Università Politecnica delle Marche
Scuola di Dottorato di Ricerca in Scienze dell'Ingegneria
Curriculum Ingegneria Biomedica, Elettronica e delle Telecomunicazioni

Digital Signal Processing and Embedded Systems for Wireless Networked Music Performance

Ph.D. Dissertation of:
Leonardo Gabrielli

Advisor:
Prof. Stefano Squartini

Curriculum Supervisor:
Prof. Franco Chiaraluce

XIII ciclo n.s

Università Politecnica delle Marche
Scuola di Dottorato di Ricerca in Scienze dell’Ingegneria
Curriculum Ingegneria Biomedica, Elettronica e delle Telecomunicazioni

Digital Signal Processing and Embedded Systems for Wireless Networked Music Performance

Ph.D. Dissertation of:
Leonardo Gabrielli

Advisor:
Prof. Stefano Squartini

Curriculum Supervisor:
Prof. Franco Chiaraluce

XIII ciclo n.s

Università Politecnica delle Marche
Scuola di Dottorato di Ricerca in Scienze dell'Ingegneria
Facoltà di Ingegneria
Via Brecce Bianche – 60131 Ancona (AN), Italy

dedicated to Daniela, Sandro and Daniele,
my family, first and greatest blessing

Acknowledgments

I need to thank Stefano first, whom has not only been my PhD advisor, but also a friend and a guide in many regards. He gave me freedom as much as he could to scamper through the wilderness of human knowledge. My warmest thoughts go to Emanuele, Andrea, Gilberto and all the furry rabbits™. Thanks to all the department guys with whom I shared a piece of life, some techie adventures and very good moments including Susanna, Adelmo, the warm TLC+openspace guys, Emap and the A3Lab guys (Mr.Spring, we miss you). How not to mention the banana republic guys Giacomo, Bisi and Giobbi for their outstanding ability in merging political commitment, signal processing and binge? Terrific (in the UK or US sense?) technical achievements are foreseen in tandem with the two Marco (Severini and Fagiani). A part of this wonderful journey has been conducted together with Mr. Paolo Bragaglia, the visionary mind behind Waterfront. Such performance would have never been possible without Acusmatiq festival The supporting role of Prof. Francesco Piazza and Prof. Sauro Longhi in making our university a partner in this venture has been extremely valuable. Acusmatiq and Waterfront would have never been possible without the commitment and competence of its artistic director, ARCI Ancona, the performers Giampaolo Antongirolami, Enrico Francioni and Laura Muncaciu, Alessandro Petrolati, the support guys Diego Droghini, Giulio della Porta and Lorenzo Baronciani, and many more.

In life I wander through the varied terrains of music, its genres, its history and its emotions. My last discovery was Jazz roots and its developments into afro-american swing dances as a source of energy, introspection and rejoice. In spite of the discrimination they were subject to, the afro-american communities in XX century gave us much... and still do nowadays: we owe you a lot Frankie, for what you did, and that special *way how you do it.* And thank you Norma, I wish we'll have another breakfast together.

Probably, all this music passion of mine has started as a child thanks to a couple of very special long playing vinyls, some messy digital piano prototypes, and my dad. But even earlier, aged 5 I asked my mum for a guitar and debuted as a political activist singer with songs about child labour in coal mines in the XIX century. A bit too far in time? Truth is, I'm always late. Gotta hand the manuscript to the secretary!

Ancona, Gennaio 2014

Leonardo Gabrielli

Sommario

Le nuove piattaforme per il mobile computing, introdotte alcuni anni fa hanno avuto un impatto notevole su tecnologia e società. Due aspetti tecnici che hanno alimentato lo sviluppo del mobile computing e a loro volta sono stati alimentati da esso, sono stati la maturazione della comunità del software open-source e la disponibilità per molto produttori di silicio della proprietà intellettuale di un'architettura di calcolo molto efficiente in termini energetici. Come conseguenza dell'ampia diffusione del mobile computing, le comunità accademiche della computer music e del music computing, hanno messo a frutto le conoscenze sviluppate nell'ambito del laptop computing per portare alla fioritura di numerosi progetti di physical computing.

In questa tesi di dottorato tutti questi aspetti vengono sviluppati e studiati in maniera organica, con un interesse specifico per l'elaborazione del segnale (DSP) nello strumento musicale digitale e la performance wireless su rete. L'idea di una piattaforma di physical computing in grado di supportare il ruolo di strumento musicale e connettersi ad altri strumenti viene elaborata. Processori di segnale e algoritmi per l'industria musicale vengono analizzati da una prospettiva tecnica e storica per definire delle specifiche tecniche. Sviluppi innovativi nel DSP per la sintesi, il trattamento, il campionamento e la trasmissione del segnale sonoro sono dettagliati Lo stato dell'arte per le performance musicali in rete (NMP) viene riportato insieme alle sfide tecniche che questo impone, in modo da definire le necessità da soddisfare per una piattaforma embedded per la NMP wireless. Da questo approccio multidisciplinare scaturisce il progetto WeMUST (Wireless Music Studio), che investiga su alcune delle problematiche attualmente riscontrate ed evidenziate in precedenza e verifica la fattibilità di realizzare un sistema di NMP wireless basato su soluzioni hardware e software di largo uso.

Giunto lo sviluppo ad uno stadio soddisfacente, una prima versione del software WeMUST è stata rilasciata ed un caso d'uso è stato mostrato in pubblico nel luglio 2014 con la performance *Waterfront*, con musicisti su barche sulla costa di Ancona. Questa occasione di valutazione del lavoro è servita anche come riferimento per future fasi di sviluppo.

Abstract

Since their inception, mobile computing platforms had a tremendous impact on technology and society. Two key technical aspects that fostered mobile computing and were further fed by its advancements are a mature open-source software community and a power-efficient processor architecture intellectual property (IP) accessible to many silicon manufacturers. In the computer music and music computing academic community, developments of mobile computing platforms, allowed for a blossoming of physical computing works stemming from previous developments in laptop computing.

In this thesis all these key aspects are intertwined and studied with specific interest in digital musical instrument signal processing and wireless network performance. A physical computing platform that satisfies both requirements and enacts both the role of an instrument and a link to other instruments is envisioned. Following the unifying trend in computing to fit all functionalities in a single device, a prototyping embedded platform has been chosen for the twofold aim. Digital signal processors and Digital Signal Processing (DSP) algorithms for the musical industry are therefore studied from a technical and a historical perspective, to look for the embedded platform technical requirements. Novel achievements in DSP for sound synthesis and processing, sampling and transmission are addressed. The needs and state of the art in Networked Music Performance (NMP) are summarized to gather insight on useful features for a wireless NMP system based on the embedded platform. This all-round design approach resulted in the conception of WeMUST (Wireless Music Studio) a project addressing several of the issues defined in the study phase and verifying the feasibility of a wireless NMP system based on commercial hardware and software solutions.

Once the development of WeMUST proved satisfying to some extent, a first release of WeMUST and its tools has been publicly rolled-out and an application case has been demonstrated in July 2014 with *Waterfront*, a novel wireless performance between boats off the coast of Ancona, Italy. This occasion of evaluation of the work served also as reference for future development.

All registered trademarks are property of their respective owner.

Contents

List of Figures

List of Tables

Abbreviations

A/D	**A**nalog to **D**igital conversion (also, ADC)
AES	**A**udio **E**ngineering **S**ociety
ALSA	**A**dvanced **L**inux **S**ound **A**rchitecture
API	**A**pplication **P**rogramming **I**nterface
DAW	**D**igital **A**udio **W**orkstation
DCF	**D**istributed **C**oordination **F**unction
DFA	**D**elay **F**eedback **A**pproach
DFT	**D**iscrete **F**ourier **T**ransform
DLL	**D**elay **L**ocked **L**oop
DNS	**D**omain **N**ame **S**ystem
DSP	**D**igital **S**ignal **P**rocessor,
	or **D**igital **S**ignal **P**rocessing depending on the context
D/A	**D**igital to **A**nalog conversion (also, DAC)
DSL	**D**igital **S**ubscriber **L**ine
DWT	**D**iscrete **W**avelet **T**ransform
DWPT	**D**iscrete **W**avelet **P**acket **T**ransform
EEF	**E**aster **E**gg **F**ollows
WRH	**W**hite **R**abbit **H**ole
FFT	**F**ast **F**ourier **T**ransform
FTA	**F**ake **T**ime **A**pproach
GP	**G**eneral **P**urpose
GPS	**G**lobal **P**ositioning **S**ystem
GUI	**G**raphic **U**ser **I**nterface
IC	**I**ntegrated **C**ircuit

IP	**I**nternet **P**rotocol
	or **I**ntellectual **P**roperty depending on the context
ISM	**I**ndustrial **S**cientific **M**edical
JACK	**J**ack **C**onnection **K**it
LAA	**L**atency-**A**ccepting **A**pproach
LAN	**L**ocal **A**rea **N**etwork
LBA	**L**aid **B**ack **A**pproach
LTE	**L**ong **T**erm **E**volution
LUT	**L**ook-**U**p **T**able
MAC	**M**ultiply and **AC**cumulate
MA	**M**usical **A**pplication
MIDI	**M**usical **I**nstruments **D**igital **I**nterface
MIMO	**M**ultiple **I**nput **M**ultiple **O**utput
MPC	**M**ixing **P**ersonal **C**omputer
MSA	**M**aster-**S**lave **A**pproach
NFC	**N**ear **F**ield **C**ommunication
OS	**O**perating **S**ystem
OSC	**O**pen **S**ound **C**ontrol
OSS	**O**pen **S**ound **S**ystem
PCF	**P**oint **C**oordination **F**unction
PDF	**P**robability **D**ensity **F**unction
PDV	**P**acket **D**elay **V**ariation
PLL	**P**hase **L**ocked **L**oop
PLR	**P**eriod **L**oss **R**ate
RIA	**R**ealistic **I**nteraction **A**pproach
RISC	**R**educed **I**nstructions **S**et **C**omputing
RTT	**R**ound-**T**rip **T**ime
SABy	**S**imple **A**utonomous **B**uddying
SFG	**S**ignal **F**low **G**raph
SoC	**S**ystem **o**n a **C**hip
SME	**S**mall to **M**edium **E**nterprise

SNR	**S**ignal to **N**oise **R**atio
SP	**S**ignal **P**rocessing
STFT	**S**hort **T**ime **F**ourier **T**ransform
STP	**S**tandard **T**emperature and **P**ressure
TCP	**T**ransmission **C**ontrol **P**rotocol
UDP	**U**ser **D**atagram **P**rotocol
UPnP	**U**niversal **P**lug and **P**lay
VCO	**V**oltage **C**ontrolled **O**scillator
WAN	**W**ide **A**rea **N**etwork
WLAN	**W**ireles **L**ocal **A**rea **N**etwork
WPEC	**W**avelet **P**acket **E**nergy **C**oefficients

Chapter 1

Introduction

An argument could be made, that - although of totally different nature - music and technology are intertwined aspects of the same human ambition. The first technical achievements in human history paved the way for a wealth of new tools to help mankind with primary objectives, but they also offered new ways to express himself in artistic invention, and of greatest interest here, create sound and practice music. This happened well before modern Science was born. Still, while in the last centuries physics managed to understand instrumental acoustics, mathematics introduced a wealth of concept useful for signal processing and communication theory helped spread music on the whole planet surface, a relevant part of contemporary music dialectic, debate and practice is inspired by or takes place with technologists and craftsmen, rather than scientists. Scientific breakthrough in the recent progress of musical ethics and aesthetics has always been a guide on par with many other intellectual currents and findings, but in essence, it could be that in the development of this Art, playing with technical inventions has a greater impact than recurring to rigorous thinking. It may not be a coincidence that the verb used for a musical instrument is *to play*. Indeed, curiosity and readiness made my PhD studies cover several topics that needed to be addressed following debate with expert in fields related to music technology, and, yet, they come out to be sufficiently organic to be weaved in this essay following a thread: the *embodiment* and *un-embodiment* of the musical act physicality, that is, the development of a *physical computing* platform for sound synthesis and processing and the virtualization of the musical action through *wireless network performance.*

The thesis sweeps through different fields of current technological development that have been studied and improved to fill in some of the gaps in current music technology achievements. All these have been gathered together into a single project that addresses several issues considered worth exploring in current (experimental or traditional) music performance practice. The reader of the thesis is therefore requested to forgive on the indulging on matters of the human knowledge displaced too far away from each other.

1.1 Reading this thesis

The main focus of the thesis is on musical performance employing wireless networking techniques. This field of research appears nowadays as a moorland, with very few contributions sparse in the last decade. The idea itself may not be novel, but surely employing the wireless medium for networked music performance (**NMP**) sounds difficult and unreliable. On the contrary, it poses a number of challenges worth addressing, especially in these days of steady wireless technology innovation. Once the signal wire is cut, the last wire to cut is the energy supply: current computing architectures and energy storage technologies allow for sufficient operation on *mobile* conditions, thus, suggesting the viability of the wireless approach to NMP. The same stands for weights and portability. These cues, were the starting point for WeMUST, the Wireless Music Studio project, which is covered in this essay and is based on existing communication standards, most notably, the IEEE 802.11 family (alsa known by its commercial name *WiFi*®). However, to build up a prototyping platform and address all the issues reported above and a few more, studies have been conducted involving many subjects that were worth investigating first, such as Digital Signal Processing (**DSP**) for Musical Applications (MA). Many academic contributions have been proposed in this process. Finally, the outcome of the WeMUST project is a tangible object (as required by thye physical computing approach) for performance, thus, part of the studies also cover embedded computing for music and digital music instrument making in general.

A summary of the chapters is given:

- Chapter 2 provides an introduction to musical signal processing, the development of digital musical instruments and digital signal processors, outlining some concepts in digital musical instrument design. It also introduces general purpose computing platforms and gives motivation for their use in digital musical instruments design;

- Chapter 3 gives a brief historical overview on networked music performance (NMP) and then introduces challenges, technical issues and approaches;

- Chapter 4 is the focal point of the thesis, introducing the WeMUST (Wireless MUsic STudio) project, its outcomes and the solutions found to some of the technical issues reported in the previous chapters;

- Chapter 5 reports on several advancements in music signal processing achieved by the author that directly address computational cost issues and scalability to embedded platforms;

- Chapter 6 concludes the thesis with a discussion on future roads for physical computing in music, including foreseeable trends.

Chapter 2

Embedded Computing for Musical Applications

Nowadays a number of musical applications (if not all) rely on Digital Signal Processing. Embedded systems perform DSP by means of specialized ICs (Integrated Circuits), programmable processors such as Digital Signal Processors[1] or more general purpose programmable processors. For this reason, this chapter includes a brief timeline of digital musical instruments, the DSP circuits and computing architectures. Moreover an overview about signal processing for musical applications is given. One basic building block is pointed out and implemented for a benchmark on a few reference platform for general purpose computing. More specific signal processing algorithms targeting embedded platforms are reported in Chapter 5.

