Distributed Audio Processing

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

October 7, 2015

Abstract

In modern profesional music studios, the computer has become responsible for tasks that were previously performed by dedicated equipement. Mixing boards, effect processors, dynamic compressors and equalizers, even the instruments themselves, are all available as software. To elivate 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.

Declaration

I declare that..

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

The purpose of this semester project is to design a software based music synthesiser that will run on a network of low cost banana pi devices. Limitations in polyphony will be alleviated by adding a new device to the network. In order to be compatible with existing music recording and composition applications the software will include a VST Plugin that allows music software to send MIDI commands to, and receive audio data from the software. All data communication between the VST Plugin and the Banana PI audio generation software will be handled via ethernet. The VST Plugin will send control data information such as pitch, volume, length, and other expression data. The Banana PI will stream back the generated audio data, as well as necessary metadata so the VST plugin can properly collect and prepare the audio data for the host software.

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?

1.4 Resultate

1.5 Glossary of Terms

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

Chapter 2

Einführung ins Thema

2.1 Background

An audio engineer's typical job is to manage the balance of multiple tracks of audio signals. The dynamic range of a signal can be compressed, in order to give quiter passages more presence. Loud peaks can be limited to balance the overall loudness of a musical piece. Using equalizers an audio engineer can make enhance or supress specific frequencies of a track to make it more present in a mix. Effects like reverb, echo, or chorus can be used to give a track more space in a mix effecting the mood or ambience of a music piece. It is typcial that each track in a recording session will be processed by a chain of several specific audio processors.

20 years ago the equipment needed for this kind of processing filled large racks. Today all of these tasks run as plugins on the CPU.

In 1996 Steinberg GmbH, the developers of Cubase, a popular audio production application, released the VST interface specification and SDK. [9] The VST plugin standard was special because it allowed realtime processing of audio in the CPU and it allowed other developers to programm plugins which could be run from within Cubase. The VST plugin standard quickly had widespread industry acceptance and was adopted by most developers of audio production applications. Although alternative standards exist, VST is still the most widely adopted crossplatform standard.

The number of realtime plugins that could run on a CPU was limited by several factors, hard disc access speeds, bus speeds, amount of ram, and OS schedulers for instance [3]. User's didn't expect to be able to run more than 10 plugin instances at a time. Simply playing back multiple tracks of digital audio in realtime was so taxinig on the CPU that an application's graphical interface would quickly become unresponsive.

Today it's possible to playback hundreds of channels of audio and hundreds of plugins in realtime. But as the performance threshhold has risen, so to have the expectations. The algorithms driving today's plugins are much more complex that those from 1996. Plugins are available today that model physical systems or emulate the analog circutry of popular vintage synthsizers. So, even though CPU performance has increased significantly, it's still easy to reach the limits, especially with the more com-

plex high qualtiy plugins.

Serval DSP based systems exist that can alleviate the load on the CPU much in the same way that GPU accelerator cards work. Audio processing jobs are delegated to external specialized hardware via PCIe or Thunderbolt interfaces. However, these DSP based accelerators are proprietary and expensive. Developing plugins for a DSP chip is also significatly more complex than developing for the CPU.

2.2 Realtime Audio Plugins

Music composition and production is typically done with the assistance of a music sequencing application. Midi events and audio recordings are arranged as tracks that can be mixed, edited, and processed. In order to make changes undo-able edits are made in a non-destructive fasion, calculated dynamically in realtime during audio playback. The original audio data is always preserved. The user can change the parameters of an effect or process in realtime and experiment with various parameters without fearing that the original audio recording might be permenantly altered.

A music sequencer or audio production application will usually include several built in realtime effects that a user can apply to an audio track. In addition to the built in options all professional applications will also be able to load 3d party effect plugins. Depending on the platform and vendor one or several available plugin standard will be implmeneted, the most common standard being Steinberg's VST standard.

Regardless of the standard most audio plugins function in a similar fashion. The host application will periodically poll the plugin via a callback, providing access to the source audio data stream and expecting the plugin to return the processed data.

Audio plugins can also provide a gui to the user that allows processing parameters to be modified, saved, and sequenced as well. This might be the cutoff frequency of a low pass filter, or the delay time of a reverb effect, for example.

On the Windows platform VST plugins are compiled to Dynamic Link Libraries, on Mac OSX they are Mach-O Bundles. The native apple Audio Unit plugins are also compiled as Mach-O bundles, they have almost identical functionality, but differ in their API implementation. Other alternative plugin formats are Avid's RTAS and AAX plugin formats, Microsoft's DirectX architecture, or LADSPA, DSSI and LV2 on Linux. From a programmer's point of view audio plugins are always compiled as dynamically loadable libraries that stricly conform to a format's specific API. The host application can load then at run time and stream audio data through them [6].

