonie oder der Prozessorleistung sollen durch einfaches Hinzufügen eines neuen Geräts im Netzwerk gelöst werden. Ein besonderes Augenmerk soll auf günstige Raspberry-Pi und ähnliche SBC Geräte gelegt werden, vor allem unter Berücksichtigung von Preis und Leistung. Es existieren viele Standards, damit Audio Applikationen miteinander kommunizieren können. MIDI und OSC definieren Protokolle zur Steuerung von Audio- Geräten und Software. VST, AU und LW2 sind Standard Schnittstellen für Plugins, die Echtzeitaudioverarbeitung bieten. AVB (IEEE 1722), Jacktrip und Dante sind Standards bzw. Software für die Übertragung von hochwertigem Audio über Ethernet mit minimaler Latenz. Diese Studienarbeit wird einige dieser Standards untersuchen, um zu eruieren, was notwendig ist, um einen skalierbaren und verteilbaren Musik-Synthesizer zu entwickeln, der mit anderen professionellen Audio Software kompatibel ist. Eine Proof-of-Concept Version der verteilbaren Synthesizer Software wird entwickelt, welche es einem Benutzer erlauben wird, ein Musikstück in Echtzeit zu spielen.

1.3 Aufgabenstellung

- Anforderungsanalyse mit Prioritätsbewertung
- Vergleich von mehreren CPUs und Embedded Systems (Banana Pi, Adapteva, Odroid) hinsichtlich ihrer Nutzbarkeit als Echtzeit Audioverarbeitungsmodule. Mit dem System, das die Anforderungen am besten erfüllt, wird die Implementierung gemacht.
- Entwicklung der Audioverarbeitungssoftware in C ++.
- Entwicklung eines VST-Plugins in C++, das als Schnittstelle zwischen gängigen Audio-Software und den Audioverarbeitungsmodule (pkt 3) dient.
- Analyse der Implementierung, um die Nützlichkeit und Skalierbarkeit zu bewerten. Es ergeben sich dadurch verschiedene Fragestellungen wie z.B. folgende: Kann die Leistung und Polyphonie durch Hinzufügen weiterer Module erhöht werden, oder wird der Kommunikations-Overhead schließlich zu gross?

Chapter 2

Einführung ins Thema

2.1 Background

Vor 20 Jahren musste man ins Tonstudio um einen Musik- oder Audioaufnahme zu bearbeiten. Dort befanden sich großer Racks voller Geräte für verscheidene Signalverarbeitungsaufgaben. Zum Beispiel, Komprssoren und Limiters um den Dynamikbereich zu verarbeiten, oder Hall und Echo geräte um einen Tonspur mehr Atmosphäre zu vergeben. Am Mischpult konnte man Lautstärke und Frequenzgang mehrere Tonspuren anpassen.

Heute werden alle diese Aufgaben von Software-Plug-ins auf der CPU berechnet.

1996 hatte Steinberg GmbH, die Entwickler von Cubase, einem populären Audio-Produktionssoftware (oder DAW, Digital Audio Workstation), die VST-Schnittstelle und SDK veröffentlicht. Das VST-Plugin-Standard war Besonderes, weil es Echtzeit-Verarbeitung von Audiodaten in der CPU ermöglichte. Dazu konnten andere Entwickler jetzt Plugins entwickeln die innerhalb Cubase ausgeführt werden konnten. Das VST-Plugin-Standard hatte schnell Akzeptanz in der Branche gefunden und wurde sogar von konkurrierenden DAWs implementiert. Obwohl alternative Standards heute existieren, ist VST immer noch die am weitesten verbreitete Crossplatform-Standard.

Die Zahl der Echtzeit-Plug-ins, die auf einer CPU laufen konnte, wurde durch mehrere Faktoren wie, Festplattenzugriffsgeschwindigkeiten, Bus-Geschwindigkeiten, viel RAM und OS-Scheduler zum Beispiel beschränkt [3]. Benutzer damals erwarteten nicht mehr als 10 Plugins gleichzeitig ausführen zu können. Schon der Wiedergabe mehrerer Spuren ohne Plug-ins konnte einen Rechner ins Stottern bringen.

Heute ist es jedoch möglich, hunderte Tonspuren und Hunderte Plugins in Echtzeit wiederzugeben. Während die Leistungsgrenze angestiegen ist, stiegen auch die Erwartungen der Benutzer. Die Algorithmen heutiger Plugins sind sehr viel komplexer, als diejenigen von 1996. Es werden Akustische Systeme detailgenau Modeliert und die Schaltkreise alte 70er Jahre Synthesisers digital emuliert. Auch wenn die CPU-Leistung sich deutlich erhöht hat, ist es noch leicht, die Grenzen zu erreichen.

Mehrere DSP-basierten Systemen existieren, die, ähnlich wie GPU-Beschleunigungskarten, die CPU Belastung lindern. Audio-Verarbeitungsaufgaben werden an externe Hard-

ware via PCIe oder Thunderbolt-Schnittstellen übertragen. Jedoch sind diese DSP-basierten Systeme proprietäre und teuer. Die Entwicklung von Software für einen DSP-Chip ist auch einiges komplexer als für einen CPU.

2.2 Echzeit Audio Plug-ins

Das Produzieren von Audio und Musik wird in der Regel mit Hilfe eines Digital Audio Workstations (DAW) Software gemacht. MIDI-Events und Audio-Aufnahmen werden als Spuren arrangiert, gemisht, editiert und verarbeitet. Um Änderungen rückgängig zu machen, werden Bearbeitungen in einer Nicht-destruktives Verfahren gemacht, dynamisch in Echtzeit während der Wiedergabe berechnet. Die ursprünglichen Audiodaten beleiben immer im original Zustand erhalten. Der Benutzer kann die Parameter eines Effektes oder Prozess in Echtzeit ändern, und mit verschiedenen Einstellungen experimentieren, ohne zu fürchten, dass die ursprüngliche Audioaufzeichnungen geändert werden.

Ein DAW-Anwendung hat in der Regel mehrere Echtzeit-Effekte eingebaut, welche ein Benutzer auf Audiospuren anwenden kann. Zusätzlich zu den eingebauten Effekte sind alle professionellen DAW-Anwendungen auch in der Lage, Dritthersteller-Plugins zu laden. In Abhängigkeit von der Plattform und Anbieter werden einer oder mehreren Plugin-Standards implmeneted. Die am weitesten verbreitete Standard ist Steinbergs VST-Standard.

Alle Standards funktionieren in einer ähnliche Weise. Der Host DAW-Anwendung übergibt in regelmäßigen Abständen das Plug-in über einen Rückruffunktion zu-verarbeitende Audiodaten. Das Plug-in muss innerhalb einer definierten Zeitrahmen die Verarbeitung abgeschlossen haben und wartet dann auf das nächste Abruff.

Audio-Plug-ins können auch einen Grafisches Oberfläche zu verfügung stellen über die ein Benutzer Parameter ändern und speichern kann. Dies könnte die Grenzfrequenz eines Tiefpassfilters oder der Verzögerungszeit eines Hall-Effekt sein, zum Beispiel.