2.1 Digital Musical Instruments: a Brief Timeline

Digital musical instruments evolved and derived their features from a few archetypal architectures, most notably the digital computer and the analog synthesizer, with a few exception in the more experimental areas of music production. Those two are of opposite nature: the analog synthesizer in many incarnations resembles the general model of many acoustic musical instruments

[1]along the thesis the acronym is used for both Digital Signal Processor and Digital Signal Processing when not ambiguous, otherwise they are reported in extenso

including the human voice, i.e. the source-filter model; the digital computer, on the other side is inherently tied to a higher-level understanding of music generation, related to the freedom available with programming, which allows sound organization and ontology creation *a priori* from timbre crafting. This dichotomy between the physically-inspired nature of the source-filter model and the conceptual nature (Platonic in some sense) of computer music is somewhat bridged by the opportunities introduced in the last years by overcoming the distinction between composing and executing, DJing and listening and some technical novelties such as touch interfaces, fast (software, hardware and mechanical) prototyping, digital manufacturing technologies, displays, wireless communication and miniaturization. Novel instruments and interfaces do not take inspiration from early electronic instruments (e.g. the keyboard synthesizer, or the DJ turntable) and represent a synergistic synthesis of existing functionalities (see e.g. the Kaoss instruments, half samplers, half effect processors). This variety brings to a large semantic discordance in "organizing sound" and its production. *Memories*, *layers*, *patches*, *timbres*, *scores*, *orchestras*, *presets*, *performance*, *sample*, these are a few of the terms used in modern computer music and digital instruments to classify sound, its control and storage - which is a crucial aspect and innovation of the digital instruments era.

Digital musical instruments history dates back to the inception of computer music, after WWII, when a few notable composers, researchers and engineers started employing mainframe computers as a resource to explore new fields in contemporary music. Apart from the first computer-synthesized music experiments in 1951 (Trevor Pearcey's and Maston Beard's CSIRAC[2] computer, and Cristopher Strachey program on the Ferranti Mark 1[3]) that were just monophonic execution of popular tunes, later experiments led to extremely innovative forms of composition through computer programming. Without doubt, the most influential scientist from this perspective was Max V. Mathews, who in 1957 realized *MUSIC*, a programming language for music composition and generation [3, 4]. From that moment on, computer music grew popular in the

[2]http://www.csse.unimelb.edu.au/dept/about/csirac/music/introduction.html
[3]http://news.bbc.co.uk/2/hi/technology/7458479.stm

academia and music programming languages continued to be refined and created. During these years, digital technology was still in its early stages, music was generated off-line and equipment was too large and heavy to incarnate anything close to a real musical instrument. While academic research progressed on DSP algorithms and architectures, the first electronic musical instruments and effects to become popular during the 1960s and 1970s were all-analog. The invention of the transistor made the integration of oscillators, filters, modulators and amplifiers feasible in a rather small space, thus analog voltage-controlled synthesizers can be numbered among the first electronic instruments to be portable and commercially successful. An early digital sampler was devised by Bruce Haack, an inventive electronic musician who in 1967 showed his "Musical Computer" on the American *Mister Rogers' Neighborhood* TV show. In 1969, Zinovieff, Grogono and Cockerell developed the EMS Musys system, based on two 12-bit microcomputers, DEC PDP-8. Their London studio was the first digital studio, controlled by the large machine.

In the 1970s, synthesizers could employ digital CPUs only to introduce so-called *memories* or *patches* and connectivity to external units. It is the case of Sequential Circuits Prophet 5, with a Zilog Z80 MCU controlling a full analog signal path and 5-voices polyphony. Audio processing was not yet feasible. For the sound synthesis, the instrument hosted DIP-package integrated oscillators and filters circuits specifically made for the music industry, such as Solid State Micro SSM2030 VCO (Voltage-Controlled Oscillator) or, in a later hardware revision Curtis Electromusic Specialties CEM3340. In these years of mature analog products and flourishing digital products the two aforementioned companies CES and SSM were supplying components specifically intended for the musical instruments market (together with electronic games or other applications). All the famous electronic instrument producers from Japan and the US, as well as the smaller Italian ones, such as Crumar, Siel, Solton and the likes employed these components in their instruments. This was possible due to a favorable market[4] and the lack - yet - of general purpose signal processors.

[4]Several successful products sold in large quantities (tens of thousands in some cases, or even more for a handful of Japanese products, which is a lot for the music market).

At the end of the 1970s the miniaturization and progress of the digital technology made new instruments feasible. It is the case, e.g. of the Fairlight CMI, the first polyphonic digital sampling system, based on a dual Motorola 6800 processor. Its street price was as high as £18,000. While digital samplers need some signal processing (envelope generation, filtering, etc) in their basic form they do not have a high computational cost, thus, for instance, one of the early digital samplers, and of exceptional quality, the Synclavier I (1977), employed a simple 16-bit ABLE processor. This was based on a *transport-triggered architecture*, meaning that it only supports one instruction: MOVE, to move 16-bit data between functional units. The processor architecture, although ingenious and scientifically challenging, never yielded large success in the silicon industry. Bleeding-edge music composers and pioneers started employing digital microcomputers in the 1970s (see also Section 3.1.1). One very popular microcomputer was the Cosmac Elf (1976) built around a RCA CDP1802 processor [5]. The Elf input was represented by an array of toggle switches for bit by bit programming and its hexadecimal output was shown in a two digit 7-segment display. One notable feature of this IC was its Q output, an output pin that was set by a dedicated processor instruction, making it easy to employ it for communication or even tone generation, by toggling it at a specified frequency. The first Philadelphia Computer Music Festival have seen a large use of Elf microcomputers, in 1979. One last notable feature of this processor is the ability to step down the operating frequency from its maximum down to zero. Stopping operation of the CPU in this way does not reset operation and it can be regarded as an early form of power saving.

A great breakthrough in digital sound synthesis was the use of FM synthesis, patented by electronic composer and professor John Chowning, developed around years 1967-1968, filed as a patent in 1974 [6] and later sold to Yamaha, which in 1983 released the famous DX7 on a slight variation of the method. This employed a 63X03, a variant of the Motorola 6800, and an additional MCU, 6805 to scan the keyboard and panel. The sound synthesis was performed with the use of a proprietary chip, YM21280 OPS as the FM operator, running at 49096 Hz. The output of this chip was a 14-bit frequency value, with 4 bits indicating the octave and the remaining 10 a linearly spaced value in the octave. The chip contained several LUTs (Look Up Tables), reversed engineered in recent times through direct inspection of the silicon die.

The DX7 was one of the first keyboard synthesizers to include MIDI[5] (Musical Instrument Digital Interface), a universal digital communication protocol ratified in 1983 by a panel of music industry representatives. An early protocol implementation was devised and shown in 1981 by Sequential Circuits' Dave Smith at an AES show and was later modified together and adopted by Oberheim and Japanese companies Roland, Yamaha and Korg. The 1983 winter NAMM show in the US saw the connection of an American Prophet 600 synthesizer and a Japanese Roland JP-6. The seamless connection of instruments coming from such distant places on Earth was a significant event from a symbolic perspective and a giant leap for music: not only new opportunities were unveiled for musicians, but barriers between different manufacturers' equipment were finally broken.

The MIDI specifications were published in August 1983 and since then underwent only minor additions and modifications. Currently MIDI is implemented in almost any hardware and software instrument, and is often supplied through USB connection, retaining the upper layers of the protocol.

In the 1980s digital chips for sound synthesis started spreading also on computers, for gaming but also for sequencing: early personal computers had sound cards with FM synthesis chips and software sequencers. The famous Cubase DAW software, e.g, was released in 1989 for Atari and was MIDI capable.

Digital sampling was exploited as well, at first in the primitive form of wavetable synthesis (by LUTs containing a waveform cycle or less) and later in form of more complete sampling synthesis, employing larger chunks of audio data.

The development of DSP techniques was at the base of two fundamental synthesis methods: DWG (discussed in detail in Section 5.3) first formulated in 1983 [7, 8], one of the most common physics-based sound synthesis techniques and digital subtractive synthesis by alias-free oscillators [9, 10], one of

[5]http://www.midi.org/

the first Virtual Analog outcomes, pioneered by a few academic articles and showcased at the NAMM show in 1994 by Swedish company Clavia. Those new synthesis methods fostered new areas of research in DSP and required specific architecture for number-crunching: by 1990s the digital signal processors were capable of handling the large number of operations required by these MAs, and improved in the years.

With programmable DSPs and MCUs, know how can be saved for future projects, and code reuse allows extra flexibility. In the digital era open-source initiatives fostered new software and hardware design paths and accelerated productivity. In years 2000s, the GNU/Linux operating system, originated by Linus Torvalds and the GNU movement efforts, started being available for an increasing number of embedded devices. Featuring low resources requirements but providing higher abstraction from the hardware compared to many embedded OSs, it was chosen as the base for control and management of several heterogeneous systems comprised of MCUs and DSPs. One notable family of products is the Italian digital organs Viscount UNICO, presented in 2004 and featuring a Linux-based ARM CPU and parallel DSPs for DWG physical modelling of organ pipes. In 2005, KORG presented Oasys, based on an Intel Pentium 4[6] and a *"customized Linux"*, followed by the KORG Kronos in 2011 with an Intel Atom D5xx[7] and a *"custom OS over Linux"*. In the same year Viscount introduced the Physis piano, with Linux running on an ARM and managing 6 parallel DSPs.

In the first 15 years of the century most musical machines, from effects processors and keyboards, to karaoke and DJ equipment started taking advantage of modern DSP algorithms and thus require capable processors. A few highlights on DSP development are given below.

[6]running at 2.8 GHz, and including SSE2 instruction set, this processor was introduced in 2002, and had a TDP of 68W.

[7]1.6-1.8 GHz, 13 W TDP, implements also SSE3.

2.2 Digital Signal Processors in Musical Instruments

As reported above, the first digital components used in musical instruments were mainly microcontroller units, and custom VLSI digital chips (as in the DX7). Some chips might be used to provide signal manipulation functions, e.g. bit-slice, multiplication and accumulation. The need for many different ICs, however, made integration hard for digital signal processing applications. However, the demanding requests of other applications, such as telecommunication systems, radar and military, automotive and so on, eventually pushed the silicon industry to come up with commercially viable VLSI solutions with steadily increasing performances. One shared definition of a DSP is a programmable non application-specific processor with concurrent data and instruction access (e.g. implementing a Harvard or modified Harvard architecture, as opposed to a Von Neumann computer architecture). In this regard the first ICs to be independent of a microcontroller or an external ROM and featuring a hardware multiplier (fundamental to most signal processing task), a control memory and a data memory were the NEC µPD7720 and the AT&T DSP1 both presented first at the International Solid-State Circuits Conference in 1980. Later in 1983 Texas Instruments presented the TMS32010, based on a Harvard architecture and capable of load-and-accumulate and multiply-and-accumulate instructions.

The subsequent generations of DSPs (e.g. Motorola 56000, also addressing 24-bit audio applications) added two simultaneous operands, increased bit-depth and improved performances. At these times DSPs were programmed in their assembly language and worked on a sample-by-sample basis for architectural reasons. Their ALUs were fixed-point and a clock cycle could be about $0.1\mu s - 0.2\mu s$.

In the 1990s DSPs started implementing more complex signal processing instructions in hardware, such as the Fourier Transform or matrix and vector operations. Homogeneous and heterogeneous parallelism was also implemented by means of parallel cores on a same die or by supporting the DSP core with a MCU core, such as Texas Instruments starting coupling TMS320C5x with

ARM7 units. Power efficiency features, VLIW and SIMD instructions set were added along the 1990s. From a developer’s perspective in the 1990s development tools were implemented to accelerate the time to market. In 1990 TI was providing a C code compiler, a debugger and starter kits with A/D and D/A on board. In the next year the first DSP Educators Conference took place.

During these years, the first virtual analog synthesizers were presented, the Nord Lead featuring a Motorola 56002 DSP (80MHz, 40MIPS, 24bit fixed point), and some years later the Access Virus.

In 2005 British semiconductor IPs designer ARM Holdings, managed to include DSP capabilities to its ARM architecture family, by including NEON 128-bit fixed/floating-point SIMD instruction set on its ARM Cortex-A8 processors (architecture ARMv7-A), in the fashion of MMX instructions on Intel x86 processors (introduced in 1997). Of course, introducing new instructions, register and data paths comes at high cost in terms of gates. On a Cortex-A9, the RISC core requires 600,000 gates, while the NEON part alone requires 500,000. From an energy standpoint, with CMOS technology, more commuting transistors increase the required power. However, employing specialized instructions achieves a higher efficiency by decreasing the execution time compared to employing RISC instructions.

Although the ARMv7 architecture and related are not meant for DSP-only applications, they are a good candidate for MAs, thanks to the availability of a floating-point unit and SIMD instructions, a large peripheral set for I/O and the low cost. This will be discussed in Chapter 4 and Section 6.2.1. In any case at the moment some demanding synthesis techniques, such as modal synthesis or FDTD, require dedicated processors. The piano described in Section 5.3.3 employs modal synthesis and needs in the worst case 6 parallel TI C674x DSPs for piano synthesis, while experiments reported by Bilbao et al. [11] make use of a set of parallel GPUs for Finite Differences modeling.

Years 2000s have seen competition increasing with other companies such as CEVA developing IP for DSP cores, Analog Devices producing the SHARC series and other manufacturers entering the market such as Marvell, Tensilica and more. By the early 2010s, however, many discrete DSP lines are retired or will not undergo development and Texas Instruments seems to be the only manufacturer to feature a consistent roadmap and investments of DSP development, at least in non communication-specific fields. This does not mean that signal processing is not anymore of interest, or that the audio market is neglected, on the contrary, automotive and mobile markets require ever more challenging DSP algorithms. The fact is nowadays discrete DSPs are only 1/10 of the estimated DSP market, with the remaining part embedded into other cores or ICs[8]. In May 2010, iSuppli determined a descending trend for the discrete DSP market, with a pace of -8.2% yearly, while the processors market was increasing by 4% every year[9].

2.3 General Purpose Processors for Musical Applications

By observing that the discrete DSP market is shrinking, while the global processor IC market is growing, it is interesting to evaluate alternative embedded computing architectures for music DSP. Cost-effective and widespread alternatives are:

- GP CPUs, such as x86,
- FPGAs,
- GPUs,
- ARM architectures.

FPGAs are used for signal processing applications were a custom computing architecture is required, but their cost, size and power requirements do not make this solution attractive. GPUs are similar and complimentary to DSPs. While they are able to process vector data in an extremely fast and efficient

[8]https://fwdconcepts.com/dsp-market-bulletin-111212/

[9]An interesting presentation adding to this small review can be found online by Francois Charlot http://www.slideshare.net/fcharlot/digital-signal-processor-evolution-over-the-last-30-years

way, they are meant for graphical processing, thus adapting their architecture to audio-specific mathematical operation can be hard. Their peripherals may not be meant for audio exchange, thus some mechanism must be employed to get audio data in and out by means of external components (or the host MCU if they are integrated in a SoC), for real-time usage. Much research has been conducted on their use in the computational acoustic field by [11, 12] which seems very promising. However, at the moment effective usage of GPUs for musical instrument prototyping seems far from feasible for quick prototype implementation.

x86 processors are already running any kind of signal processing application, they are widespread, compilers are very efficient in optimizing code and there is already a vast number of libraries, tools and code to reuse. One issue with these platform is power. Desktop-class CPUs have TDP (Thermal Design Power) figures of 50-150W which are not affordable in terms of heat dissipation or are prohibitive in battery powered applications. Laptop-class x86 CPUs have lower requirements, even less for the Atom x86 processors that were originally targeted to stay below the 2W point (Silverthorne series, circa year 2008), but are currently rated at 6-20W TDP. On the contrary, mobile ARM platforms rarely surpass the 2W limit. Their performance is comparable to laptop x86 processors, much code exists that can be reused and porting of Linux distributions are widespread.