Additionally Realtime Audio Plugins, as the name implies, must be able to complete thier tasks fast enough to comply with realtime audio requirements. How fast is fast enough? Well, that depends how you define "real time". In audio applications, real time is defined in terms of an audio system's latency. The total delay between the time an audio signal enters the system (at the analog to digital converter for example), is processed, and leaves the system (at the digital to analog conterver) is the latency. If a musician is performing live and simultaneously hearing the result of the performance after being processed digitally, the system's latency must be low enough to feel instantaneous. The maximum acceptable latency is considered to be around 10ms [7]. Any higher and the latency becomes disturbing and not acceptable for live performance

applications.

Any process will introduce some ammount of delay. Some processes, like those that rely on an FFT for example, need to work on a group of samples, introducing additional latency. Within the audio processing function, the programmer must take care not to introduce any unneccessary or uncalculateable delays. Examples for things to avoid are memory allocations, conditional expressions inside loops that might break pipe-lining optimizations [6], or updating the graphic interface directly.

2.3 Audio Over Ethernet

Sending audio data over a network is not new. Sending audio data in ”realtime” is also not new. The IETF (Internet Engineering Task Force) RFC 3550 Describes the Real-time Transport Protocol for delivering audio and video in real-time over IP networks. RTP however is mostly concerned with transmitting a few channels of media as quickly as possible over IP networks with lot of other traffic, reducing jitter (variations in latency) and providing Quality of Service strategies to achieve ”acceptable” audio and video quality for streaming and conferencing purposes.

Other very recent specifications such as AVB[1] and AES67[2] build on top of RTP and add more mechanisms to guarentee acurate timing and synchronization across a network for professional audio applications. The synchronisation is important in these standards because they are concerned with driving audio hardware attached to different hosts on a network.

Hardware synchronization and jitter management are not relevent to this project because we are not concerned with external audio hardware. The goal of this project it to utilise external CPUs as audio coprocessors connected via gigabit Ethernet. Even so, the AVB and AES67 standards offer many insights into how to optimize data transmission for low latency applications and they also offer a proof-of-concept that it is possible. The AES67 defines guidlines that can achieve latencies well below 1 ms for hundreds of simultaneos channels of high quality audio. This is much faster than the legacy PCI and Firewire rates used in many DSP based coprocessing systems [1].

2.4 Single Board Computers

The popularity of the Raspberry Pi has spurned a whole industry around single-board computers (SBCs). Based on hardware used in mobile phones, these small low power devices are extremely popular because they are inexpensive and easy to use. The biggest advantage of SBCs compared to other embeded devices ist that they can run the Android and Linux operating systems, allowing them to be programmed using the same tools available on desktop computers.

[1] Audio Video Bridging refers to a set of IEEE standards that allow time-synchronized low latency streaming services

[2] AES67,created by the Audio Engineering Society, defines standard that allow existing low latency streaming systems to interoperate. AES67 does not define any new technologies but attempts to set a lowest common denominator by which existing standards can be compatible.

Recent higher-end SBCs even come equiped with gigabit Ethernet and Dual and Quad Core CPUs running at rates well over 1GHz. If we compare these systems to the 450MHz G3 PPC systems that the first VST Software was available for we can expect that the newer high-end SBCs should be excellent audio coprocessors.

2 SBCs are worth special consideration because they potentially offer even better performance as audio coprocessors. The Parallella Board has a 16-Core Epiphany coprocessor that could be used to perform audio processing in parallel. Standard frameworks such as OpenCL, MPI, or OpenMP can be used to target the Epiphany cores. The Odroid-XU4 SBC includes a Mali-T628 GPU coprocessor which is also OpenCL compatible. Both are available for under $100.

Programming audio processing routines as OpenCL kernals might be considerably more complex than in C++, but OpenCL offers vendor-independent access to GPGPU computing and has the added benefit that it can also be used on a CPU without GPU acceleration [5].

Investigating these, and other OpenCL enabled SBCs might be an interesting followup project.

2.5 Virtual Analog Synthesis

Virtual analog synthesis is the term used to describe the emulation of analog synthsizers of the 60s and 70s digitally in real time. The complexity and goals of an emulation can vary. Some emulations go so far as to simulate the actual electronic components of vintage synthesis circuits, others just model ths signal flow loosly.