Aus der Sicht des Programmierers sind Plug-ins dynamisch ladbare Bibliotheken, die ein spezifiziertes API implementieren. Der Host-DAW-Anwendung dann sie zur Laufzeit laden und Audiodaten durch sie verarbeiten lassen [6]. Auf der Windows-Plattform werden VST-Plugins als $Dynamic\ Link\ Libraries$ (DLL) kompiliert, auf Mac OSX sind sie Mach-O-Bundles. Die native Apple AudioUnit Plug-ins sind auch als Mach-O Bundles zusammengestellt, sie haben fast identische Funktionalität. Andere alternative Plugin Formate sind Avids RTAS und AAX-Plug-Formate, Microsofts DirectX-Architektur oder LADSPA, DSSI und GW2 auf Linux.

Echtzeit Audio Plug-ins, wie der Name schon sagt, mussen in der Lage sein, ihre Aufgaben schnell genug, um die Anforderunen Echtzeit-Audio zu erfüllen. Wie schnell ist schnell genug? Nun, das hängt davon ab, wie man "Echtzeit" definiert. Für Audio Anwendungen wird Echtzeit in Bezug auf das Latenz des Audiosystems definiert. Die Gesamtverzögerung zwischen das Eintreff eines Audiosignals in das System (beim Analog-Digital-Wandler zum Beispiel), Verarbeitung, und das Verlassen des Systems (beim Digital-Analog-conterver oder Audio Ausgang) ist die Latenz. Die maximal zulässige Latenzzeit für Audioanwendung ist ungefähr 10 ms [7]. Höher als 10ms

wird die Latenzzeit als störend empfunden und ist nicht mehr akzeptabel für Live-Performance-Anwendungen.

Jeder Berechnung wird eine gewisse Verzögerung hinzufügen. Doch innerhalb der Audio-Verarbeitungsfunktion, muss der Programmierer darauf achten, keine unnötigen oder unberechenbare Verzögerungen zu erzeugen. Es ist auch wichtig zu verstehen, dass die Audio-Verarbeitungsfunktion im Kontext eines Hoch-Prioritäres System-Thread berechnet wird. Nichts innerhalb dieser Funktion sollte auf Ressourcen warten welche von anderen Threads mit niedrigerer Priorität berechnet werden. Beispiele für Dinge zu vermeiden, sind Speicher Allozierungen oder Deallozierungenoder, Warten auf- oder Sperren eines Mutexes, bedingte Ausdrücke innerhalb von Schleifen, die Pipe-Lining-Optimierungen brechen könnten [6] oder direkt Aktualisierungen des Grafischen-Oberfläches vornehmen.

2.3 Audio Over Ethernet

Das Versenden von Audiodaten über Netzwerke ist nicht neu. Auch das Senden von Audiodaten in Echtzeit nicht. Das IETF (Internet Engineering Taskforce) RFC 3550 definitert das Real-time Transport Protocol für die Bereitstellung von Audio und Video in Echtzeit über IP-Netzwerke. RTP wird als Grundlage für die meisten Medien-Streaming und Video-Conferencing-Anwendungen eingesetzt.

Andere neue Spezifikationen wie AVB und AES67 bauen auf RTP und ermöglichen acurate Timing und Synchronisation für professionelle Audio-Anwendungen. Die Synchronisation ist in diesen Standards wichtig, weil sie sich mit der Steuerung von Audiohardware vetteilt auf mehreren Hosts befassen.

Hardware-Synchronisation ist nicht für dieses Projekt relevent, weil es sich nicht mit externen Audio-Hardware befasst. Das Ziel dieses Projektes ist es, externe CPUs als Audio-Koprozessoren über Gigabit-Ethernet zu nutzen. Trotzdem, die AVB und AES67 Standards bieten viele Erkenntnisse darüber, wie die Datenübertragung für Low-Latency-Anwendungen optimieret werden kann und ist ein Proof-of-Concept, dass es möglich ist. AES67 definiert Richtlinien um Latenzzeiten deutlich unter 1 ms zu erreichen, auch bei hunderte parallel laufende Audiostreams. Dies ist viel schneller als die Legacy-PCI und Firewire Verbindungen welche in vielen DSP-basierten Systemen verwendet worden [1].

2.4 Single Board Computers

Die Popularität der Raspberry Pi hat eine ganze Industrie rund um Single-Board-Computer (SBC) generiert. Basierend auf Hardware welches in Mobiltelefonen zu finden ist, sind diese kleinen Low-Power-Geräte sehr beliebt, weil sie preiswert und einfach zu bedienen sind. Der größte Vorteil von SBCs im Vergleich zu anderen eingebetteten Geräten ist, dass sie mit Android- und Linux-Betriebssysteme ausgeführt werden, so dass sie mit den gleichen Tools auf Desktop-Computern zur Verfügung programmiert werden.

Aktuelle High-End-SBCs werden sogar mit Gigabit-Ethernet ausgestatet und Quad-Core-CPUs mit Geschwindigkeiten weit über 1 GHz getaktet. Vergleicht man diese

Systeme mit den 450 MHz G3 PPC-Systeme auf dem die ersten VST Software Plug-ins berechnet wurden, sollten wir erwarten können, dass die neueren High-End-SBCs exzellente Audio Coprozessoren sein sollen.

Zwei SBCs sind es wert, in Betracht gezogen zu werden, da sie möglicherweise noch bessere Performance als Audio Coprozessoren beiten. Das Parallella Board hat einen 16 Core Epiphany Coprozessor welches verwendet werden kann, um die Audioverarbeitung parallel auszuführen. Standard Frameworks wie OpenCL, MPI oder OpenMP können verwendet werden, um die Epiphany-Cores zu benützen. Der ODROID-XU4 SBC enthält eine Mali-T628 GPU-Coprozessor, der auch OpenCL kompatibel ist. Beide sind für weniger als $ 100 erhältlich.

Der Programmierung von Audioverarbeitungsalgorithmen als OpenCL Kernels könnte einiges komplexer sein als in C ++, aber OpenCL bietet herstellerunabhängige Zugriff auf GPGPU-Computing und hat den zusätzlichen Vorteil, dass es auch auf einem CPU ohne GPU-Beschleunigung verwendet werden kann [5].

Die Untersuchung dieser und anderer OpenCL fähiger SBCs könnte eine interessantes Folgeprojekt sein.

2.5 Virtual Analog Synthesis

Der Begriff "Virtueller Analoger Synthese" wird verwendet, um die Echtzeit-Emulation des analogen Synthesizern der 60er und 70er Jahre zu beschreiben. Die Komplexität und die Ziele einer Emulation können variieren. Einige Emulationen gehen so weit, dass sie die tatsächlichen elektronischen Komponenten der Vintage-Synthese-Schaltungen zu simulieren, andere modellieren nur grob den Signalfluss.

Unabhängig von der Art der Emulation hat virtuell analoger Synthese zwei Probleme mit dem es sich Beschäftigen muss, Latenzzeit und Aliasing. Das Problem der Latenzzeit wurde bereits oben beschrieben. Die Verarbeitung wird eine Verzögerung des Signals erzeugen, die Komplexität der Verarbeitung kann die Verzögerung erhöhen oder mehr CPU-Zyklen verbrauchen. Aliasing ist ein hörbare Verzerrungen des Signals, welches durch Frequenzen verursacht werden die höher sind als die Abtastrate des Systems erlaubt.