For reasons of cost, code reusability, availability of open-source kernels and drivers, power efficiency, availability of development kits including the audio chain, ARM-based platforms have been evaluated for prototyping. All the work described in Chapter 4 is conducted on a board named BeagleBoard xM, based on TI DM3730 ARM Cortex-A8 processor.

To assess the convenience of shifting computing on inexpensive ARM platforms, a basic benchmark involving execution of sinusoidal oscillators is reported here. The benchmark has been performed on four processors, a recent x86 Core i5 PC CPU (in short x86-i5), a dated x86 Centrino Duo (x86-Duo), a quite recent ARM Cortex-A7 (ARM-A7) and a dated ARM Cortex-A8 (ARM-A8). The benchmark evaluates the maximum number of sine oscillators that can be computed in a certain time frame, i.e. that keep the RTF (real-time factor) slightly below 1. No attempts have been made so far to benchmark memory transfers, which are equally important, especially with sampling synthesis. The implemented oscillator is based on the Chamberlin topology [13, 14] of the second-order resonant filter, widely used in the music field. The code is plain C, compiled with the GNU compiler with or without optimization (-*O2* and -*O0*, respectively). A C listing including the oscillator function is reported below. The results are reported in Figure 2.1. Some details are given about the processors in Table 2.1.

A few comments can be outlined from this benchmark. It is quite clear that the x86-i5 outperforms the other processors, being one of the more recent in the test, and aimed at high performance, although in a laptop-class TDP. It must be noted, however, that the performance of the Cortex-A7 would theoretically exceed that of the x86-i5 if their clock speed could match. Similarly it reaches the same performance of a 9-years old x86 CPU, but would double it reaching its clock speed. That would, however, increase (nonlinearly) its TDP, admitting that the process and design allows the core to reach such high frequencies. One last thing to note is the relatively low difference between the two x86 CPUs, (a ratio between 1.5 and 1.7). Since the increased performance of the x86-i5 is justified by the increased core frequency, the net performance increase is low. This could be related to an inefficient use of the x86-i5 additional features. One main difference with its sibling is the increased parallelism and larger instruction set. While proper exploitation of parallelism may be a subject for further tests, an improvement in the instruction set (e.g. SSE3 versus SSE4.2) is not easy to exploit or not feasible at all, as the Chamberlin oscillator is based on simple operations. For code loops of this kind, such as filter kernels, architectural features of use are those exploiting tight loops, together with the compiler ability to automatically detect these and efficiently translate it into machine language. The compiler ability to optimize code is clearly seen with the Cortex-A7, which obtains a 4 times speed up when compiled with optimization (probably the compiler optimization exploits the processor 4-threads parallelism capability). The Cortex-A7 code, when compiled with optimization, proves extremely efficient in terms of energy, given its low TDP, in providing laptop-grade performances. The author believes that higher values can be obtained from the processors from careful optimization and parallelization. In that regard, early experiments with OpenMP compiler directives showed the Cortex-A7 performance to increase of at least a factor 2 by exploiting parallel code execution. Further investigation is, however, required for a detailed evaluation.

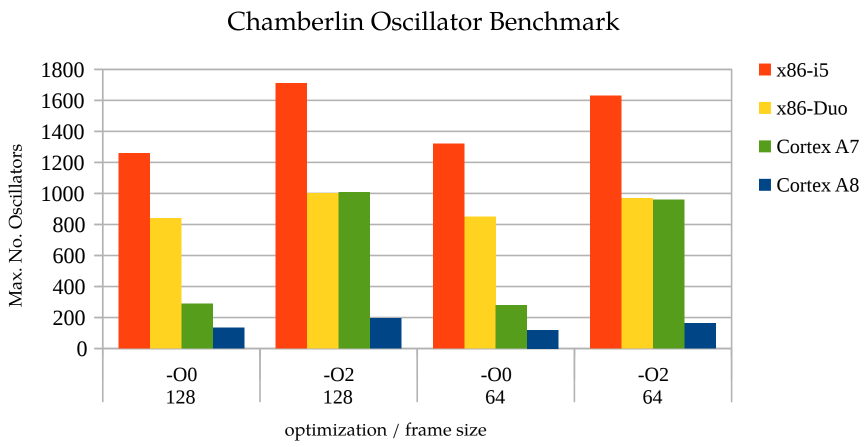


Figure 2.1: Benchmark results from the implementation of Chamberlin oscillators on the four processors described in Table 2.1. Benchmarks are executed with frequency scaling disabled and maximum CPU clock, in repeated batches of tests in order to discard outliers, with highest userspace process priority. All the tests perform processing at 44100 Hz.

	x86-i5	x86-Duo	ARM-A7	ARM-A8
Core	Core i5-3210M 2 cores 2.5 GHz (4 threads)	Core Duo T2500 2 cores 2 GHz (2 threads)	ARM Cortex-A7 (Allwinner A20) 2 cores 1 GHz (4 threads)	ARM Cortex-A8 (TI DM3730) 1 core 1 GHz (1 thread)
L1/L2/L3 Cache	32KB/256KB/3MB	32KB/2MB/N-A	32KB/256KB/-	64KB/256KB/-
Instruction Set	x86 64-bit	x86 32-bit	ARMv7 + Thumb-2 + VFPv4-D32	ARMv7 + Thumb-2 + VFPv3
SIMD	SSE4.2	SSE3	NEON	NEON
Process	22nm	65nm	55nm	45nm
TPD	35W	31W	N/A	1.9W[a]
DMIPS (estimate)	49300	13600	1900,2900	2000 (declared), 1138[b]
MFLOPS (estimate)	36400	10000	N/A	500[c]
Transistor count[d]	63410^6	15110^6	N/A	N/A
Launched	Q2 2012	Q1 2006	Q4 2012	Q4 2010

Table 2.1: Information about the processors used in the benchmark. Please note that information about DMIPS (Dhrystone instructions per second), MFLOPS (Whetstone floating point instructions per second) and transistor count (not to be confused with gate count and highly dependent of the memory) are estimated or taken after comparison from different sources and are only indicated for the sake of a brief comparison. Sources reporting data not compatible or substantially different from the expected numbers have been discarded.

[a] calculated from datasheet worst case values.
[b] http://processors.wiki.ti.com/index.php/Android_Comparative_Benchmarks
[c] http://processors.wiki.ti.com/index.php/Android_Comparative_Benchmarks
[d] Unofficial estimates based on different sources available online

```
void benchmark1 (timespec* tStart, timespec* tEnd)
{
    int ncycles = nOsc;
    size_t len;
    float* op = flovec;

    clock_gettime(CLOCK_REALTIME, tStart);

    while (ncycles--)
    {
        len = frameSize;
        op = flovec;
        while (len--)
        {
            sinZ = sinZ + filtCoeff * cosZ;
            cosZ = cosZ - filtCoeff * sinZ;
            *op++ = offset + amp * sinZ;
        }
    }
    clock_gettime(CLOCK_REALTIME, tEnd);
    return;
}
```

Listing 2.1: Listing of the code implementing the Chamberlin oscillator

Chapter 3

Networked Music Performance State of the Art

One interesting field in music research is that of networked performance. History and challenges about this rather recent research branch are given in this chapter.

3.1 Networked Music Performance: A Brief Timeline

Transmitting music over a distance by technological means was accomplished at the outset of telephone and radio technologies, for technological display. And probably for the sake of annoyed bourgeois mistresses too. One of the first wired music transmission was possible due to Thaddeus Cahill, inventor of the Telharmonium (patent no. 580035, 1896), an "apparatus for generating and distributing music electrically", which since 1906 was employed in performances and could drive 15,000 to 20,000 telephone receivers, according to its main capital investor, Oscar T. Crosby [15]. Its demise was soon to come[1], as radio broadcast was going to spread in the subsequent years. And annoyed mistresses

[1] Cable radio was anyway employed all along the 20th century, with a varying degree of success from country to country and different developments. Its main advantages over radio transmission are a more capillary reach, a higher quality compared to AM transmission and reduced costs compared to digital transmission. Music broadcasting services by means of cable radio are still in use in several countries including Italy.

were not ready for electronic music, yet.

The first radio broadcast dates back to 1910, with an experimental transmission of Mascagni's *Cavalleria Rusticana* and Leoncavallo's *Pagliacci* featuring the Italian tenore Enrico Caruso from the Metropolitan Opera House, in NYC, USA. The amplitude-modulated broadcast, provided by the American inventor Lee De Forest's Radio Telephone Company had some issues, especially with the scarce microphone signal loudness, but the path was set. In Figure 3.1 a New York Times advertisement for the radio (called *wireless*, despite De Forest's preference for the former term) is shown. Although other musical wireless transmission were experimented before[2], the one from 1910 was the first to address a public.

Despite the long history of music broadcasting, music performance over a distance has a much more recent history. Radio modulated waves had an influence over John Cage, who is credited by some authors to be the first composer for a NMP piece [16], with his "Imaginary Landscape no.4 for Twelve Radios" in 1951 (see also [17]). However disputable whether this can be considered an NMP piece, it is surely one of the first human attempt to explore musical interaction at a distance. The same author in 1992-1993, composed *Helikopter-Streichquartett* a string quartet piece to be played on four helicopters. The piece is the third scene of the opera *Mittwoch aus Licht.* The musicians are separated aurally and visually, and are synchronized by a click-track and do not hear each others, while the audience can see and hear all of them through audio and video signals transmitted from each helicopter. Networking was first employed by Max Neuhaus in 1966 in his pieces for public telephone networks. In *Public Supply I*

"*he combined a radio station with the telephone network and created a two-way public aural space twenty miles in diameter encompassing New York City, where any inhabitant could join a live dialogue with sound by making a phone call. Later, in 1977 with Radio Net, he formed a nationwide network with 190 radio stations.*"[3]

[2]most notably, the one from Dr. Nussbaumer at the University of Graz, in 1904, who yodeled an Austrian Folk song from a room to another by means of a receiver and transmitter.
[3]from Max Neuhaus official webpage.

Mazarin Sings by Wireless

"Carmen" Heard a Mile Off

Mme. Mazarin and diagram sketch showing how her songs from "Carmen" were transmitted by wireless.

2-25-1910

Figure 3.1: An advertisement of *wireless music* well before the WeMUST project.

3.1.1 Early Computer Network Experiments

Computer networking experiments in music were introduced by the League of Automatic Music Composers (LAMC), an experimental collective formed by San Francisco Bay area musicians John Bischoff, James Horton, Tim Perkis, and others. The production of the LAMC was improvisational and the group was open. The group recorded music in the years 1978-83. The LAMC experimented with early microcomputers, and also very primitive computer networking. Specifically, they employed multiple MOS KIM-1 microcomputers programmed by themselves in the 6502 CPU machine language, inputting the assembly by means of a numeric keypad. Programs were stored in audio cassettes. Without getting into the aesthetic and compositional aspects of the LAMC experience, it is of interest here to report on the technical details. The microcomputers were interconnected through parallel ports or interrupt signals, directly handled by the microcode written by the composers. No standard networking was employed. The interaction was based on musical representation data, later fed to analog synthesizers or direct D/A for sound synthesis. The flier from an early concert from band members Bischoff, Behrman, Horton and Gold, depicted in Figure 3.2 shows the data path and algorithms employed during the then-upcoming event. This picture is still quite representative of laptop orchestra performances taking place nowadays. Please note that audio signals were not transmitted digitally.

In [18], Bischoff, Gold and Horton, report on another performance taking place at the same venue in July 1978, where three KIM-1 are interconnected in a different fashion and output sound through direct D/A or a 8253 programmable interval timer chip. The interconnection is made through serial or 4-bit parallel ports.

The LAMC proposed the term Network Computer Music for their performances. Such performances provided the basis for the typical laptop orchestra paradigm, with different units exchanging musical or audio data which are subsequently processed by other units for synthesis or manipulation. Notably, the setup would algorithmically generate music and sound in a deterministic but

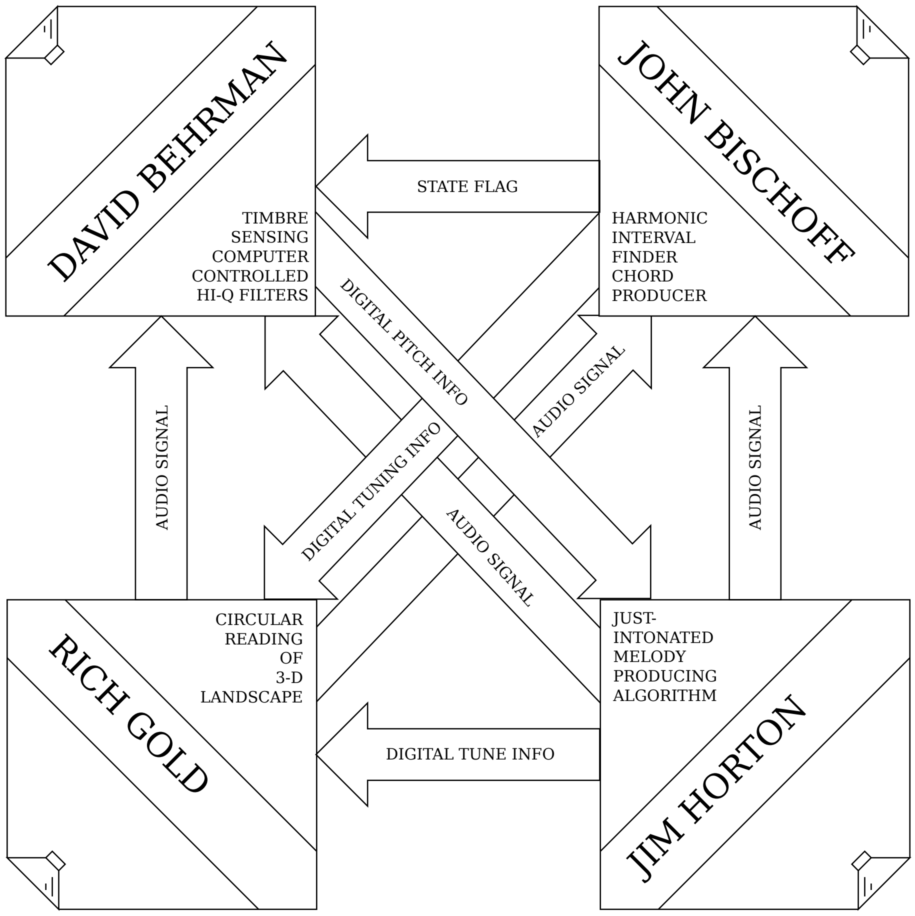

DAVID BEHRMAN
TIMBRE SENSING COMPUTER CONTROLLED HI-Q FILTERS
JOHN BISCHOFF
HARMONIC INTERVAL FINDER CHORD PRODUCER
STATE FLAG
DIGITAL PITCH INFO
AUDIO SIGNAL
DIGITAL TUNING INFO
AUDIO SIGNAL
AUDIO SIGNAL
AUDIO SIGNAL
RICH GOLD
CIRCULAR READING OF 3-D LANDSCAPE
JIM HORTON
JUST-INTONATED MELODY PRODUCING ALGORITHM
DIGITAL TUNE INFO