Regardless of the type of emulation, and model of analog systhesis has two primary concerns. Latency and aliasing. The problem of latency has already been described above. Any processing will introduce a delay in the signal, the complexity of the processing can increase the delay, or use more CPU cycles. Aliasing is audible distortion introduced by signals that have a higher frequency content than the sampling rate of the system allows for [8].

Analog synthsizers usually employed what is refered to as subtractive synthsis. One or more sound generators or oscillators would create signals with particular harmonic qualities. These signals would then go through filters that would "subtract" frequencies from the signal. The oscillators and filters can be modulated as well as the amplitude of the filtered signal. Figure 2.1 is a simple block diagram of a typical subtractive synth voice.

The voltage controlled oscillators generate simple waveforms at the pitch coresponding to the note played on the keyboard. The user can typically choose between some combination of sawtooth, squareware, or triangle waveforms. The frequencies or timbre of the waveforms can then be modulated by the following filter and amplitude blocks.

One's first impression might be that modeling the oscillator would be simple. A digitally generated squarewave or sawtooth waveform should be trivial to implement. The 5kHz sawtooch waveform for example, would have a period of 8.82 samples when generated in a 44.1kHz audio environment. So the waveform would increase linearly from -1.0 to 1.0 over a length of 8.82 samples, then jump back to -1.0 and cycle through

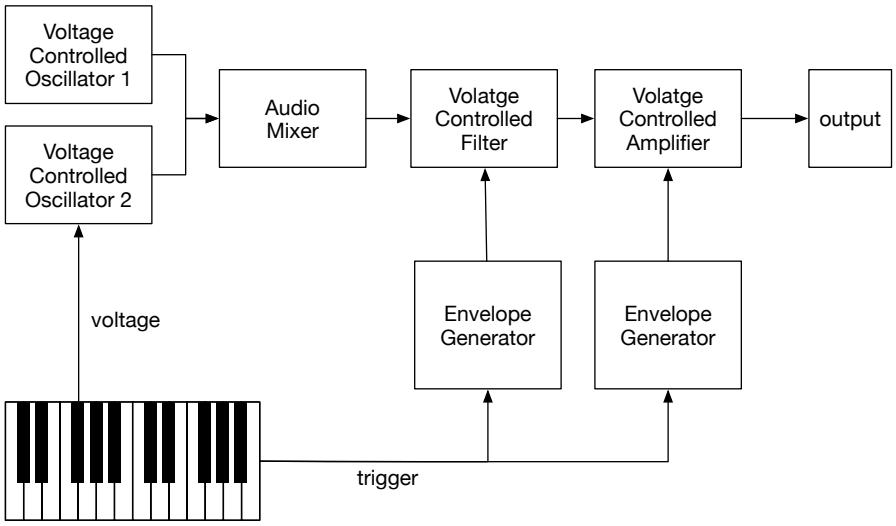

Figure 2.1: Block Diagram of a Subtractive Synthesis Voice

again. What does 0.82 sample mean in a discreet digital system? Figure 2.2 illustrates the problem with a trivial implementation. The left column shows a portion of an idealized 5kHz sawtooth waveform and the coresponding frequency content. Above the 5kHz fundamental frequency are harmonics that will be audible well beyond 100kHz. The right column shows the same portion of a 5khz sawtooth waveform in a 44.1kHz environment. The waveform itself is distorted and the higher harmonics are visible reflected back from the 22.05kHz nyquist limit.