Analog-Synthesizer verwenden üblicherweise Subtraktive Synthese. Ein oder mehrere Klangerzeuger (Oszillatoren) erzuegen Signale mit besondere harmonische Qualitäten. Diese Signale werden durch Filter geschickt, welche die Frequenzen aus dem Signal "Subtraktieren". Die Oszillatoren, Filter und Amplitude können moduliert werden. Abbildung 2.1 ist ein einfaches Blockschaltbild eines typischen Subtraktive Synthesizer Stimme.

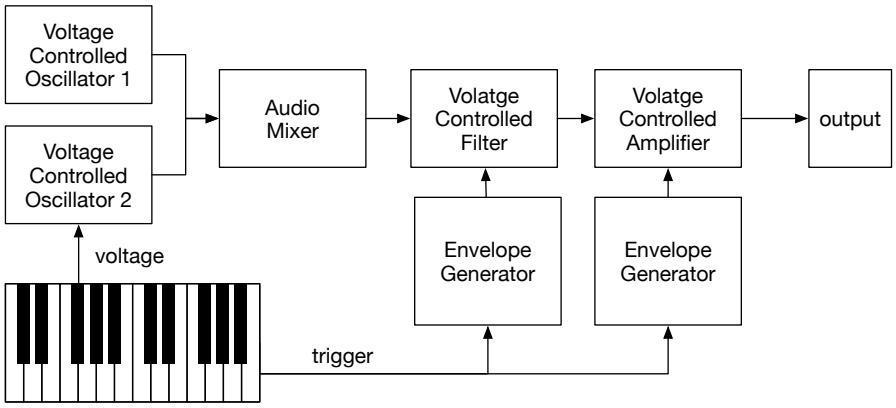

Figure 2.1: Blockschaltbild eines Subtraktive Synthesizer Stimme

Die spannungsgesteuerte Oszillatoren (voltage controlled oscillators, VCO) erzeugen einfache Wellenformen in der Tonhöhe, die auf der Tastatur gespielt wird. Der Benutzer kann in der Regel zwischen einer Kombination aus Sägezahn, Rechteck oder Dreieckwellenform wählen. Die Frequenzen oder die Klangfarbe der Wellenformen können dann durch die folgenden Filter und Amplituden Blöcke moduliert werden.

Der erste Eindruck könnte sein, dass die Modellierung des Oszillators wäre einfach. Ein digital erzeugte Rechteck oder Sägezahn-Wellenform sollte trivial zu implementieren. Die 5 kHz Sägezahn Wellenform beispielsweise würde sich alle 8,82 Samples weiderholen bei einer Taktrate von 44,1 kHz. So würde die Wellenform linear von -1,0 bis 1,0 auf erhöhen, dann zurück auf -1.0 springen auf -1,0 und erneut anfangen. Was bedeutet aber 0,82 Samples in einer diskreten digitalen System? Abbildung 2.2 veranschaulicht das Problem. Die linke Spalte zeigt einen Abschnitt eines idealisierten 5kHz Sägezahnwellenform und der entsprechenden Frequenzinhalt. Oberhalb der Grundfrequenz 5 kHz sind Oberwellen, die weit über 100 kHz hörbar sein werden. Die rechte Spalte zeigt den gleichen Abschnitt eines 5kHz Sägezahnwellenform in einem mit 44,1kHz getakteter diskreter Umgebung. Die Wellenform selbst ist verzerrt und die höheren Harmonischen sichtbar vom 22,05 kHz Nyquist-Grenze nach unten reflektiert.

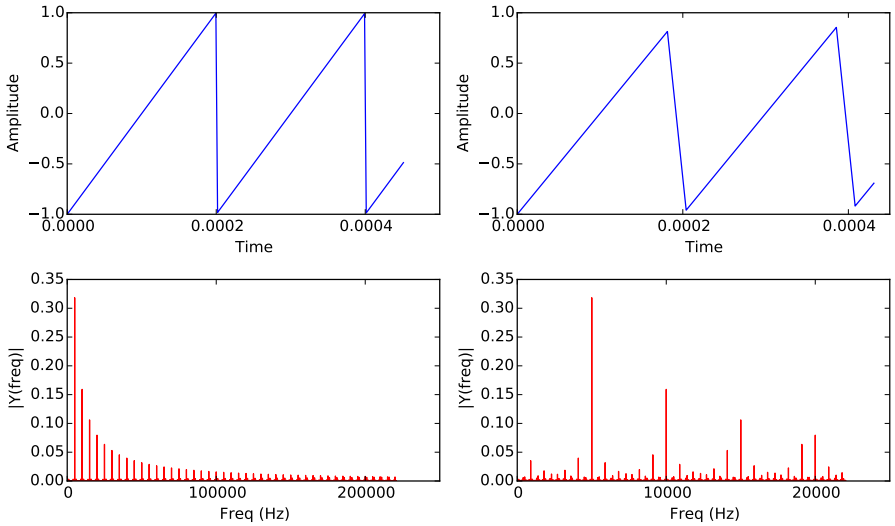

Figure 2.2: Ideal and Aliased 5kHz Sawtooth waveform

Es gibt verschiedene Verfahren zur Beseitigung oder Verminderung von Aliasing. Die effektivste ist es, die Oberwellen mit einer Reihe von Sinus Wellen bis zu der Nyquist-Grenze oder die Hälfte der Abtastrate des Systems zu konstruieren, aber dies ist sehr rechenintensiv. Weniger CPU-intensive Strategien sind Oversampling, Bandlimiting oder andere Alias-Unterdrückungs Methoden [8].

Chapter 3

Anforderungsanalyse

3.1 General Requirements

Die Anwendung besteht aus zwei Komponenten, einen Audio Plug-in, dass auf der haupt CPU-Maschine lauft, und die Rechenknoten, die auf vernetzten SBC Geräte laufen. Das Audio-Plugin leitet MIDI-Steuerung und Audio-Daten zu den Rechenknoten. Die Rechenknoten senden die verarbeiteten Audiodaten zurück an dem Audio-Plugin, das wiederum die Daten an den Host DAW-Anwendung übergibt. Die Gesamtumlaufzeit, einschließlich der Verarbeitung sollte 10 ms nicht überschreiten. Dies ist die maxium erlaubte Latenzzeit für Live-Anwendungen [7].

Die Rechenknoten und Audio Plug-in müssen in sich abgeschlossen sind und Funktionieren ohne dass der Benutzer, externe Libraries, Frameworks oder Servern installieren muss.[1]

3.2 Distributed Processing

Um die Belastung der Host-CPU zu reduzieren sollen Audio-Verarbeitungsaufgaben an externen SBCs über Gigabit-Ethernet verteilt werden. Auf der Host-CPU arbeitet der Audio Plug-in als Master-Knoten. Für den Host-DAW, sollte die Verteilung der Audio-Verarbeitungsaufgaben vollkommen unsichtbar sein. Der Master-Knoten empfängt, MIDI- und Audiodaten von der Host-DAW und gibt die Ergebnisse zurrück wie jede andere Audio-Plugin.