Figure 3.2: Reproduction of a flier publicizing a concert at the Blind Lemon in Berkeley, USA in Nov. 1978. The flier features a diagram of the data paths between computers and indicates the musical algorithm running at each microcomputer.

unpredictable way, thanks to the feedback nature of the system. The composers themselves claimed to be influenced by some of the intellectual currents of the time, suggesting that complex phenomena could emerge from the interconnection and interaction of simple components. By the way, it is worth citing some of this scientists and writers since their writings were necessary to much computer music theory and practice, still influencing laptop orchestras and NMP composers: Ilya Prigogine (complex system theory and self-organizing systems), Warren S. McCullough (Neural Networks), Gregory Bateson (cultural ecology), John Holland (genetic algorithms). More introspection on the LAMC can be found on a document written by Perkis and Bischoff available on the Internet "The League of Automatic Music Composers 1978–1983" [4]. After the LAMC stopped its activities in 1983, due to Jim Horton's health problems, Perkis and Bischoff followed on the same path and worked on a digital system interconnection interface for musical practice called *The Hub*, which eventually led to the creation of a stable group of performers with the same name. The KIM-based Hub had four UARTS to allow four players to network using 300bps serial connections. This central unit allowed for an easier connection (during the LAMC days the computers were connected by directly wiring and soldering the machines) and standardized format for data exchange. The musicians would, in fact, employ a shared memory (nicknamed "The Blob") where to store data. This was referred to as "blackboard system" and allowed for asynchronous data exchange. The hub would keep information about each player's activity accessible to other players' computers. As reported in [19], after the 1985 Network Muse festival held in San Francisco, featuring Bischoff, Perkis and more artists, *The Hub* collective would develop and add distance to their networked performances. In a series of concerts in 1987 six musicians would play in two different venues in New York City, split into two groups, connected through a telephone line via a modem. The technical effort was considerable, but successful, although one of *The Hub* members, Gresham-Lancaster in [20] comments

"Although the group performed at separate locations a few times, it created

[4]http://www.newworldrecords.org/uploads/fileIXvp3.pdf

its strongest and most interesting work with all the participants in the same room, interacting directly with each other and with the emergent algorithmic behavior of each new piece"

With the advent of MIDI and software sequencers, in 1990, Gresham-Lancaster and Perkis designed a MIDI-based Hub, where each musician's machine was assigned a MIDI port and MIDI messages were employed to exchange data. This way each musician could address privately each other machine, adding flexibility. At the end of the 1990s *the Hub* members explored the use of the Internet as a means to develop their network computer music experiments. Again, the comments by Gresham-Lancaster are not entirely positive,

"In the only test case so far, two of us performed from each of three sites [...]. This formidanle test actually ended up being more of a technical exercise than a full-blown concert. [...] In this case, the technology was so complex that we were unable to reach a satisfactory point of expressivity."

Later in 1997, *The Hub* musicians were asked for a new remote performance, which was called "Points of Presence", a live performance produced by the Institute for Studies in the Arts (ISA) at Arizona State University (ASU), linking members of the Hub at Mills College, California Institute for the Arts, and ASU via the Internet. Communication technology was not mature for musical usage at the time and the experience reported by member Chris Brown with networking at a distance was not adequate enough, as he explains in an article [5]:

"... the performance was technically and artistically a failure. It was more difficult than imagined to debug all of the software problems on each of the different machines with different operating systems and CPU speeds in different cities. In part, because we weren't in the same place, we weren't able to collaborate in a multifocal way (only via internet chats, and on the telephone); and in a network piece, if all parts are not working, then the whole network concept fails to lift off the ground. We succeeded only in performing 10 minutes or so of music with the full network, and the local audience in Arizona had to be supplied with extensive explanations of what we were trying to do, instead

[5]http://crossfade.walkerart.org/brownbischoff/IndigenoustotheNetPrint.html

of what actually happened. The technology had defeated the music. And after the concert, one by one, the hub members turned in their resignations from the band."

3.1.2 Current Networked Music Performance Trends

While at the beginning of the 2000s The Hub collective slowed down its activity, similar experiences started being common practice and nowadays laptop orchestras and NMP are widespread. The LAMC and The Hub experiences were fundamental for the development of current computer music performance. Following their activities these two important lines of music research, namely the laptop orchestra paradigm and the networked music performance, were created. They are both interesting as laptop orchestras may employ networking as well. Tens of music schools, universities and conservatories run a laptop orchestra. It is not by coincidence that many tutorial books on Computer Music, host a section on Networking or Laptop Ensembles. The Viennese Farmers Manual in 1995 formed a multi-laptop ensemble, while MIMEO (Music In Motion Electronic Orchestra) mixed acoustic instruments and computers from 1997. An extremely interesting ensemble for the purpose of this thesis is PowerBooks_UnPlugged, a collective of varying number of musicians only employing laptops. Not thriving into the details of their very interesting aesthetic, a point to highlight is the use of laptop and wireless networking to share data, algorithms and more. In the last ten years ensemble laptop performance has become common in the literature and a plethora of papers dealing with the aesthetics, the composition techniques or the software employed are published yearly on the proceedings of computer music conferences. Stable laptop orchestra are based in Princeton (PLOrk), Stanford (SLOrk), University of Colorado Boulder (BLOrk), Dublin, Huddersfield, Virginia Tech (L2Ork), etc. although not all these laptop orchestras employ networking. For more historical information read, e.g. [21].

Gathering information regarding the technical setups these orchestra employ for their performances is not trivial, as most often they are barely cited or neglected in favor of the aesthetic aspects. In the literature, technical aspects are often disregarded as computer musicians and composers prefer to focus on the artistic aspects and rely on widely adopted technology. This, however, has led to complete withdrawal of challenges offered by the new technologies, such as audio streaming between performers, the use of wireless networking, off-stage performance (with the notable exception of PowerBooks_UnPlugged and few others), the shift from laptop to embedded/embodied instruments and more.

On the other side, acoustic networked music performance is relegated to a niche in the literature. There are relatively few occasions for acoustic NMP, due to technical issues, albeit the human interaction potential is much higher than that available in a networked laptop performance [22, 23]. To address this some researchers are experimenting the use of visual cues in laptop ensembles introducing functionalities in the software employed by the performers [22, 23]. On the contrary, a *purist* approach is the use of high-bandwidth, low-latency fiber links allocated on purpose, which, although available at few institutions, provides much insight for research. Needless to say, the effort and challenges of this kind of NMP are numerous. Experiments have been going on both on LAN and WAN since more than a decade. Technically speaking, the first consistent attempt to address NMP over the Internet was conducted by the SoundWire group at CCRMA starting from year 2000 [24], which introduced bidirectional uncompressed audio streaming at low latency, employing the US Internet2 network. At McGill, the Ultra Video-Conferencing Research group started reporting about a similar approach, also including video, from the same year [25]. A first wave of experiments were spun in 2000 and following years, followed by a period of decay in the research community interest [16]. After a few years interest in Internet NMP started again to grow and maintained quite stable till nowadays. In Europe, the LOLA project [26] was conceived by GARR (the Italian research institutions network) and G. Tartini Conservatory of Trieste (Italy) in 2005 and developed since 2008 and now is growing its network of users[6].

[6]http://www.garrnews.it/index.php?option=com_content&view=article&Itemid=235&id=265:lola-

Besides the few aforementioned projects focusing on technical improvement, and although technical progress in communication technologies is highly relevant to NMP, most of the development efforts from the music computing research and artists communities have gone into software architectures and frameworks for NMP [27, 28, 29] or analysis and speculation of the NMP paradigm and aspects of Human-Computer Interaction [30, 31, 32]. Finally psychoacoustic studies related to the effect of latency on synchronization and tempo keeping have been addressed by several authors [33, 34, 35, 36]. Some more technically oriented works of recent publication deal with packet delay reduction by direct access to the network hardware [37] and low-latency error concealment [38].

3.2 Technical Issues in Networked Music Performance

So far, all the NMP categories have been addressed, to give a historical perspective. However, those contexts of peculiar interest in this thesis are those involving technical challenges, such as those where audio data is transferred in real-time, through LAN or at remote locations, e.g. through the Internet. All these research efforts generally rely on a very reliable link for audio (and often video) transmission, the Internet2 in the US and GEANT network in Europe. These are fiber networks connecting selected institutions (such as Universities) with high bandwidth (generally > 1Gbit) and low-latency, by manually routing the signals, reducing the number of switches in the path and assigning a high QoS to the audio/video signals. Both the LOLA and SoundWire projects designed their own applications for signal transmission. Stanford's CCRMA SoundWire group designed Jacktrip, detailed in Chapter 4, a simple multi-platform application for audio transmission. LOLA designed a performing Windows application sampling audio and video at extremely low-latency. The LOLA software is very hardware dependent, but allows for total latencies

il-conservatorio-da-il-la-all-innovazione of a few milliseconds also for the video signals. The project has a focus on the video latency and quality, to improve the interaction between musicians with clear and quick visual feedback.

The aforementioned projects mostly deal with NMP on highly reliable links. Although they stand as a reference for NMP and provide research insight on human interaction in music, they remove constraints typical of more common network infrastructures. The approach they use for NMP is called by Carôt et al. [16] RIA (Realistic Interaction Approach) or Realistic Jam Approach (RJA). In [39] many different approaches to NMP are reported, namely:

- Realistic Interaction Approach (RIA),
- Master Slave Approach (MSA),
- Laid Back Approach (LBA),
- Delay Feedback Approach (DFA),
- Latency Accepting Approach (LAA),
- Fake Time Approach (FTA).

The **RIA** is the most demanding, as it tries to simulate the conditions of a real interplay with the musicians in the same space. The general latency threshold for this approach is set by Carôt at 25 ms for the one-way delay (or latency), following an early technical report from Nathan Schuett at Stanford in 2002 [40], although more accurate studies exist in literature which suggest slightly different (but comparable) values. In the **MSA**, a master instrument provides the beat and the slave synchronizes on the delayed version that is coming from the master. The audio is in sync only at the slave side, while the master does not try to keep up with the slave's tempo, but he can barely get a picture of what is going on at the slave's side, trying not to get influenced in his tempo by the incoming delayed signal. Clearly, the interaction gets reduced in this approach, but the acceptable latency increases. The **Laid-Back-Approach** is based on the "laid back" playing manner, which is a common and accepted solo style in jazz music. Playing "laid back" means to play slightly behind the groove, which musicians often try to achieve consciously in order to make their solo appear more interesting and free. Similarly to the MSA, at the master side the beat is built and at the slave side, a solo instrument can play. At the master side, the round trip delay, if higher than 25 ms but below 50 ms, creates an artificial laid back style of playing. LBA of course doesn't work for unison music parts in which both parties have to play exactly on the same beat at the same time. A commercial software Musigy exist that implements LBA. A **DFA**, tries to fill the latency gap between the two ends by introducing an artificial delay in the listening room at the master end (if one of the ends have a master role), or at both ends (if interplay is not hierarchical). In the first case, e.g., delaying the master's signal in the room allows to make it closer to the slave's delayed signal. Similarly for the non-hierarchical case. The approach however, introduces a latency between the user action (e.g. key press) and aural response (in case of an electronic instrument) or simply adds a delayed version to an acoustic instrument, which can deteriorate playing conditions, however is suitable with turntables or sound sources with little human interaction. A commercial software, eJamming, employs this approach. The **LAA**, simply neglects synchronization and is used for contemporary avantgarde music, music with very low timing constraints or computer music which employs the network as part of the performance. The SoundWire group promoted this approach with several performances of contemporary music.

Finally, an approach that accepts latency but allows for tempo (but not beat) synchronization, is the **FTA**. In this case the latency is artificially adapted to be one measure or multiples. This way, any performer plays on the previous measure executed by the other performer. This approach requires a tempo to be known a priori and fixed. A further hypothesis is needed, that the music does not change drastically from measure to measure, which is the case for many improvisational genres, such as blues, funk, etc. The Ninjam open source software employs this approach. For a different, more aesthetics-related classification of NMP, refer to [19].

3.2.1 Dropout

Several technical difficulties to overcome are outlined in this section, in order of importance. Audio dropouts, i.e. loss of audio packets, must be avoided by any means, since a regular audio flow is the mandatory requirement for audio transmission. Source of audio dropouts are manifold:

- loss of packets along the route
- late arrival of packets at the receiving end
- loss of transmitter and receiver audio clock synchronization
- failure in the scheduling of the audio capture or playback process

Loss of packets is often solved in networking applications by triggering a new transmission after a certain timeout time has passed. This is done, e.g. in the TCP transport-level protocol, or in wireless protocols it is done at lower levels when the received packet checksum is incorrect. However such procedures require time and are not suitable for time-critical application such as NMP (unless transmission time is $\ll$ than the deadline imposed by audio buffering). Furthermore, in NMP the only viable solution is to trust the network robustness to failures and losses or to apply some redundacy, in order to greatly reduce the probability of lost packet. Under the hypothesis of a random distribution for packet loss, redundacy can be exploited. However, if the losses are correlated, e.g. due to a link prolonged failure, redundacy provides no improvement.

Packets may also arrive at the receiver after the time they should have been sent to the audio card for playback. This is discussed in Section 3.2.2. Moreover, when the clock rates of the transmitter and receiver audio cards are not synchronized, packet loss may occur as described later in Section 3.2.4.

Finally, one of the two machines, may fail in correctly scheduling the audio processes, so that the audio card is not timely supplied with or read for new audio data. Issues with process scheduling is briefly recalled in Chapter 4, where a practical implementation of audio capture and playback under a Linux operating system is described.

Unfortunately, even with a good NMP deployment and best networking conditions, it is a good practice to consider occurrence of dropouts. Loss concealment is a good practice to avoid all the dropouts due to the network and clock issues. For this reason many audio coding and decoding algorithms also include error or loss concealment. Indeed, when dropouts cannot be avoided, the last tool to resort to is psychoacoustic masking. Currently, the OPUS codec (formerly CELT), is widespread in audio streaming application for its very low delay (less than 10 ms), objective audio quality and loss concealment [41].

Errors can be concealed too, and though communication technologies are nowadays very robust to errors, a packet may still get corrupted. Generally packets containing errors (which are calculated through checksum or similar mechanisms) are discarded at the receiver end, however, some transport protocols such as UDPlite allow the user application to forward a corrupted packet for further error correction or concealment in software.

While planning a NMP deployment, it is a good practice to figure out an upper bound to dropouts during a session. A strict bound could be zero dropouts tolerated for an entire session. A more loose constraint may be q dropouts per session duration s, i.e. a probability

$$p_t \leq q/s. \tag{3.1}$$

In Section 4.7, results are reported for packet loss over a wireless link. In that case the PLR, i.e. Period Loss Rate is reported, that is the rate of audio buffers that are lost, which is more psychoacoustically meaningful than network packets. An adequate psychoacoustic research work could investigate further on the topic, to link PLR to perception of glitches from the audience. It is possible, e.g. that the loss of two consecutive or very near audio periods is perceived as one glitch. Furthermore, the audience may not expect dropouts at first, thus, cognitive effects may make the listener unconsciously discard a dropout.