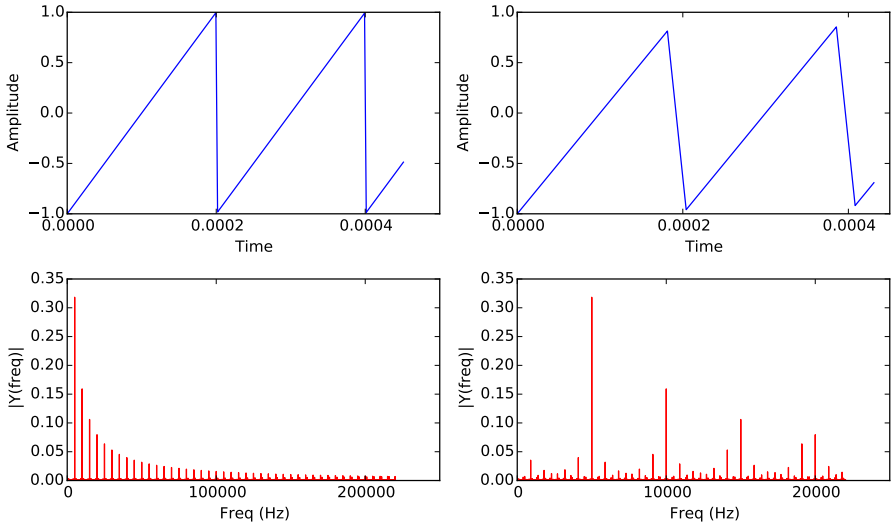

Figure 2.2: Ideal and Aliased 5kHz Sawtooth waveform

Chapter 3

Anforderungsanalyse

3.1 General Application Requirements

The Application has two components, the audio plugin hosted on the main CPU machine, and the processing node which runs on a networked SoC device. The audio plugin forwards midi control and audio data to the processing nodes. The nodes stream the processed audio data back to the audio plugin, which in turn streams it back to the host audio application. The total round-trip time, including processing, should not exceed 10ms. This is the maxium allowed latency for live sound applications. [7]

The applications must be self contained and work without the user needing to install any system libraries, frameworks or servers.[1]

3.1.1 Audio Plugin

The audio plugin has the following requirements:

- runnable as a realtime VST audio plugin in a standard audio application
- locate and connect with one or processing nodes on the network
- forward midi and audio data from the host audio application to the networked nodes
- receives audio data from the networked nodes and streams this back to the host application

3.1.2 Processing Node

The processing node has the following requirements:

- broadcasts its availability and location on the network

[1] The only exception might be ZeroConf/Bonjour on Linux or Windows. See Appendix

- accepts session initiated by the audio plugin
- accepts controll data from the audio plugin
- processes incoming audio data and midi data from the audio plugin
- streams audio and midi data back to the audio plugin or to the next node in the processing chain

3.1.3 Software Requirements

In realtime audio applications timing is critical. This may sound obvious, but to a programmer it means giving up many of the comforts of modern programming made available working with high level interpreted languages such as java or python. Most audio application interfaces and SDKs such as the VST SDK require a knowledge of C and C++.

Professional audio applications generally run on Mac OSX or Windows Operating Systems, therefore the audio plugin must be compileable on these systems. The processing node will be run on SoC devices which typically run with a Linux based OS. Yet both applications should share much of thier codebase since they must do similar things and be compatible.

3.1.4 Evaluated C++ Frameworks

There are many C++ libraries and frameworks that..

Software Framework Criteria:

- Crossplatform for OSX, Linux, and Windows
- Offers high level constructs like smart pointers
- Support for crossplatform audio integration
- Should be well documented and have an active community
- Support for crossplatform network streaming

Several C++ Frameworks were evaluated for this project. The criteria used to evaluate the frameworks are the ease of integration, size of community and degree of acceptence. Does the framework cover the software requirements defined above?

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

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 networking utilities, not appyable to this project though
[3]enabled using one of the additional iplug libraries
[4]the ofxNetwork addon allow simple management of TCP or UDP sockets

Chapter 4

Implementation

4.1 Architektur

4.2 Developement Environment

4.3 Networking Components

These are general C++ components that are used in both the audio plugin and the processing nodes. They encapsulate the network communicatation handling.

4.3.1 Socket Monitor

The socket monitor class uses a posix select() function to montitor the status of a collection of socket file descriptors. When a file descriptor becomes ready to read from or write to the socket monitor can notify a registered delegate class via a specified callback.

- runs as a thread, blocks on select() until one of the managed filedescriptors becomes ready to read then notifies corresponding listener
- define abstract listener class: FileDescriptorListener
- select can be configured with a timeout, unblocking the call, inorder to check status of application (everything ok? should i shut down? should i update? etc..) but it is more effiecient to use a control socket that can be trigged by the app to unblock the select call when needed.
-

4.3.2 ZeroConf Manager

The ZeroConf Manager class encapsulates calls to the system's bonjour/zeroconf deamon.

4.3.3 AudioStream Manager

This class is responsible for streaming audio data to or from a specified udp port. It implements

Chapter 5

Conclusion

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 eect 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.

[9] Wikipedia. Virtual studio technology — Wikipedia, the free encyclopedia, 2015. [Online; accessed 05-August-2015].

Chapter 6

Appendix A

6.1 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

6.2 Bonjour

Instructions for installing bonjour:
If bonjour / zeroconfig is not installed on the bananapi 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
./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
Requirements: sudo apt-get install intltool sudo apt-get install libperl-dev sudo apt-get install libgtk2.0-dev sudo apt-get install libgtk3.0-dev sudo apt-get install libgdbm-dev sudo apt-get install libdaemon-dev