Das Plug-in bekommt Zugriff auf eine kleine Puffer Audiodaten, in sehr kurze regelmässigen Abständen. Das Plugin muss in kurzer Zeit die Daten verarbeiten und zurück an das Host DAW-Anwendung, bevor der nächste Puffer bereitsteht. Dies stellt einige Grenzen, wie Bearbeitungsaufträge verteilt werden kann.

Es gibt verschiedene Arten Verarbeitungsaufgaben zu parallelisieren. Jedes Plug-in-Instanz könnte den gesamten Auftrag an einem Rechenknoten delegieren wie in Abbildung 3.1 dargestellt. Besteht einer Verarbeitungsaufgaben aus mehreren, in se-

[1] Die einzige Ausnahme könnte Zeroconf / Bonjour unter Linux oder Windows sein. Siehe Anhang

rie geschalteten, Verarbeitungsblöcke, dann könnte jedes Block einen Rechenknoten zugeteilt werden, wie in Abbildung 3.2 dargestellt. Speziell bei einer Virtuellen-Synthesizer-Plug-in könnte jede gespielte Stimme seinem eigenen Rechenknoten zugeteilt werden, wie in Abbildung 3.3 dargestellt.

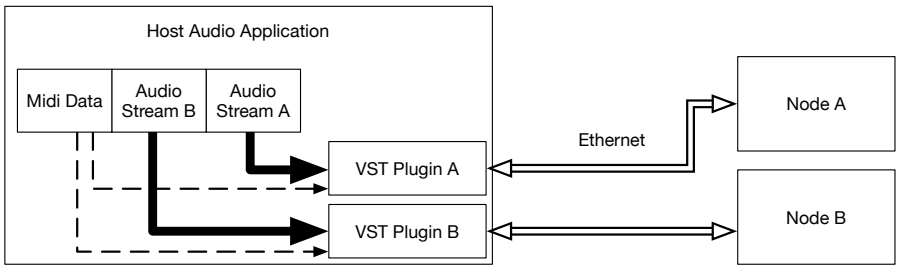

Figure 3.1: Jedes Plugin Verteilt an einen Knoten

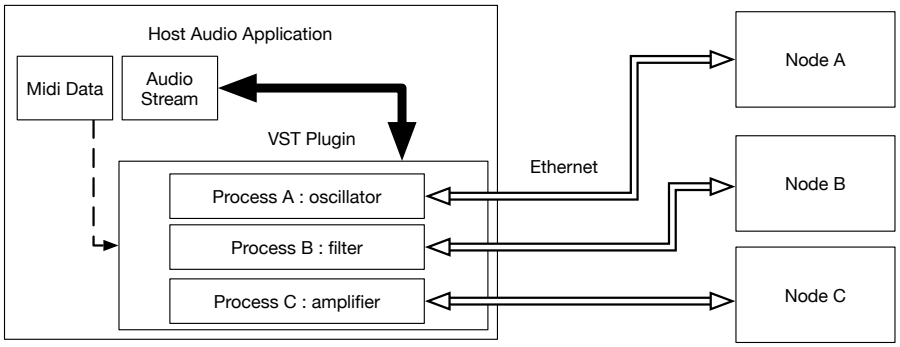

Figure 3.2: Jeder Verarbeitungsblock Verteilt an einen Knoten

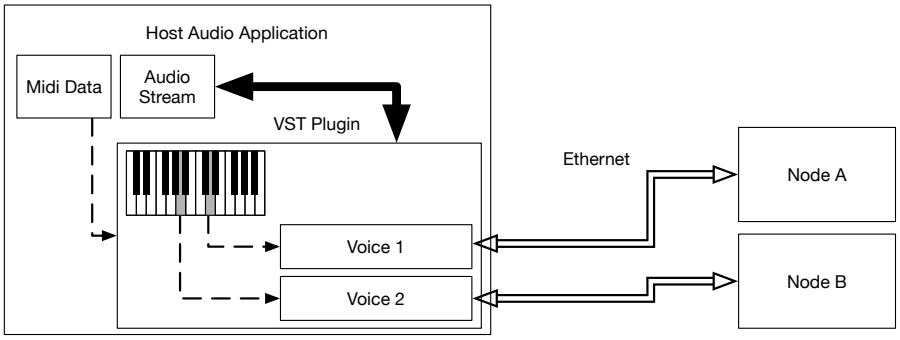

Figure 3.3: Jedes Synth Stimme Verteilt an einen Knoten

Für dieses Projekt wird die zweite Option implementiert aus Testbarkeitsgründen, auch wenn es vielleicht nicht der effizienteste Implementierung ist.

3.3 Audio Plugin

Aus der Sich des Host DAW-Anwendungs muss das Plug-in wie ein ganz normales VST-Plugin aussehen. Das Verteilen der Aufgaben auf vernetzte Rechenknoten muss transparent funktionieren. Das Plug-in wird nicht in einer isolierten Umgebung ausgeführt. Es teilt die Rechenresources mit der Host DAW, so wie eine beliebige Anzahl anderen Plugins. Daher sollte sorgfälltig die Belastung auf der CPU auf das Nötigste minimiert werden.

Audio Plug-ins haben typischerweise einen Zustand, der Parametereinstellungen enthält die vom Audio-Host oder vom Benutzer gesteuert werden können. Der aktuelle Zustand muss den vernetzten SBC Modulen auch mit geteilt werden. Dies ist notwendig, damit ein Audio Plug-in, der Zustand einer fortlaufender Prozess, nahtlos einer vernetzte Rechenknoten überreichen kann. Es könnte auch verwendet werden, um eine vernetzte Rechenknoten durch Umschalten des aktiven Zustandes die Verarbeitung für mehrere Audio Plug-ins zu übernehmen.

Das Audio-Plugin stellt die folgenden Anforderungen:

- lauffähig als Echtzeit VST Audio Plug-in in einer Standard DAW Anwendung
- lokalisieren und verbinden mit einer passende Rechenknoten im Netzwerk
- überreichen von MIDI- und Audio-Daten vom Host DAW-Anwendung, zu den vernetzten Rechenknoten
- empfängt Audiodaten von den Rechenknoten und übergibt diese zurück an der DAW-Anwendung

- ist keinen entsprechenden Knoten im Netzwerk vorhanden, muss die Audioverarbeitung Lokal statfinden
- die Audio Plug-in muss sienen aktuellen Zustand an den Rechenknoten übermitteln können

3.4 Remote Processing Node

Die Rechenknoten sind Anwendungen, die auf den vernetzten SBC-Geräten laufen. Je nach Art der implementierte Parallelisierung in der Audio Plug-in wird das Rechenknoten die gesamte Verarbeitungsaufgabe oder einfach nur ein Teil der Aufgabe bekommen. Für dieses Projekt werden die Rechenknoten als Reihe von Prozessoren in eine übergeordnete Anwendung umgesetzt werden. Die übergeordnete Anwendung wird eine Socket-Listener implementieren, welche den zugeorneten Rechenknoten aufrufft sobald ein Audio Verarbeitungsaufgabe eintrifft. Die Verfügbarkeit, Art und Lage der Rechennoten und das entsprechende Socket wird über das Netzwerk via Bonjour / Zeroconf übertragen.