3.2.2 Latency

As reported above, a variable amount of latency is inherent to NMP and determines the feasibility of the performance and the approach to consider. Along the thesis, only the RIA approach is considered, being the most technically challenging and requiring the lowest possible latency. Where not otherwise specified latency is defined as the delay it takes for the sound to reach the remote listener from the source, while the Round-Trip Time (RTT) will be referred to as the time the audio takes to travel from the source to the remote end and back to the source. The RTT is important in assessing the ability of a NMP setup to build an interactive interplay between musicians.

As reported above, Carôt takes the 25 ms figure as a threshold for latency in RIA, drawing from [40]. There are, however other studies yielding different values and points of view. First and most importantly, most studies conclude that the tempo change slope can be fitted with a linear model to a good approximation, thus, with increasing delays the tempo decreases [33, 42]. In [43] a simple model for human performer tempo deceleration based on perfect memoryless tempo detection is first assumed and then proved inaccurate in predicting human performers in an NMP context. The model assumes that tempo M decreases at each round according to

$$M(n) = \frac{60}{c_0 + nd} \tag{3.2}$$

where $c_0 = 60/M_0$ i.e. the quarter-note interval in seconds with initial tempo M_0, n is the quarter note sequential number and d is the delay imposed by the network. A steady tempo would be kept, following this model only for $d = 0$. Tests proved humans to perform differently than the memoryless model. For this reason Driessen et al. [35] investigated on human player modelling in the delayed performance scenario, following coupled oscillator theories, summarized in their paper. By fitting data a good model was found to approximate the human player behavior, i.e.

$$M = M_0 - kM_0d \tag{3.3}$$

where the constant k is found to be $k \simeq 0.58$ from data fitting. The authors hypothesize that results obtained with simulated NMP setups are probably to be found in an acoustic environment where the subjects are situated at a sufficient distance to impose propagation delays similar to those of an NMP setup. The reader must not forget that given the sound propagation speed in STP conditions each meter adds approximately 3 ms delay. In an orchestra, for instance, a maximum of 46 ms is reached between the most distant players (hence the need for a conductor as a visual cue).

In [33, 34] it is found that at delays $\leq 11.5ms$ the tempo accelerates. Farner et al. [42] report the same conclusions. Values are however different. In [42] musicians and non-musicians are discovered to have different thresholds of acceleration, respectively 15 ms and 23 ms. It must be noted, however, that the musicians were subject to a complimentary rhythm task (see Figure 3.3) while the non-musicians performed the same rhythmic pattern. It is unknown whether this difference was relevant to the results. Imprecision in timing was also evaluated in this study and delays over 25 ms were found to introduce imprecision. Acoustic conditions were relevant to imprecision: anechoic conditions imply a higher degree of imprecision, while room-reverberant conditions imply a slightly lower tempo.

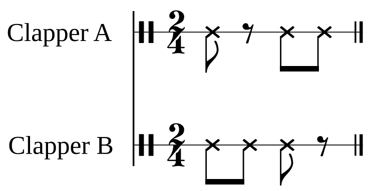


Figure 3.3: Score of the clapping pattern used in Farner's and Chafe's complementary rhythm tests.

All the aforementioned papers conducted tests at 90 BPM. Other tempos are worth investigating to obtain a more general model. A last related study [36] reports results similar to the above, and states that until 60 ms the performance can endure harmoniously, breaking down only over that value.

* * *

After this brief report on psychoacoustic findings related to latency, what are the factors that introduce latency in an audio link? With current audio equipment, one unavoidable source of latency is that related to audio buffering after or before the A/D and D/A conversion. The A/D and D/A converters are explicitly designed to convert audio in real-time (the conversion time is totally negligible in this context), however, the issue is with the data storage and buffering. Since in most modern networking technologies data is exchanged in packets and computing architectures are more efficient with chunks of data - rather than on a sample-by-sample basis - the only feasible way to treat audio data in this field is to store and exchange buffered data. The buffer size and the sampling rate determine a blocking time, or a **buffer time**, which delays operations such as transmission. Thus, for a simple audio link, with one end acquiring and transmitting, and the other receiving and playing it back, a minimum latency is imposed by one buffer time at each end. The buffer is filled of several audio slots (at least two), called *periods*. Typical values for the period size with modern computing architectures are 128 to 512 samples at 44.1 to 96 kHz, i.e. 1.3 to 11.6 ms. The buffer time is not hard-wired, but depends on a trade-off between CPU time and latency, since short buffer times lead to a higher interrupts frequency, and more operating system overhead. Embedded hardware for music employ shorter period sizes, e.g. 16, 32 or 48. For the sake of comparison, a digital mixer, typically imposes a input-output delay to the signal of 1.6 ms or 0.8 ms, while many digital keyboards output sound at 32 kHz and a period sizes of 16 to 64 samples.

On WAN the most obvious addendum to latency is the propagation delay. Depending on the medium, be it copper wire or optic fiber it can be somewhere around 0.7 times the speed of light in void, i.e. about 5 ms every 1000 km. Unfortunately, the routing between distant locations can be much longer than the direct path. With distances below 100 km or LANs, however, the switching delay can be much more relevant than the signal propagation, introducing a time-varying delay (or packet delay variation, PDV), which is unpredictable, what is called Network Jitter (not to be confused with Clock jitter). When the network includes a DSL link, overhead and other issues are also to be considered, detailed in [44]. Any filtering, coding or compression applied to the digital audio adds extra latency. Only a very few audio compression algorithms provide latency of a few milliseconds. This is why in many NMP projects it is preferred to stream uncompressed audio.

3.2.3 Delay Jitter

The network delay is not constant, and can quickly vary from packet to packet. Network delay is subject to jitter. Scheduling of the audio process at the two ends is also subject to a small delay random delay (interrupts of higher priority may postpone execution of audio processes or delay in the processing of packets queued at the network drivers level of a kernel, which, to optimize throughput, are not sent at once).

A brief formalism for network delay jitter is provided. Abstracting from the communication medium and network topology, a certain delay will occur between the transmission of a packet and its reception by the recipient. This delay is subject to jitter, i.e.

$$n(i) = \bar{n} + \nu(i), \tag{3.4}$$

where the time-varying delay $n(i)$ depends on a constant component $\bar{n}$ (propagation speed, switching, etc.) and a stochastic process $\nu(i)$. The stochastic component cannot be predicted, thus a buffering mechanism must be employed. Choice of the buffer size depends on the shape of the probability density function (PDF) estimated for $\nu(t)$. If the PDF $\nu(\tau)$ reaches zero at ∞, a probability - however small - of dropout must be tolerated. A probability of tolerated dropouts p_t in the form of Eq. 3.1 must be chosen. In the case, e.g. of one dropout tolerated over a 1 hour performance

$$p_t = \frac{1}{\frac{F_s}{P} \cdot 3600} \tag{3.5}$$

with F_s being the sampling frequency, P the fixed packet payload size in terms of audio samples. Once the PDF is known, the maximum delay D_M is the delay value for which the residual area (the tail of the PDF) is $A(D_M) \leq p_t$, as clarified by Figure 3.4. The buffer size should be $\geq D_M$ to accommodate for late packet arrivals up to the maximum delay. For each specific application, the PDF should be characterized. This, however, is often difficult, and the PDF may be highly time-varying. It also does not take into account other impacting factors such as momentary link failure due to accidents, network failure, power shortages and such. Thus, more pragmatic strategies for buffer size choice may well be employed.

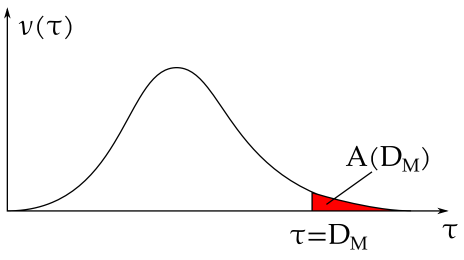


Figure 3.4: An example Probability Distribution Function of the jitter and choice of the buffer size accounting for the maximum tolerated delay D_M.

Jitter cannot be avoided, but mitigated or compensated for, by estimating a worst case and allocating sufficient buffering to allow late packet arrival. However the worst case condition may generate a buffering latency too high to sustain NMP, therefore a trade-off must be done. In that case a late packet may generate a dropout, i.e. arriving too late for the software to send the data to the audio hardware.

To summarize, a few graphical representation of the concept described above are proposed. Figure 3.5 reports the delays to take into account for an audio link:

- the period time, i.e. the time for a period to be recorded (orange),
- the delay involved with calling an interrupt service routine, waiting for the audio thread to be scheduled by the operating system the timestamp for this period to be calculated and the packet to be put into a queue for transmission,
- the delay to cross the network and reach the receiver,
- the receiver queues the period into a buffer, according to its own audio card period rate when an interrupt occurs the period is copied into the hardware buffer for later playback.

It must be noted that the network jitter effect is partially removed by the buffering mechanism. In Figure 3.6, e.g., a packet affected by a random delay and arriving in the range $t_i < t < t_{IRQ2} - \epsilon$ (with ϵ very small) is played back in the same audio card time slot. Provided that the receiver incorporates a packet reordering mechanism and the packets are provided with a counter by the transmitter, a packet arriving between T_i and T_{IRQ2} can be played back properly. Furthermore, latency is insensitive of the exact arrival time, due to quantization of time in periods.

In Figure 3.7 an example transmission where the latency is initially set by the transmission delay imposed by packet 1, incur into dropout of packet 4 when the network delay for that packet exceeds the buffering capacity of the receiver. A larger buffer would take into account for additional jitter in the network delay and, accordingly, allow packet 4 to be scheduled for playback notwithstanding its large delay.

3.2.4 A/D and D/A Clock Synchronization

Even on an ideal network, there is another issue that is worth addressing which is audio card clock skew, i.e. a slight difference between the nominal clock and the actual value. This may be due to errors in components rating as well as a change in environment conditions such as temperature. Even a small difference between the two remote ends clocks can lead to dropouts in a relatively short

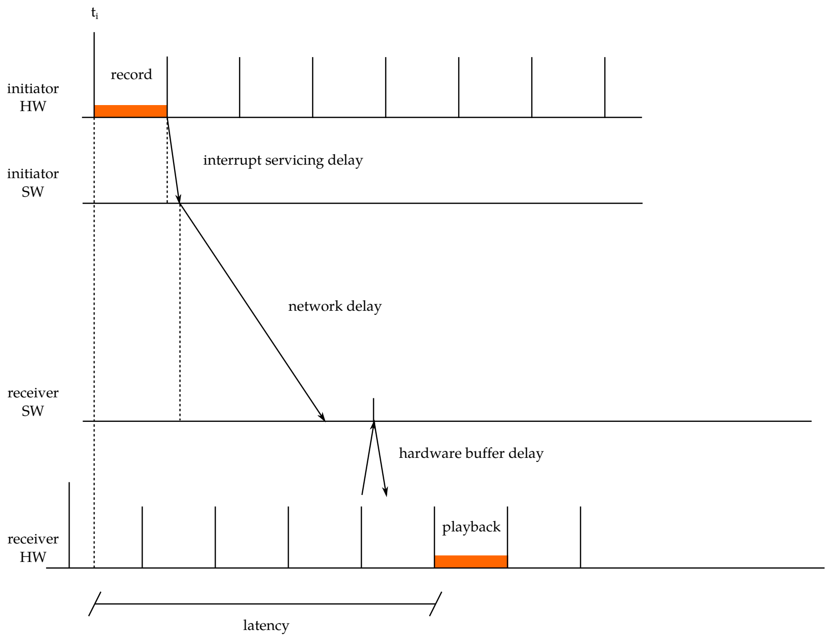


Figure 3.5: Delays related to an audio link. Interrupt servicing delay, network delay and queuing delay. All these are subject to latency.

time frame. For instance, a 20-60 ppm skew (e.g. 1-3 Hz at 48 kHz) with short buffers can lead to dropout at the receiving end in a few minutes. Assuming a buffering mechanism at the receiving end compensating for network jitter, of N slots of B samples each, and assuming (for simplicity) that at both ends the machines process audio buffers of B samples with the same nominal samplerate but different actual clock frequencies F_1 and F_2, a dropout will occur every

$$\Delta = \frac{N \cdot B}{|F_1 - F_2|}. \tag{3.6}$$

For a skew of 1 Hz at 48 kHz nominal samplerate with buffers of 128 samples and a circular buffer of 4 slots this yields to 8.5 minutes, i.e. 7 dropouts in 1 hour, which is not acceptable for a regular performance. Depending on the buffering mechanism the Δ may be even lower. Let us consider a circular implementation of the buffer, i.e. a ring buffer. Without prior knolwedge regarding the two clocks frequency the safest way to initialize the read and

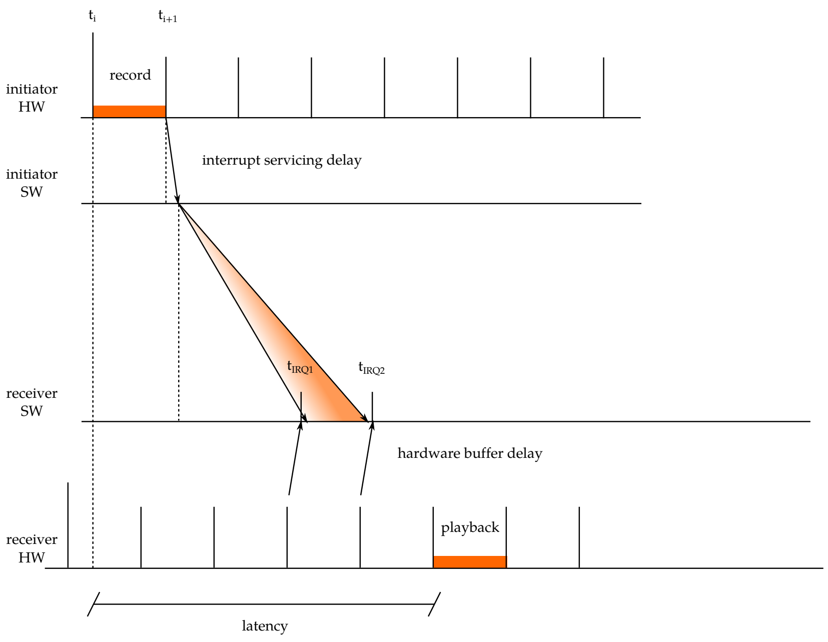


Figure 3.6: A packet arriving slightly after IRQ1 or right before IRQ2 due to different network delay has the same effect on latency due to quantization of time in periods. The larger the period (and the buffering) the more tolerant the system to network jitter.