Die Rechenknoten sollten stateless sein. Jeder Zyklus des Audio-Verarbeitungsalgorithmus sollte nur den Zustandsdaten des entsprechenden Pakets berücksichtigen müssen. Werden die Zustandsdaten durch den Verarbeitungsalgorithmus geändert müssen sie an das Audio Plug-in zurück gesendet. Damit soll sichergestellt werden, dass ein Rechenknoten zu jeder Zeit in ein laufendes Session einspringen ohne etwas über vorherrige Events wissen zu müssen. Es hat auch den zusätzlichen Vorteil, dass ein Verarbeitungsknoten in der Lage wäre, Aufträge für mehrere Instanzen eines bestimmten Plug-ins verarbeiten zu können.

Die Rechenknoten müssen die folgenden Anforderungen erfühlen:

- gibt seine Verfügbarkeit und Standort im Netzwerk via Bonjour / Zeroconf bekannt
- akzeptiert Steuerdaten von der Audio Plug-in
- verarbeitet eingehende Audiodaten und MIDI-Daten von der Audio Plug-in.
- schickt die verarbeitende Daten sofort an das Audio Plug-in zurück.

3.5 Software Requirements

In Echtzeit-Audio-Anwendungen das Einhalten zeitlicher Vorgaben ist kritisch. Dies mag selbstverständlich klingen, aber für einen Programmierer bedeutet es den Verzicht auf viele Komforten der modernen Programmierung, besonders das Arbeiten mit High-Level Programmier-Sprachen wie Java oder Python. Echzeit-Audio ist ein ”Hard Realtime” Aufgabe und dafür eignen sich nur Low-Level Sprachen wie C oder C++. Auch die meisten Audio-Anwendungsschnittstellen und Bibliotheken wie die VST SDK sind für C/C ++.

Professionelle Audio-Anwendungen laufen in der Regel auf Mac OSX oder Windows-Betriebssysteme, daher muss der Audio Plug-in auf diesen Systemen Kompilierbar sein. Die Rechenknoten, welche auf den SBC-Geräte laufen, müssen für Linux Kompilierbar sein. Doch beide Anwendungen sollten ein gross teil ihre Quellcode teilen können, da ihre Aufgaben sich grossteils überschneiden.

Es gibt viele C ++ Bibliotheken und Frameworks, welche Cross-Plattform-Kompatibilität ermöglichen und nebenbei auch den Programmierer Zugriff auf High-Level Konstrukte wie automatischen Speicherbereinigung mittels intelligente Zeiger und Referenzzählung, die das Programmieren in C++ einfacher machen.

3.5.1 Evaluierte C++ Frameworks

Boost ist die beliebteste plattformübergreifende C ++ Framework. Viele Boost Funktionalitäten wurden sogar in die C ++ 11 Standard-Bibliothek hinzugefügt. Andere Frameworks wie Cinder und Open Frameworks bieten viele High-Level Funktionen, um schnell interaktive medien-reiche Anwendungen zu programmieren. Zwei Bibliotheken die hervorzuheben sind, JUCE und WDL, beiten besonders für Audio Plug-ins und DAW-Anwendungen zugeschnittene Funktionen und Klassen. Von diesen beiden hat Juce eine viel größere Benutzergemeinschaft (einschließlich Anbieter von DSP-basierten Audio-Coprozessoren).

Software Framework Kriteria:

- Plattformübergreifend für OSX, Linux und Windows
- Bietet Hohe Konstrukte wie intelligente Zeiger
- Unterstützung für plattformübergreifende Audiointegration
- Sollte gut dokumentiert sein und eine aktive Benutzergemeinschaft haben
- Bietet plattformübergreifende Netzwerkzugriff

Framework	High Level Utilities	Audio Utilities	Network Utilities	VST Utilities	Community
JUCE	ja	ja	ja[2]	ja	gross
WDL	ja	ja	nein	ja[3]	klein
Open Frameworks	ja	ja	ja[4]	nein	gross
Boost	ja	nein	ja	nein	gross
Cinder	ja	ja	ja	nein	klein
LibSourcey	nein	nein	ja	nein	nein
Qt	ja	nein	ja	nein	gross

[2] basic socket management classes
[3] enabled using one of the additional iplug libraries
[4] the ofxNetwork addon allow simple management of TCP or UDP sockets

Auf Grundlage der Kriterien Vergleich und früheren Erfahrungen bei andere Projekten wurde für diese Projekte JUCE für die Implementierung ausgewählt.

Chapter 4

Implementation

4.1 Architecture

Abbildung 4.1 zeigt einen Überblick über die Audioverarbeitungsumgebung. Der Benutzer des Systems arbeitet mit der DAW-Anwendung welches auf dem Host-CPU läuft, und kann einen Audio Plug-in auf eine bestimmte Audiospur aktivieren. Ein einzelner Audio Plug-in kann einer oder mehrere Audio Prozessoren enthalten. Prozessoren, die fähig sind ihre Aufgaben zu Verteilen suchen nach einem entsprechenden Rechenknoten auf einem Vernetztes SBC-Gerät.

Die Geräte werden über Gigabit-Ethernet vernetzt. Es wird angenommen, dass das Netzwerk nicht für sonstiges signifikanten Verkehr benützt werden.

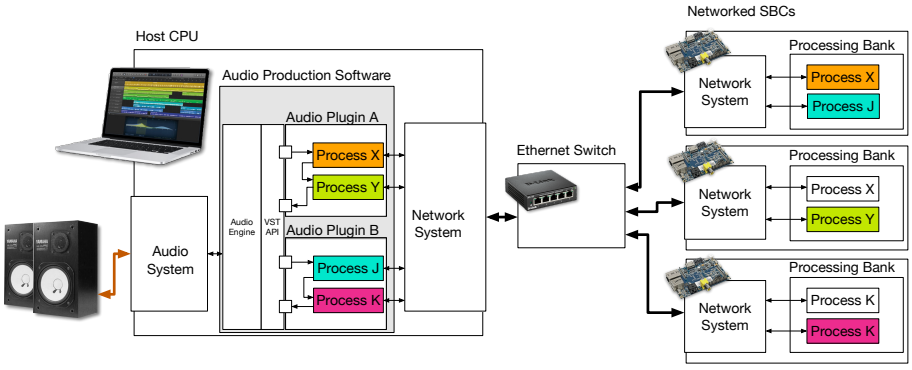

Figure 4.1: Architectural Überblick

4.1.1 Host CPU

Abbildung 4.2 zeigt die Komponenten welche für Echtzeit-Audio auf der CPU benötigt werden. Audio-Hardware benötigt, einen kontinuierliches genau-getaktetes Daten-

strom um es den Digital-Analog-Wandlern bereitzustellen. Dies geschieht durch periodisches Abfragen des Betriebssystems über die Hardware-Interrupts. Die angeforderte Puffergrße kann wenige Abtastwerte beinhalten und die Abfrageintervall weniger als 1 ms. Dies ist abhängig von der benützten Hardware und der Audio-Treiber.

Das Betriebssystem bietet eine Abstraktionsschicht in Form einer API für die Anwendungssoftware. Dies gibt der Anwendungs-Software einer einheitlichen Schnittstelle, unabhängig von der Marke der Audio-Hardware und Treiber.