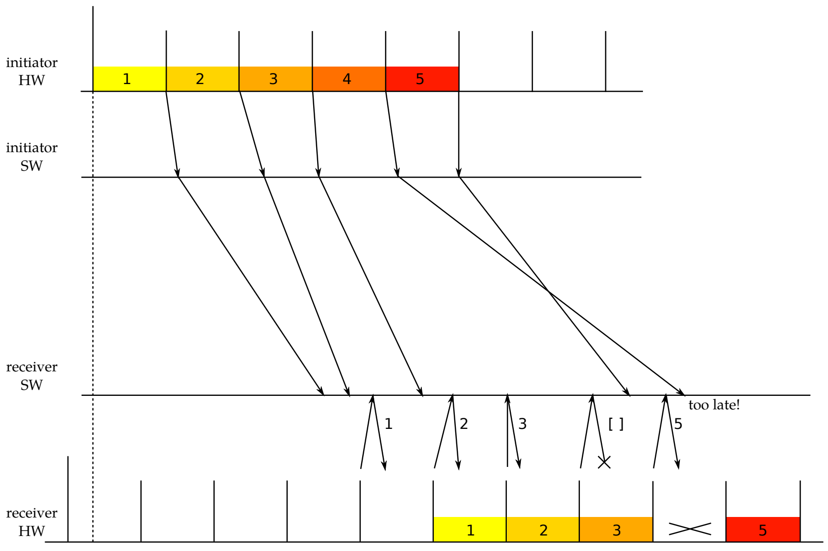


Figure 3.7: An example of packet dropout due to network delay jitter. The larger delay imposed by network to packet 4 forbids it to be played back as it arrives too late. The average latency is imposed by the first packet.

write pointers is at the two furthest position, i.e. one at the slot $i = 1$ and the other at slot $j = N/2 + 1$ (assuming an even N). The under-run condition occurs when the read pointer proceeds faster than the write pointer (i.e. the remote end has a slower clock) and reaches the write pointer slot. The over-run condition occurs when the write pointer proceeds faster than the read pointer and finally has no empty slots to write to. With the aforementioned initial conditions i, j, the time between two consecutive overruns Δ_o or underruns Δ_u is $\Delta_o = \Delta_u = \frac{1}{2}\Delta$.

To recover from underrun state, the read pointer is spun back m slots, thus it reads again part of the previous audio (or simply null data if these were erased after reading). To recover from overrun, recovery is done by advancing the write pointer as it is supposed to do and place the read pointer m slots ahead. With such a ring buffer implementation, if

- $F_1 > F_2$: $\Delta_o = \frac{m}{N}\Delta$,
- $F_1 < F_2$: $\Delta_u = \frac{m}{N}\Delta$,

Clearly, m can be at best $m < N - 1$. The closer to 1 the ratio m/N is, the closer to the ideal case.

To express the problem in more rigorous terms the following formalism is introduced. Any hardware clock source is employed to obtain a system time that is used in software for many purposes. A clock can be seen as an oscillator generating periodical events and a (software) accumulator that increases its count at each event. An accumulator function, or time function $T(t)$ [45], is a mapping between the real time t and the clock events, i.e. a piecewise continuous function that is twice differentiable

$$T : \Re \longrightarrow \Re. \tag{3.7}$$

Let $T_1(t)$ and $T_2(t)$ be two independent clock generators, and $T_1'(t)$, $T_2(t)'$ their time derivative, then the:

- **offset**: is defined as the difference $\rho_{21} = T_2(t) - T_1(t)$;

- **frequency**: is the rate at which a clock progresses. The instantaneous frequency of the first clock at time t is $F_1(t) = T'_1(t)$;
- **absolute skew**: is the difference between a clock frequency and the real time, e.g. for the first clock $\beta_1 = T'_1(t) - t'$;
- **relative skew**: is the difference between two clock frequencies, e.g. $\sigma_{21} = T'_2(t) - T'_1(t)$;
- **clock ratio**: the ratio between two clock instantaneous frequencies, e.g. $\alpha_{21} = T'_2(t)/T'_1(t)$.

When a sentence clearly refers to the frequency difference between two clocks, the relative skew is simply called skew.

The general model for a clock time function is

$$T(t) = \beta(t) \cdot t + \xi(t) + T(0), \tag{3.8}$$

where β is the absolute skew, which determines the slope of the function, $\xi(t)$ is a power-law random process modelling jitter of the clock events and $T(0)$ is the initial value of the function. with c onstant $F_1.F_2$, the relation between clock ratio and skew is

$$\sigma_{21} = F_2(t) - F_1(t) = \alpha F_2 - F_1 = (\alpha - 1)F_1 \tag{3.9}$$

In non-ideal oscillators, unfortunately, the frequency $F(t)$ is slowly time varying and departs from the nominal value F_0, i.e.

$$F(t) = F_0 + F_e + \nu(t), \tag{3.10}$$

where F_0 is the nominal frequency, F_e is a frequency offset, i.e. a constant departure from the nominal value and $\nu(t)$ is a slowly time-varying stochastic process.

Instantaneous synchrony takes place when $T_1(t) = T_2(t)$, meaning that timestamps generated by two different machines at a specific time t are the same. Of greater interest for a long run synchronization of two remote ends is the case when $T_1'(t) = T_2'(t)$, i.e. two clocks have same pace, and $\beta_1 - \beta_2 = 0$. Otherwise the offset between the two time functions diverges $T_2(t) - T_1(t) \to \infty$ for $t \to \infty$. Clock frequency synchronization is also called along the essay *relative time approach*, while *perfect* or *absolute synchronization* is the case when both $T_1(t) = T_2(t)$ and $T_1'(t) = T_2'(t)$.

Several synchronization mechanisms are proposed for wireless networks. In [46], e.g., a mutual synchronization mechanism between two or more ends is proposed employing a control loop that minimizes the error between two time functions. This mechanism called CS-MNS, is an absolute synchronization mechanism. In the audio case, however, the time function generally starts when the audio process is started. Relative time synchronization is already sufficient since two time functions may have different offset if two audio processes are started independently. A last approach worth to mention here was proposed by Carôt [47] that employs an external frequency generator driven by software. The frequency generator replaces the sound card clock. Unfortunately the approach requires a frequency generator and a sound card that is capable of receiving a so-called word-clock, i.e. an external clock reference. Two approaches for synchronization through resampling are provided in the next chapter.

It must be noted that professional equipment is prone to clock skew as well as inexpensive OEM hardware.

3.3 Communication Technologies for Networked Music Performance

Communication technologies are leveraged daily in the music broadcasting and distribution fields [48]: Audio-over-Ethernet technologies are widely used for the local area [49, 50] (broadcasting studios, large rehearsal studios and so on), and Audio-over-Internet is widely adopted for broadcasting contents to the end-users [51, 52, 53]. The Internet and 4G radio communication deliver music contents to hundreds of millions of users daily. These technologies, however have no connection with music interaction and performance, since they are designed for one-way multimedia streaming. As a general purpose network, however, the Internet has been explored for compositional purposes and NMP [54]. One of the most notable technical results is the use of research fiber network links for audio and video transmission in the context of real-time music performance, first explored in the late 1990s [55] and nowadays exploited for a number of performances [26, 56]. Wired Local Area Networks are well known to sustain audio communication, and even distributed computing [57]. Wireless technologies are, however, relegated at the moment as an auxiliary feature for remote control of digital devices such as mixers and such, neglecting the potential that wireless networking can have even in the mission critical field of music performance.

3.3.1 Suitable Wireless Communication Technologies

So far a general introduction to NMP has been provided, with many example related to Internet NMP or wired LAN NMP. Wireless communication technology are ready to deliver robust audio transmission for NMP? At the time of writing, the use of wireless communication is limited to:

- analog or digital point-to-point unidirectional links transmitting microphone or guitar signals to custom receiving stations, to allow speakers or musicians for a higher freedom of movement,
- unidirectional A2DP (Advanced Audio Distribution Profile) Buetooth links from a smartphone or music player to a public address speaker system or loudspeaker system.

Either options feature a point-to-point unidirectional signal transmission. No networking can be performed and different devices communicate on different channels. Bluetooth links are designed only for playback of compressed audio, and do not provide low latency.

Another interesting field is that of home cinema loudspeaker, where wireless networks are used for unidirectional transmission of audio packets to the loudspeakers. One interesting development [58], shows that 802.11n networks can provide a high-bandwidth streaming of audio with a very low latency (in [58] a unidirectional stream of 8 audio channels at 192kHz 32-bit with audio packets as small as around 50 samples is reported). Commercial applications are available since years and most of them rely on WiFi links, with a few exceptions[7] [8], motivated by the need to differentiate from other products in terms of reliability, bandwidth and features.

These wireless technologies alternative to 802.11 may be appealing in terms of throughput, robustness to interferences, latency and bandwidth. Skaa[7], for instance, is based on a proprietary protocol and System On a Chip from Eleven Engineering Inc., which implements an adaptive frequency hopping algorithm (called Walking Frequency Diversity, patented), to dodge potential interferers in the 2.4GHz ISM band, and a set of constant latency values, down to a minimum of 10 ms, with a maximum latency offset between two receivers of $40\mu s$. The protocol allows only for unidirectional signal transmission, although communication is bidirectional to collect RF link statistics and ask for packet retransmission.

Two silicon manufacturers that provide a *wireless audio* solutions are Texas Instruments and Microchip, among others. TI provides the CC85x family of ICs, that operates at maximum 5 Mbps in the 2.4 GHz ISM band, avoiding interferers by adaptive frequency hopping and listen-before-talk, and providing forward error correction, buffering, retransmission to minimize errors. The audio links are CD quality, maximum 4 channels unidirectional and the minimum latency 10.6 ms. These characteristics make it unfeasible for NMP. Microchip branded the KleerNet technology that is targeted at unidirectional audio streaming to loudspeakers and wireless microphones. It operates in the 2.4, 5.2 and 5.8GHz bands and guarantees an indoor range of up to 60m. The marketed ICs host a large set of features, including audio processing and user input. The latency is declared to be under 20 ms and clock sync may be achieved by means of digital interpolation. Again, this technology is not

[7]http://www.sonos.com/
[8]http://www.skaa.com/

feasible for quick exploitation in wireless NMP.

A last interesting opportunity to mention is the use of sub-1GHz ISM bands for increased range. Analog wireless audio transmitter for stage use are traditionally in the VHF band. Analog or digital UHF equipment also exists and is widely used. No networking protocol is associated, however, to any sub-1GHz technology that has sufficient bandwidth for high quality audio, as these protocols are meant for wireless sensor networks and similar uses.

Once all the aforementioned wireless solutions were discarded, the only good candidate remaining for a viable implementation of a wireless NMP system was the widespread IEEE 802.11 family of protocols. At the physical level, the 802.11a, b, g, n and ac protocols rely on the 2.4GHz and 5GHz ISM bands. By employing different modulations, channels and antenna diversity techniques they obtain very different bandwidths and coverage ranges.

The 802.11a and 802.11g are very mature and are of interest for their wide acceptance. They reach a maximum raw data rate of 54Mbps, in the 5GHz and 2.4GHz bands respectively. The 802.11b first extended the 802.11a to the 2.4GHz band and achieved a maximum raw data rate of 11Mbps. Considering that the raw data rate only applies in the best case of transmission conditions and that error correction and overhead must be taken into account, the effective throughput is significantly lower. IEEE 802.11 is half-duplex, hence, for bidirectional audio transmission the required bandwidth must be correctly estimated to fit the effective available throughput. In the case of multiple stations the available bandwidth must be divided by the number of stations and extra overhead due to collisions and coordination must be considered.

The 802.11n amendment improves on the previous ones by achieving a data rate of maximum 600Mbps by introducing MIMO (Multiple Input Multiple Output) antenna diversity over both 2.4GHz and 5GHz bands. A last amendment recently implemented in devices and access points is 802.11ac, which further increases the data rate by increasing MIMO number, channels width and modulation order. Typical maximum raw data rates obtained with this protocols are in the order 1300Mbps.

Critical factors in using 802.11 protocols for NMP follow:

- the 2.4 and 5 GHz ISM bands are crowded by several communication technologies and electric appliances
- medium access may sensibly delay audio transmission, especially with increasing size of the network
- the hidden station problem may arise

Regarding access to the medium, the DCF (Distributed Coordination Function) is a medium access protocol meant to be fair with respect to all stations, it does not cope with time-bounded traffic. For this reason the PCF (Point Coordination Function) is also present which allows a coordinator to schedule transmission during a contention-free period. The DCF and PCF are not mutually exclusive and can coexist in a superframe. A superframe includes a contention period and a contention-free period.

In [58], 8-channel unidirectional audio transmission is reported with 802.11n in the 5GHz frequency band, at 108Mb/s data rate, employing antenna diversity. Each channel carries 32-bit audio at 192 kHz and is delivered using multicast. Audio is transmitted in the contention-free period using PCF. A contention period is available for additional data transmission. The contention free period lasts 2.7 ms and is employed to transmit 12 audio blocks of approx. 50 samples each (approx 3 ms of audio). UDP multicast packets are sent and not require an ACK. Forward Error Correction is employed to recover from bit errors. In the prototype implementation two access points (built on desktop Linux machines) have been used to handle the high traffic. This work proves that IEEE 802.11 is worth considering for wireless audio streaming. It also shows that much low-level development must be done to make such a system work properly.

Chapter 4

WeMUST: the Wireless Music Studio project

Some of the technical challenges in the choice and design of platforms for music signal processing and NMP have been reported in the previous chapters. This overview make it easier to discuss the development of the Wireless Music Studio, in short WeMUST, a project conceived to explore the possibilities offered by wireless networking in the music performance and studio context with a special focus on technical challenges and their possible solution with widespread hardware and protocols.

The key concept underneath the investigations within the WeMUST project is the use of wireless transmission and IP networking to enable audio and control signal transmission in music recording and performance. Neither of the concepts is a novelty in itself: point-to-point radio technologies have been available commercially since a few decades for large stage usage and networking is currently employed with wired studio networks based on several Audio-over-Ethernet standards. However the adoption of general purpose communication technologies (in this case IEEE 802.11 and IP networking) in challenging scenarios such as those under investigation within the project have been scarcely addressed by academic research, and is still to be considered by the industry. The reason for the latter may partly be due to a negative bias among musicians and technicians towards digital wireless technologies reliability, re-

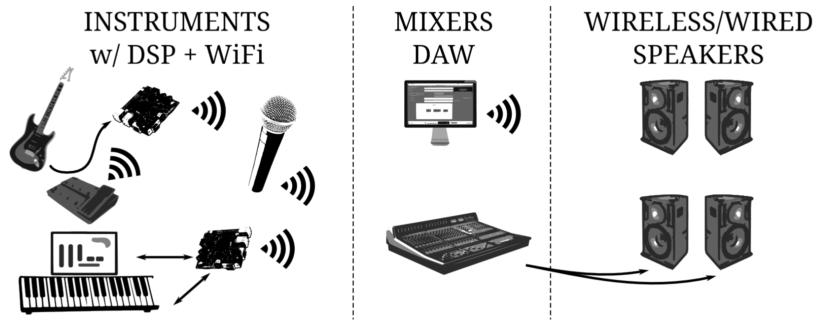


Figure 4.1: A schematic view of possible scenarios enabled by the WeMUST architecture.

ducing the commercial potential of new products featuring such technologies. A common-ground experience regarding wireless networking is most probably that of personal computer and mobile devices relying on 802.11 protocols. While this set of protocols is mature and globally adopted, users may still experience troubles due to its complex software architecture, to signal coverage, interference and channel crowding, etc.

4.1 Application Scenarios

By taking advantage of wireless communication, a portable platform and, where possible, battery energy storage, several application scenarios are possible. Figure 4.1 depicts some of them.