Das VST-API ist eine weitere Abstraktionsschicht, die weiderum eine einheitliche Schnittstelle für Plugin-Anbieter anbietet, unabhängig von der OS. Aber VST ist nicht die einzige Plugin API. Die JUCE Bibliothek bietet einen eigenen Plugin-API, die einfacher ist und abstrahiert die Unterschiede zwischen verschiedene Plugin-APIs.

Die Datenanabfrage, ausgelöst duch Interrupts von der Audio-Hardware, werden durch das Betriebssystems an die DAW-Anwendung durch Rückruffunktionen weitergeleitet. Das DAW hat zuvor, beim aufstarten, die entsprechende Rückrufffunktionen an das Betriebssystems registriert. Die DAW-Anwendung wiederum lietet die Datenabfrage, ebenfalls duch die vom VST Spzifizierten Rückruffunktionen an alle aktiven Plug-ins weiter.

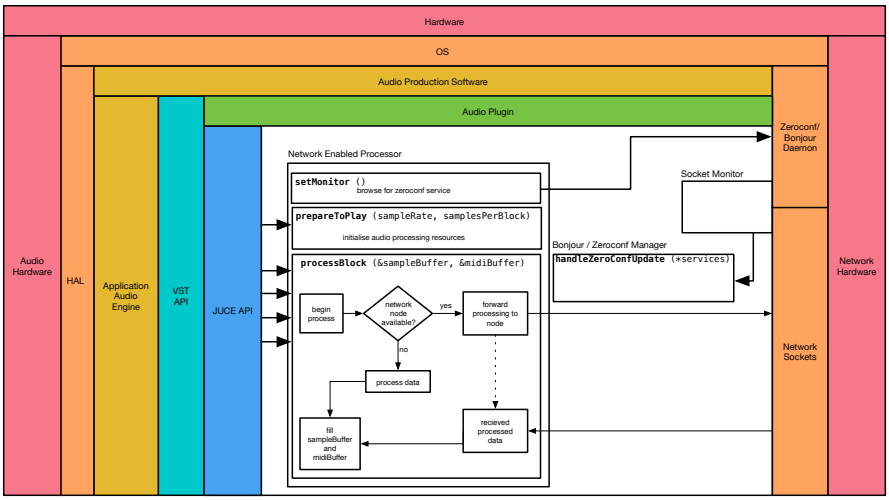

Figure 4.2: Host CPU Überblick

Die für dieses Projekt umgesetztes Audio Plug-in hat mehrere Netzwerkprozessoren aktiviert, in Abbildung 4.2 wird nur eine dargestellt als Beispiel. Wenn das Plug-in von der Host-Software instanziiert wird, instanziiert es weiderum jeder seiner internen Prozessoren. Die Prozessoren rufen je den Bonjour / Zeroconf-Daemon des Betriebssystems auf, um eine passende im Netzwerk registriertes Rechenknoten zu finden. Der Aufruf übergibt das Daemon eine Socket über eine benachrichtigung zurück schicken kann wenn es einen passendes Service im Netzwerk gefunden hat.

Die Bonjour / Zeroconf-Daemon übergibt dem Audio Plug-in mittels der Socket einer Liste der gefundenen Services. Die Audio Plug-in durchsucht die Liste nach einer freien Rechenkonten. Das ausgewählter Rechenknoten wird in das ”activeNode” Variable gespeichert.

Wenn eine Anfrage für Audio-Daten von der DAW-Anwendung an das Plugin übergeben wird, dann wird der ”processBlock” Funktion des Plugins aufgerufen. Das Plugin ruft wiederum die ”processBlock” Funktionen jedes seiner internen Audio Prozessoren nacheinander. In Abbildung 4.2 wird dies vereinfacht dargestellt, mit der ”processBlock” Funktion von einem einzigen Audio Prozessor. Der ”processBlock” Funktion wird einen Referenz auf die aktuelle zu verarbeiten Audiopuffer und MIDI-Puffer übergegeben. Der Audiopuffer enthält die einzelnen Audioabtastungen für jeden Kanal als Float-Werte. Die MIDI-Puffer enthält Performance-Daten wie der Eintrittszeit und Tonhöhe der gespielten Noten.

Innerhalb der ”processBlock” Funktion prüft der Prozessor, ob ein Rechenknoten im ”activeNode” gespeichert ist. Wenn ja, dann leitet er sofort die MIDI- und Audiodaten sowie die eigene Zustandsdaten an die Rechenknoten und wartet auf die Antwort. Wenn die Antwort eintrifft werden die Daten zurück in die entsprechende Puffer kopiert. Die DAW-Anwendung geht dann weiter die nächsten Plug-ins abfragen, bis die gesamte Verarbeitungskette abgeschlossen ist. Die resultierenden Puffer wird an die Audiohardware über die HAL API geschickt.

4.1.2 Vernetzte SBCs

Abbildung 4.3 zeigt die Komponenten der vernetzten SBC-Geräte. Eine Verarbeitungsanwendung kann eine beliebige Anzahl von Prozessoren erhalten, die jeweils ihre Dienste über den Zeroconf/Bonjour-Daemon im Netzwerk anbieten. Das Zeroconf/Bonjou-Daemon wird als teil des Betriebsystems hier dargestellt. Die Zeroconf/Bonjour-Daemon sendet die Verfügbarkeit, der Verarbeitung-Art und die Portnummern von jedem der verfügbaren Prozessoren.

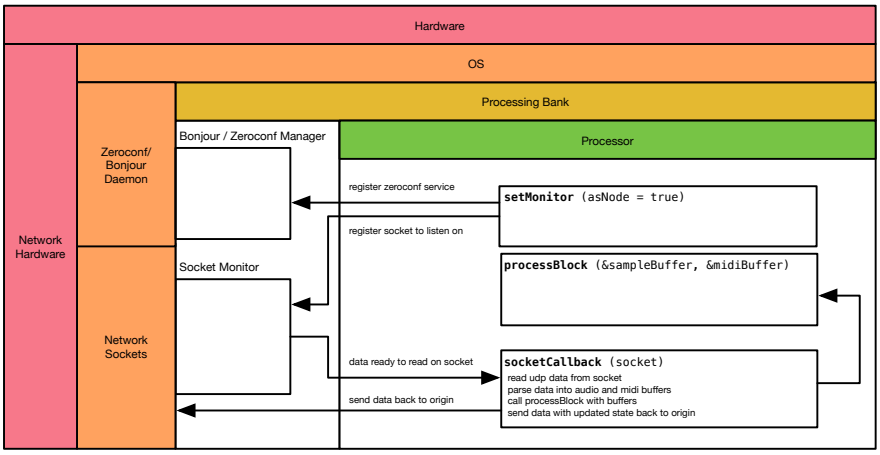

Figure 4.3: SBC Processor Überblick

Der Prozessor übergibt eine offene Socket an einen Socket-Monitor und registriert sich als dazugehörende Socket-Zuhörer-Objekt. Die Socket-Monitor besitzt eine Reihe von Sockets und einen Verweis auf jeden Zuhörer. Es führt eine ”select” Funktion auf der Reihe von Socket und wartet. Wenn Daten an einem der Sockets eintreffen, erwacht der ”select” Funktion und der Socket Monitor benachrichtigt den entsprechenden Socket-Zuhörer-Objekte über eine Rückruffunktion, das Daten zu verfügung stehen.