Home or small studio music production systems can be envisioned. In such a scenario musicians can setup their instruments in the live room, turn their transceivers on and quickly connect to WeMUST studio-provided amplifiers and loudspeakers. The signal can be routed at the same time to a digital mixer for recording in the monitoring room. In a home studio all the instruments including MIDI controllers can be routed to a PC acting as DAW (Digital Audio Workstation). The computational workload can be distributed among several devices in a network, relieving the DAW running on a PC from part of the processing, following recent experiments with LAN [57, 59]. For instruments such as guitars which require effects to be applied, either the clean signal from the instrument either the output from the amplifier can be streamed to the recording mixer. In the former case applying more specific effects is left to post-production while the musician can perform with a sound he feels comfortable playing with. If a technician is not required the musicians can control the digital mixer in the monitoring room together with the multi-track recording gear with a tablet device directly from the live room.

Furthermore the time consuming task of deploying cables from the front and stage mixers to the stage, installing microphones and sound checking can be avoided and made more reliable with the use of a wireless technology that allows to quickly and flexibly connect the sound sources to the mixers. No check need to be performed after the devices are connected and each instrument can diagnose the network reliability and bandwidth with ease by automatic software checks.

Common USB musical keyboards and controllers can thus convey the MIDI or OSC data to the PC wirelessly. Regular expander racks and keyboards can also be fitted with a WeMUST transceiver to send audio data directly to the PC. If they are fitted with hardware controls or a touch surface they can control the mixing parameters. The same wireless architecture can be also employed for control of live lighting, smoke and fog.

Devices that transmit and receive audio can also manipulate audio. The envisioned hardware platform must be, thus, also capable of performing DSP.

Avoiding cables may also reduce the risk of ground loops that introduce hiss and hum into the audio equipment and avoid risk of accidental tripping over cables.

Drawbacks must be highlighted as well. In that regard, the linux-audio community mailing list provided several critiques in the last two years, including latency, cost and reliability of the solution in terms of dropouts. Reliability as with any kind of wireless communications is related to time-varying fading and multi-path which decrease the signal quality, especially with moving people or objects. Furthermore wireless technologies are nowadays of public concern for health related issues, hence attention must be paid to reduce as much as possible the amount of radiated energy from devices in a given space. Directional antennas may help in reducing the radiated energy.

4.2 WeMUST: System and Topology

As a whole, WeMUST is currently a set of open hardware and software (from now on HW and SW) tools designed to fulfill the requirements specified in the introduction of this Chapter. The supported HW is the Beagleboard xM (BBxM), based on an ARM Cortex-A8 core, and any x86-powered personal computer running Debian or similar Linux distributions. Additional HW that may be required for specific purposes is detailed in Sections 4.7, 4.8.1. Other audio equipment such as high quality sound cards can possibly be employed but will not be discussed here. A WeMUST system can be, thus, built with rather inexpensive hardware.

At the audio level, the networking paradigm is that of a peer-to-peer network, where every node can act as a transmitter and receiver, with no hierarchical pattern. This is required to allow for the maximum flexibility of the network topology that only requires two nodes of any kind to be active in the network. For this reason, any master-slave scheme, as this enforced with other software tools [1] is disregarded. Nodes of this network can independently enact any of the following: acquire and/or emit acoustic signals, transmit and/or receive digital audio or control signals, record and store signals. Nodes receiving signals from multiple nodes and/or recording them are called *mixing* nodes (in short MPCs in the case of personal computers). Nodes can also monitor the network for debug purposes and issue commands to other nodes for connection, shutdown, activation, etc. These nodes are called *supervisors* and are not necessary if the other nodes are programmed to act independently or are controlled by the user from a user interface (e.g. buttons). A simple diagram exemplifying a WeMUST network with three nodes, one mixing node and a supervisor is depicted in Figure 4.2.

An example setup is shown in Figure 4.3, where an MPC can transmit control data (e.g. coming from a DAW or an algorithm), a click track for musicians synchronization, and receives audio data for recording. The nodes can capture or synthesize audio, or only be employed for remote MIDI control.

[1]such as *netjack*, the network driver built in JACK

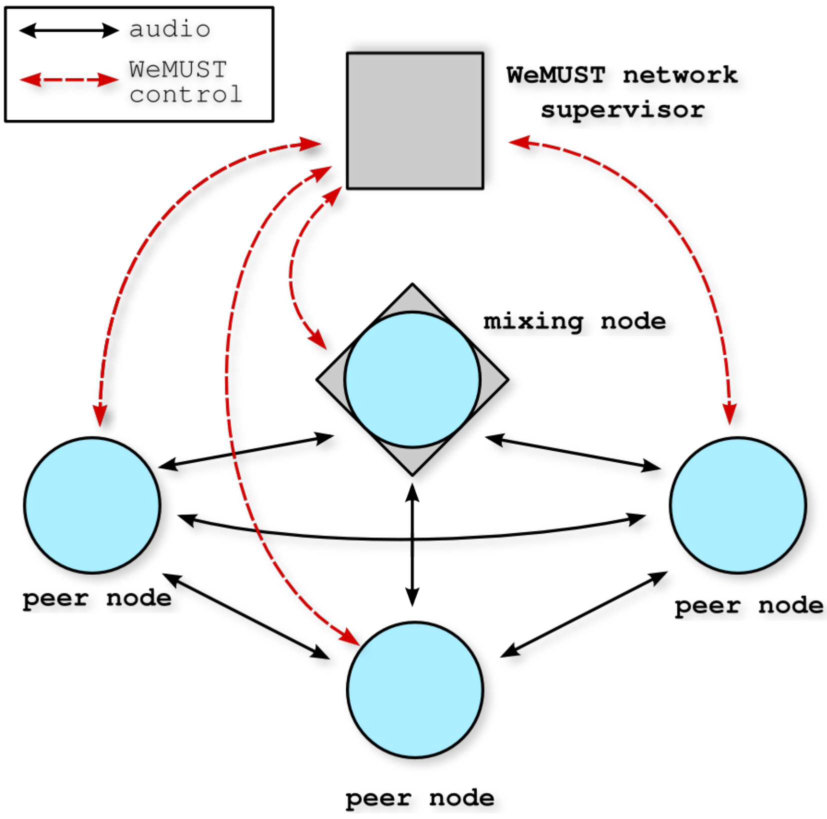


Figure 4.2: Schematic view of a WeMUST network comprising three nodes and a mixing node mesh-connected. The number of nodes and their connection can be arbitrary. A supervisor can scan the network for devices, control the devices and their connections, receive debug messages.

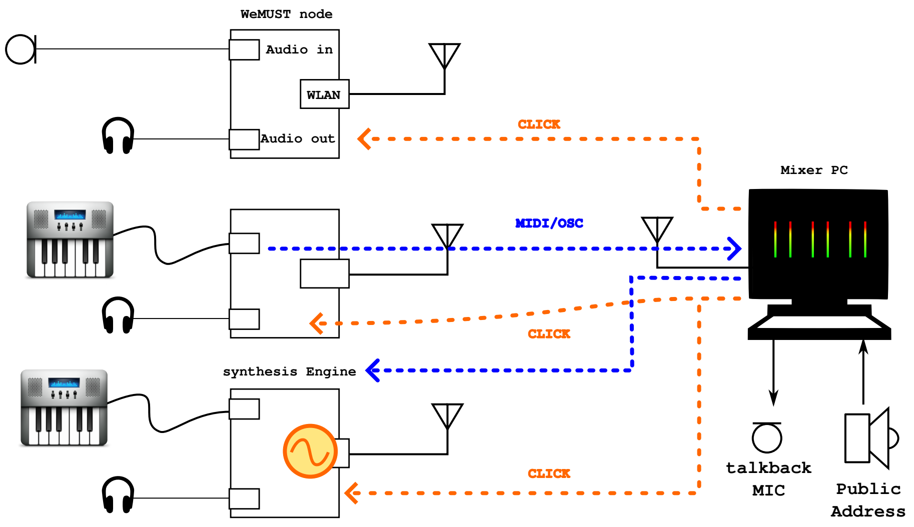


Figure 4.3: Example of a WeMUST setup, involving audio and control data transmission between embedded devices and an MPC.

Each node must be connected to the network through a suitable wireless or wired interface, currently a WiFi transceiver or an Ethernet controller. Commercial wired routers and/or wireless Access Points are required. To establish a connection with other nodes connection parameters must be configured by the user. A script is provided to help the user configure the OS during installation. Implementing the SABy protocol [60], devised for use in WeMUST similar contexts, facilitates connection. SABy is based on a simple multicast and unicast messaging system (conceptually similar to UPnP) and DNS.

The nodes can run WeMUST OS, a specific Debian image customized for the goal at hand. WeMUST OS already contains all the packages required for the applications targeted by the WeMUST project. As such, all the software is based on GNU/Linux and a collection of standard tools required for audio and networking, supplied in the WeMUST OS image. Most of the software is written in Bash and Python scripting languages for ease of use, distribution and update. Jacktrip, an open source C++ application [61] is supplied for the audio data transmission.

4.3 Software

4.3.1 Operating System

The chosen Linux distribution is Debian, a popular distribution in both embedded devices and personal computers, ported to a number of microarchitectures. As one of the earliest distributions to date it is well established as a stable and lightweight one, featuring three development stages, named unstable, testing and stable. "Debian stable" (currently codenamed *Wheezy*) comes from a freeze of the testing codebase and undergoes several months of debug and fixes. It has been chosen for its greater stability. In the Debian jargon, two ARM portings exist, named *armel* and *armhf*. The former can run almost on any ARM, while the second runs only on those featuring a floating point instruction set. Being the DM3730 floating-point-enabled, the armhf has been preferred to speed up floating point calculations, useful for audio data conversion and processing. The kernel chosen for the current version of WeMUST OS is 3.14.5-armv7-x8.

The current WeMUST OS image draws from a minimal Debian installation image, devoid from almost any user application software. A set of useful tools have been added from the official repositories, or after compilation from sources:

- development tools (compiler, automake, etc.),
- ALSA utilities and headers,
- JACK (Jack Audio Connection Kit) binaries and headers,
- Python 2.7 and libraries,
- Jacktrip (later detailed),
- networking tools (Avahi, etc.) and other utilities.

This allows the image and required disk space required to keep low, while allowing for all the functions expected to be implemented in the WeMUST framework.

One of the most relevant software stacks is that related to audio. In general, the sound card A/D gathers samples at a certain rate imposed by the clock and generates an interrupt every time a *period* (power of 2-sized chunk of the audio buffer) is ready to be passed to the software. The interrupt is not serviced instantaneously by the operating system, but may be deferred depending on its priority. Furthermore, the function calls triggered by the interrupt service routing, through a number of software layers take some time to reach obtain a timestamp from the system clock, to be used for audio management purposes. This timestamp, linked to the first sample of the audio collected from the hardware, is delayed with respect to the time the sample was obtained by the A/D, and this delay suffers some jitter, imposed by the aforementioned factors. For this reason, JACK implements a means to estimate the period time (from which the actual sample rate can be calculated), the beginning of the current period and the beginning of the next one. This is done in software by a digital phase locked loop, i.e. a DLL, a control loop, described in [62]. Filtering the system time by means of a DLL allows for a smooth and monotonous time function $T(t)$. From the DLL, an adaptive estimate of the clock frequency can be obtained.

The WeMUST system is currently based on ALSA and JACK[2]. ALSA (Advanced Linux Sound Architecture) is the currently maintained audio driver and API for low-level audio on Linux, replacing the legacy driver OSS. Jack is an open-source sound server available for PCs and ARM devices and handles low-latency audio and MIDI under many operating systems including UNIX and Linux. It enables interconnecting different applications, sources and sinks. JACK is a set of APIs and an audio server. It enables real-time processing within its active clients, i.e. real-time threads from running applications. A userspace application can register one its functions to be executed at each JACK cycle (i.e. every period time) via the following callback.

```
int jack_set_process_callback (jack_client_t *client,
                               JackProcessCallback
                               process_callback,
                               void *arg)
```

[2]jackaudio.org

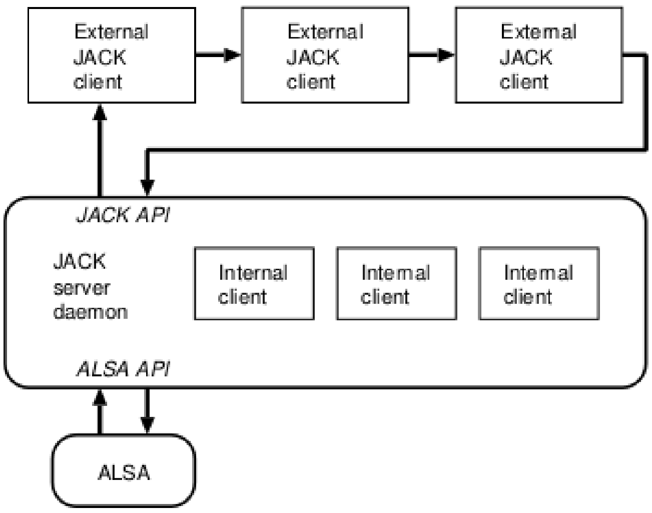


Figure 4.4: Internal and external clients in JACK.

Listing 4.1: JACK process callback method

When the audio engine starts, the `process_callback` function is executed at each period time and the internal time function $T(t)$ is started. All JACK clients are executed in a serial fashion and their input and output ports can be daisy-chained, thus they all must complete processing in the current time-frame in order avoid audio dropouts. This is called synchronous mode in Jack. Clients can be internal (those strictly related to JACK inherent functions, e.g. audio capture and playback) or external, ase seen in Figure 4.4.

Audio samples are handled in arrays, called *periods*. Each period consists of an interleaved sequence of left/right samples (in the case of stereo audio), called *frames*. Period sizes can range from 32 to 1024 and above frames. Shorter periods allow for reduced latency, but they increase the computational overhead as the number of interrupts increase. The latency is also related to the buffer size, i.e. the number of periods that are allocated for exchange of audio data to and from the audio card. A minimum of two periods must be allocated in the buffer, in order to perform a *ping-pong* strategy, since to concurrent read or write can be done: while the hardware writes a buffer, the other one is passed to the software for reading, and viceversa. Figure 4.5 clarifies the concept related to buffering. Note that two periods are always in use by the software or the hardware for passing audio data. By allocating more periods in the buffer the process of exchanging audio to and from the hardware is safer, however extra latency is added. The input-output latency is

$$D_{i/o} = 2 \cdot (N) \times \frac{P}{F_s}, \tag{4.1}$$

where N the number of buffers (min:2 for the ping-pong case), P the period size - normally of 2^i samples. To this, the delay introduced by any algorithms (zero only if audio is sent back to the output or no components with memory are employed in the processing) must be added. Interrupts from the audio card signal the presence of a new period to read and require a new period to play. For each period, JACK reads the input period and writes back an output period with audio to play back. The reading, processing and writing back must be done in the time between two audio card interrupts, i.e. a *period time*. In order to make audio work properly, JACK is called whenever an interrupt from the audio card is generated and allows its clients to be scheduled by the operating system. JACK real-time thread is scheduled as a real-time process by the `SCHED_FIFO` queue, to preempt other userspace processes that do not have strict timing requirements. Jack clients run a real-time DSP thread under `SCHED_FIFO` and other non real-time threads under `SCHED_OTHER` scheduler. Every Jack client RT thread gets a priority lower than that of Jack but higher than most userspace processes, thus, Jack can preempt clients that do not release the CPU when the deadline is reached (new interrupt). Every time this happens a dropout can be noticed, due to JACK giving back to the hardware a period not containing new data. The scheduling mechanism is reported in Figure 4.6.