Der entsprechende Prozessor wird über die Rückruffunktion benachrichtigt, liest die Daten, entpackt die Audio-, Zustands- und MIDI-Daten, und führt den processBlock Funktion aus. Die Ergebnisse und aktualisierte Zustandsinformation werden am Sender zurück geschickt.

4.2 Software Komponenten

Die Anwendung ist in Komponenten (Klassen) aufgebaut welche jeweils die Verantwortung für einen Teilbereich entkapselt. Jedes Modul wurde in einem ”pseudo Test Driven” Methodik entwickelt. Die Tests wurden nicht zwingenderweise entwickelt, bevor der entsprechende Code geschrieben wurde, aber auf jeden Fall kurz danach. Dies führte zu stabilen Code, dass in jeder Iteration geprüft werden konnte, isoliert von anderen Komponenten.

Die Interaktion mit einer Klasse wird in einer dazugehörenden ”Zuhöhrer” Klasse definiert. In der ”Zuhöhrer” Klasse sind Rückruffunkionen spezifiziert, die einer Client-Klasse zu implementieren hat. Ein Client-Klasse kann von der Listener-Klasse erben und die Rückruffunktionen überladen.

4.2.1 Socket Monitor

Die Monitor-Klasse wurde entwickelt, um Sockets effizient zu verwalten. Client-Klassen erben von der FileDescriptorListener Klasse, um benachrichtet zu werden, wenn Daten verfügbar sind auf einer dazugehörendes Socket.

Die Monitor-Klasse hat einen Thread, welches eine select-Funktion aufruft. Die select-funktion wird einen Array von Sockets als Filedescriptors übergeben, dann blockiert es. Sobald Daten auf einer der Sockets eintreffen, fird die select-Funktion wieter fahren. Der Thread geht prüfft in einer Schleife alle Filedescriptors bis er das richtige gefunden hat und benachrichtigt das entsprechende FileDescriptorListener Objekt.

Eine der Sockets welches an die select-Funktion übergeben wird der Monitor-Klasse eigenen control_listener Socket. Die Monitor-Klasse kann ein Signal an dieser Socket schicken, um den blockierten Thread aus dem select-Funktion zu wecken und den Zustand aktualisieren. Die Monitor-Klasse tut dies, wenn eine neue Client sich registriert bzw. sich entfernt, und wenn der Monitor-Klasse heruntergefahren werden muss.

4.2.2 ZeroconfManager

Zeroconf (kurz für Zero Configuration Netzwerk- und auch als Bonjour auf OSX bekannt) ist einer Technologie, die es einen Softwaredienstleistung erlaubt ihre Verfügbarkeit und Standort über einen Netzwerk bekannt zu machen. Drucker und Multimedia-Geräte verwenden Zeroconf z.B., damit sie für andere Netzwerkteilnehmer leicht zu finden und nutzen sind.

Bonjour auf OSX und den compatibile Funktionen von Avahi unter Linux definieren eine Rückruff-basiertes API, die zu einem Bonjour oder avahi Daemon auf dem OS-Schnittstellen. Die Bonjour-API ist in C. Die ZeroConfManager Klasse kapselt die Kommunikation an die bonjour oder avahi Daemon in einer objektorientierten Weise. Clients, die mit dieser Klasse Schnittstelle muss zu erweitern und überschreiben Sie die ZeroConfListener Klasse möchten, sich zu registrieren, sie mit der Klasse zusammen mit der Service-Tag für sie erhältlich sind.

4.2.3 DiauproMessage

Die DiauproMessage Klasse verwaltet die Serialisierung und Deserialisierung von Daten zu, beziehungsweise von, Datagramm-Pakete. Die Datagramme selbst bestehen aus einer Header mit festen Länge, und Nutzdaten mit variable Länge, welche die Audio-, MIDI- und Zustanddaten enthält.

Der Header wird wie folgt definiert:

```
struct diapro_header {
    uint16 sequenceNumber;
    uint16 numSamples;
    uint16 numChannels;
    double sampleRate;
    uint16 audioDataSize;
    uint16 midiDataSize;
```

```
    uint16 stateDataSize;
    double cpuUsage;
};
```

Die DiaproMessage Klasse bemüht sich, bestehende zugewiesenen Speicher zu verwenden, wenn möglich. Instanzen DiauproMessage sollte nicht alloziert werden in den Thread, der die Audio-Verarbeitungsroutinen aufruft. Sie sollten lieber voralloziert werden mit genügend Speicher, um einen UPD Datagram mit maximale Größe (64 KB) zu speichern. Dieser Instanz soll dann für jeden Aufruf-Zyklus wiederverwendet werden damit Speicher nicht neu alloziert werden muss während der Zeit kritische Prozess.

4.2.4 DiauproProcessor

Die DiauproProcessor-Klasse ist die Basisklasse, von der anderen Prozessoren erben sollen. Die DiauproProcessor-Klasse erbt wiederum von der JUCE Audioprocessor-Klasse, die alle Funktionen der verschiedenen Audio-Plugin-Formate kapselt. Klassen, die von Audioprocessor erben können direkt in ein VST-Plugin gewickelt (wrapped) werden.

Eine Klasse, die von DiauproProcessor erbt muss die Funktionen ”localProcess” und ”getServiceTag” überladen. Die ”localProcess” Funktion führt die Audio-Verarbeitung aus. ”getServiceTag” gibt einer Zeichenfolge zurück, die verwendet wird, um seine Dienste im Netzwerk bekannt zu machen oder danach zu suchen.

Klassen, die von DiauproProcessor erben können instanziert und eingerichtet werden so, dass sie entweder in der Audio Plug-in oder in der Rechenknoten laufen. Im Plug-in ausführt wird es ein entsprechendes Rechenknoten im Netzwerk suchen. Falls nichts passendes gefunden wird, wird das ”localProcess” Funktion lokal verwendet. Es sie finden eine vernetzte Service für alle ankommenden Audio- und MIDI-Daten werden auf diesen Dienst weitergeleitet.

Wenn in einem Rechenknoten ausgeführt, werden Klassen, die von DiauproProcessor erben sich im Netzwerk anmelden und warten, um ankommende Verarbeitungsanforderungen.

Beispiele für Klassen, die von DiauproProcessor erben sind DiauproVCOProcessor und DiauproVCAProcessor. Sie werden unten in Detail beschrieben.

4.2.5 DiauproVCOProcessor

Die DiauproVCOProcessor erbt von DiauproProcessor und implementiert die Oszillatorblock eines virtuellen synthsizer. Es erzeugt einen Klang in der Tonhöhe einen gespielten MIDI-Note. Die einzigen Funktionen aus DiauproProcessor, die überladen werden müssen, sind ”localProcess”, ”getState”, ”getStateSize” und ”getServiceTag”.

”localProcess” wird wie folgt implementiert:

```
1  void DiauproVCOProcessor::localProcess(AudioSampleBuffer &buffer,
2                                         MidiBuffer &midiMessages,
3                                         void* state)
4  {
5    processState = *(vco_state*)state;
```

```
6     int sampleNr;
7     int nextEventCount = -1;
8     MidiBuffer::Iterator midiEventIterator(midiMessages);
9     MidiMessage nextEvent;
10    bool hasEvent;
11
12    for(sampleNr = 0; sampleNr < buffer.getNumSamples(); sampleNr++)
13    {
14      if(nextMidiEventCount < sampleNr)
15      {
16        hasEvent = midiEventIterator.getNextEvent(nextEvent,nextEventCount )
17      }
18      if(hasEvent && nextEventCount == sampleNr)
19      {
20        if(nextEvent.isNoteOn())
21        {
22          processState.voice_count++;
23          processState.frequency = MidiMessage::getMidiNoteInHertz(nextEvent.getNoteNumber());
24          double cyclesPerSample = processState.frequency / getSampleRate();
25          processState.step = cyclesPerSample * 2.0 * double_Pi;
26
27        } else{
28          processState.voice_count--;
29        }
30      }
31      if(processState.voice_count > 0)
32      {
33        const float currentSample = (float) (sin (processState.phase) * processState.level);
34        processState.phase += processState.step;
35        for(int i = 0; i < buffer.getNumChannels(); i++)
36        {
37          float oldSample = buffer.getSample(i, sampleNr);
38          buffer.setSample(i, sampleNr, (currentSample + oldSample)*0.5);
39        }
40      }
41    }
42  }
```

Wenn die Funktion aufgerufen wird, wird die aktuelle Audio-Puffer übergeben, ein Puffer von MIDI-Events für den korespondierende Zeitrahmen, und ein Zeiger auf einen Block von Daten, die verwendet werden, um den Zustand zu speichern, damit es zwischen Aufrufen beibehalten werden.

Wird diese Funktion innerhalb eines Rechenknotens ausgeführt dann werden Audio-, MIDI- und Zustandsdaten aus einem DiauproMessage, als UDP-Paket von der Master-Audio-Plugin gesendet, entnommen.

Im obigen Beispiel wird einer Sinuswelle generiert, indem die Sinus Wert für jedes Element in der Audio-Puffer berechnet wird.

4.2.6 DiauproVCAProcessor

Die DiauproVCAProcessor moduliert die Lautstärke des Signals über Zeit. VCA ist die Abkürzung für Voltage Controlled Amplifier, dies bezieht sich auf den Verarbeitungsblock in einer virtuellen Synthesizer, der für die ”Form” eines Klangs zuständig ist.

Eine Glocke, zum Beispiel, hat einen schnellen und lauten Anfangs-Ton, der über langen Zeitraum abklingt. Im Gegensatz dazu ist ein Streicher Klang, wie eine Geige, hat, wenn sanft gespielt, eine langsame Anfangs-Ton, der plötzlich endet. Die VCA ist für die Gestaltung der Amplitude eines Klang zuständig.

Chapter 5

Conclusion

Chapter 6

Glossary

MIDI

The Musical Instrument Digital Interface specification, first introduced in 1983 defines an 8-bit standard for encoding and transmitting music notes. It's original purpose was to allow one keyboard based synthesizer to controll other music devices. [2] Allthough the specification also describes the hardware and wiring for daisy chaining instruments in a midi "network" most midi communication today transmitted via usb or virtually between audio software.

Zeroconf

Zeroconf is a network service discovery protocl. It is also known as Bonjour, and occasionally refered to as Rendezvous, from the Apple implementations. Howl and Avahi are alternative open source Zeroconf implementations for Linux. Application can use Zeroconf to register or browse for service on a network without the need for a user to provide a specific IP Adress or port number. [4]

AES67

Latency

VST

SBC

AVB

Bibliography

[1] Nicolas Bouillot, Elizabeth Cohen, Jeremy R. Cooperstock, Andreas Floros, Nuno Fonseca, Richard Foss, Michael Goodman, John Grant, Kevin Gross, Steven Harris, Brent Harshbarger, Joffrey Heyraud, Lars Jonsson, John Narus, Michael Page, Tom Snook, Atau Tanaka, Justin Trieger, and Umberto Zanghieri. Aes white paper: Best practices in network audio. *J. Audio Eng. Soc*, 57(9):729–741, 2009.

[2] Richard Boulanger and Victor Lazzarini. *The Audio Programming Book*. The MIT Press, 2011.

[3] Eli Brandt and Roger B Dannenberg. Low-latency music software using off-the-shelf operating systems. 1998.

[4] Stuart Cheshire and Daniel H. Steinberg. *Zero Configuration Networking: The Definitive Guide*. O'Reilly Media, Inc., first edition edition, 2006.

[5] Wolfgang Fohl and Julia Dessecker. Realtime computation of a vst audio effect plugin on the graphics processor. In *CONTENT 2011, The Third International Conference on Creative Content Technologies,*, pages 58–62, 2011.

[6] Vincent Goudard and Remy Muller. Real-time audio plugin architectures, a comparative study. pages 10, 22, 9 2003.

[7] AES Standards Committee Gross, Kevin. *AES standard for audio applications of networks - High-performance streaming audio-over-IP interoperability*. Audio Engineering Society, Inc., 60 East 42nd Street, New York, NY., US., 2013.

[8] Vesa Välimäki and Antti Huovilainen. Oscillator and filter algorithms for virtual analog synthesis. *Computer Music Journal*, 30(2):pp. 19–31, 2006.

Chapter 7

Appendix A

7.1 Compiling the Source Code

Download the Juce C++ Library from GitHub
github.com/julianstorer/JUCE
Additionally download and install the DrowAudio Juce Module Extensions
github.com/drowaudio/drowaudio.git

7.2 Bonjour

Instructions for installing bonjour:
If bonjour / zeroconfig is not installed on the SBC device you will get a "fatal error: dns_sd.h: No such file or directory" error. You can fix this by installing the Avahi library
http://www.avahi.org/download/avahi-0.6.31.tar.gz
Requirements: sudo apt-get install intltool sudo apt-get install libperl-dev sudo apt-get install libgdbm-dev sudo apt-get install libdaemon-dev sudo apt-get install pkg-config sudo apt-get install libgtk2.0-0 sudo apt-get install libdbus-1-dev
sudo apt-get install intltool libperl-dev libgdbm-dev libdaemon-dev pkg-config libgtk2.0-0 libdbus-1-dev
./configure –prefix=/usr –enable-compat-libdns_sd –sysconfdir=/etc –localstatedir=/var –disable-static –disable-mono –disable-monodoc –disable-python –disable-qt3 –disable-qt4 –disable-gtk –disable-gtk3 –enable-core-docs –with-distro=none –with-systemdsystemunitdir=no
make make install
sudo cp /usr/include/avahi-compat-libdns_sd/dns_sd.h /usr/include/

7.3 Evaluated Frameworks

WDL : http://www.cockos.com/wdl/ (+iplug library)
Juce : http://www.juce.com

Open Frameworks : http://openframeworks.cc
Boost : http://www.boost.org
Cinder : http://libcinder.org
LibSourcey : http://sourcey.com/libsourcey/
Qt : http://www.qt.io