The software stack dealing with audio is reported in Figure 4.7. Figure 4.8 reports all the components of ALSA and OSS (Open Sound Server), the legacy

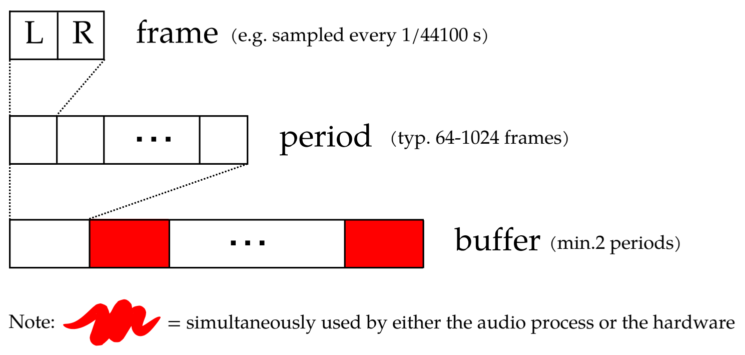

L
R
frame (e.g. sampled every 1/44100 s)
...
period (typ. 64-1024 frames)
...
buffer (min.2 periods)
Note:
= simultaneously used by either the audio process or the hardware

Figure 4.5: Naming convention used in the Linux audio community.

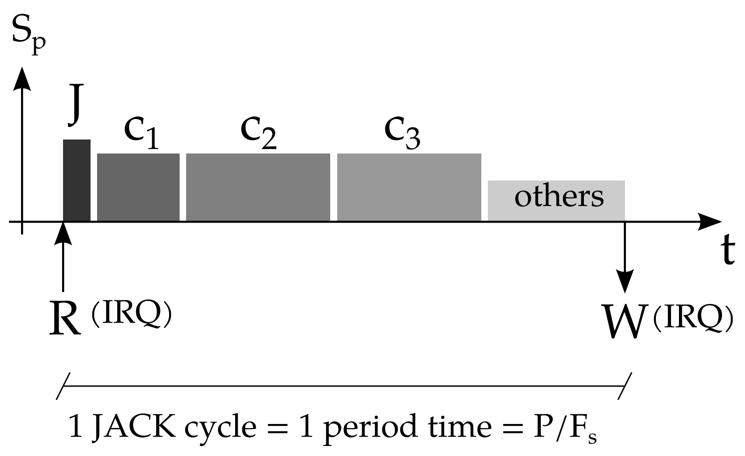


Figure 4.6: Audio processes with scheduling priority S_p scheduled on a single-thread processor. The JACK audio routine (J) is executed first, due its higher priority, at the interrupt of the audio card. Three JACK clients (c1, c2, c3) follow. Their scheduling order is imposed by JACK according the dependencies imposed by their routing. For the sake of simplicity only the read from the audio card (R) and the write (W) are shown.

sound server on Linux systems.

Further details on the audio software stack are given. ALSA is a kernel module that manages the hardware, and provides the buffering mechanism for audio input and output. It only allows one client to take exclusive control of the hardware. In this case JACK takes control of the hardware, allowing a number of audio clients to exchange audio with the hardware and among themselves. All the clients are synchronized to the audio codec interrupts. The interrupts depend on the period size. As reported above, the shorter the period size, the more frequent the interrupts, hence the more the overhead. In our case the shortest period size is 64 4-channels frames. When JACK is started all the audio parameters (bit-depth, sample rate, period size, etc.) must be specified. These parameters cannot change dynamically, and all the audio clients must work at the specified samplerate and period size (although internally they may resample). No resampling is performed inside JACK itself, differently from other audio servers, such as Pulseaudio. This ensures synchronization, high quality and reduced overhead.

4.3.2 Networking and Debugging

The JACK audio server already supports remote audio streaming employing one of the following modules: *netjack1*, *netjack2*. In alternative the jacktrip [63] and zita-njbridge clients exist. Each one of these has its own pro and cons, making it desirable for certain features but inadequate for some uses. For instance, netjack1 supports uncompressed audio streams as well as CELT-compressed[3] ones. Typical CELT coding delay is very low, but stands between 5 to 22.5 ms making its use undesirable for the application at hand. Table 4.1 highlights the main features for each one.

Desired features for the WeMUST platform are the availability of a device discovery mechanism, and multiple peer-2-peer links. The master-slave link, employed by netjack1 and netjack2, is not as flexible as it would be desired for the WeMUST platform. Synchrony between two devices in master-slave

[3]CELT stands for "Constrained Energy Lapped Transform" and is a low-delay audio codec[41]. CELT is obsolete, replaced by Opus: www.opus-codec.org.

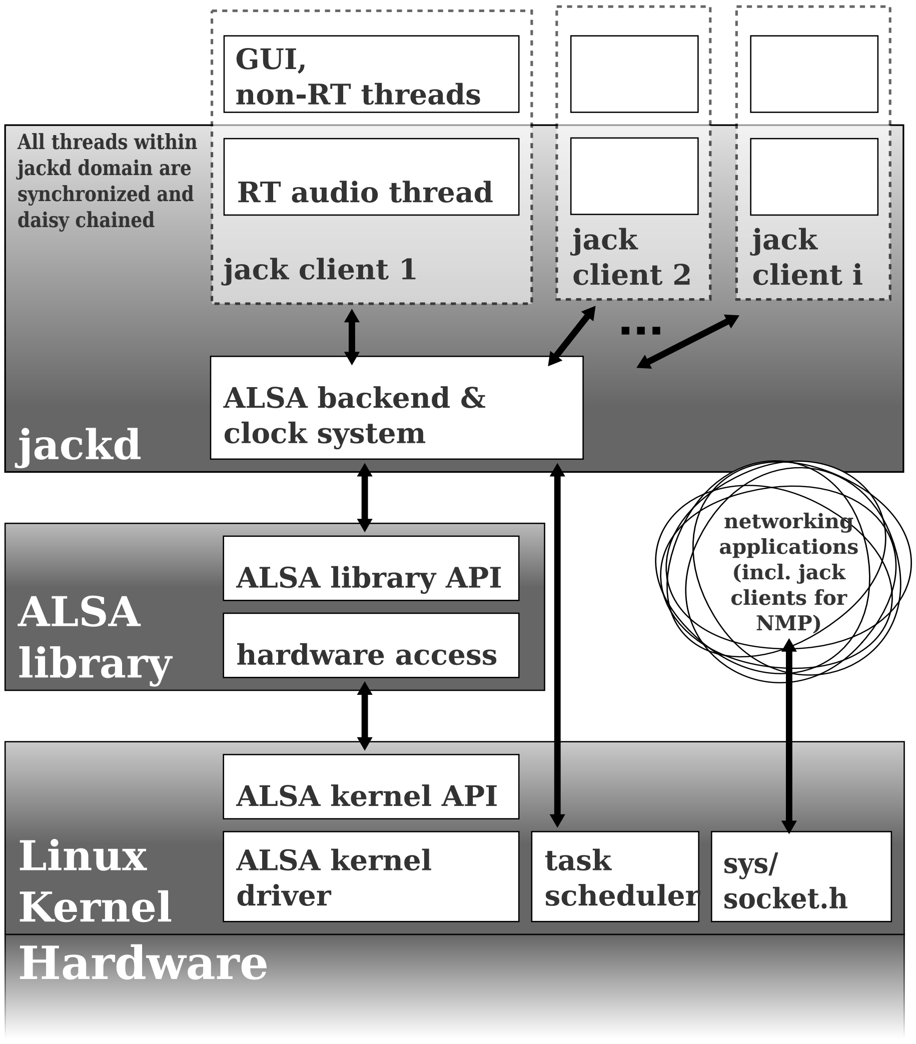


Figure 4.7: Overview of the audio software stack with a generic Linux use case. JACK is employed to run several audio threads.

	compression	link	auto discovery	synchrony
netjack1	CELT or no compress.	master-slave	none	yes
netjack2	uncompressed	master-slave	yes	yes
jacktrip	uncompressed	peer-2-peer	none	none

Table 4.1: Comparison of different audio networking applications in JACK.

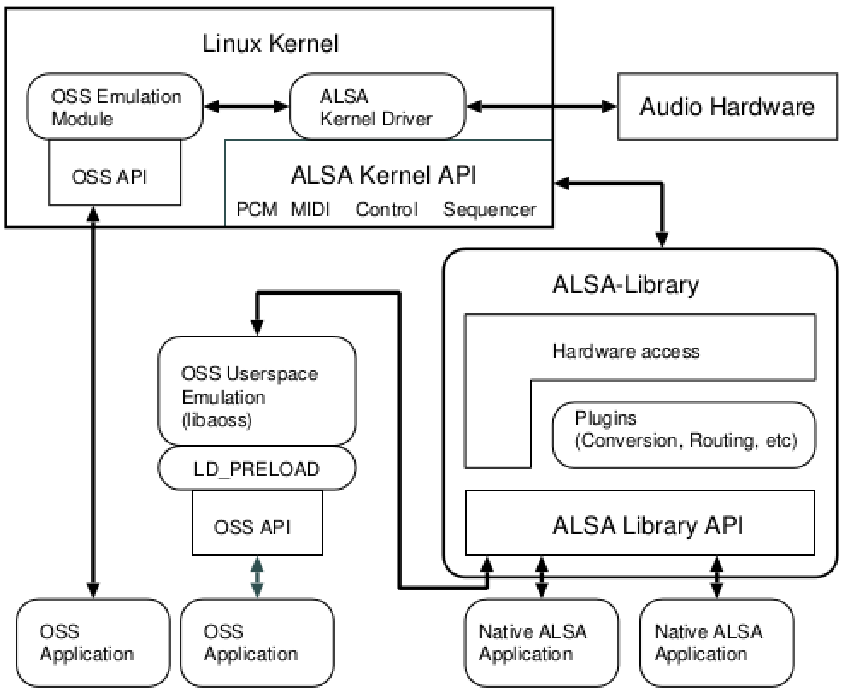


Figure 4.8: Overview of the different software components of ALSA in a Linux system.

mode is guaranteed by disconnecting the slave end from its audio card. This allows for processing on the slave end but no local sound card playback nor retransmission to another device. In order to do so, audio must be resampled by an additional module called `audioadapter`. Setup of such a master-slave link requires several minutes, a keyboard, a terminal window and an expert computer user. The WeMUST platform points toward a connection system based on automatic device discovery which automatically lists sources or sinks compatible with the current device, allowing selection of connections from a graphical display.

Jacktrip has been considered in this work. It is released as an open-source software[4] and has been improved during the development of WeMUST [64], branching the original source code of the v1.1.0 release. Either the Debian repository version (Jacktrip v1.0.5) and the modified one have been tested and employed in the WeMUST project. The v1.1.0 is more CPU-intensive than v1.0.5, mainly because of the v1.1.0 architecture reworking. When all the devices in the network share the same audio parameters, v1.0.5 is viable, while the modified code is needed for heterogeneous settings due to the addition of resampling. Jacktrip is employed to acquire audio from the hardware and send it to a remote client through the network. It also allows to receive audio from the same client. This could be a processed version of the original signal, or a monitor mix of several sources as in the *Waterfront* performance.

At the time of writing, a new open-source client has been released, called *zita-njbridge*, which has been briefly tested and proved efficient. However it was not available at the time the majority of the development was being conducted. In Section 6.1 it is discussed for future addition to WeMUST.

A set of Python scripts have been developed to allow easy audio transmission, debug and control. The scripts running on the BBxM are:

- jackaudio
- wemust-daemon
- read-usrbutton

[4]https://code.google.com/p/jacktrip/

At boot time the latter two are started by an `init` script. This script is started when networking services, the frequency scaling service and all the other required functionalities have been started.

On a supervisor machine other Python tools are required: wemust-netdbg and wemust-connect. Both the BBxM and the PC scripts are based on a Python library of function, wemust.py.

Figure 4.9 reports a simplified diagram that clarifies interactions and functionalities of these scripts. When all the required system services are loaded, an `init` script is started that loads the wemust-daemon and the read-usrbutton daemon. The latter controls the user button available on the BBxM, exposed as a `sysfs` item. Two modes are available: the button can act as a reset button (each time it is pressed jackaudio is stopped and restarted), or as a toggle to start and stop jackaudio. Wemust-daemon takes care of two parallel threads that will run until the system stops. The first, `getTempThread` periodically reads the CPU core temperature and sends it to the supervisor PC. The second, `ctrlThread` waits for commands from the supervisor PC, and executes them, sending back to the supervisor any command output. This is used to control the BBxM remotely or request for status updates (e.g. running processes, or kernel logs).

When either the user button is pressed, or the supervisor PC issues a "Start" command, jackaudio is started, which will set the frequency scaling to the maximum core frequency, start the JACK server, launch Jacktrip and connect it to the input and output. All the options to JACK, Jacktrip and the connections can be set by editing the fields at the top of the script or passing command line options (as done by wemust-connect).

On the supervisor PC, the scripts present a functional GUI, as reported in Figure 4.10. The problem of presenting the supervisor PC user with log messages coming from multiple IPs and multiple applications requires a markup strategy. Multiple consoles could be created, one for each peer. However, in this case the higher the number of peers, the higher the number of consoles, with the risk of crowding the PC screen. A functional distinction has been made instead, allocating one console per function, i.e.: general logging information,

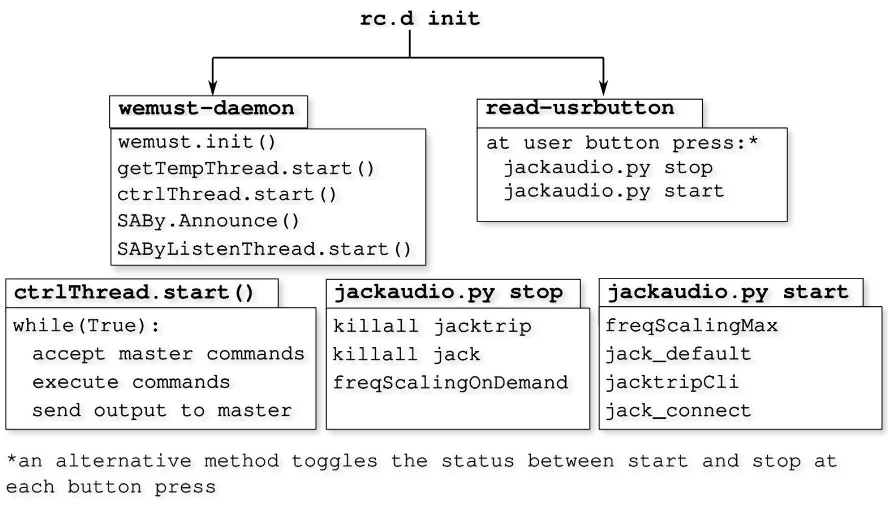
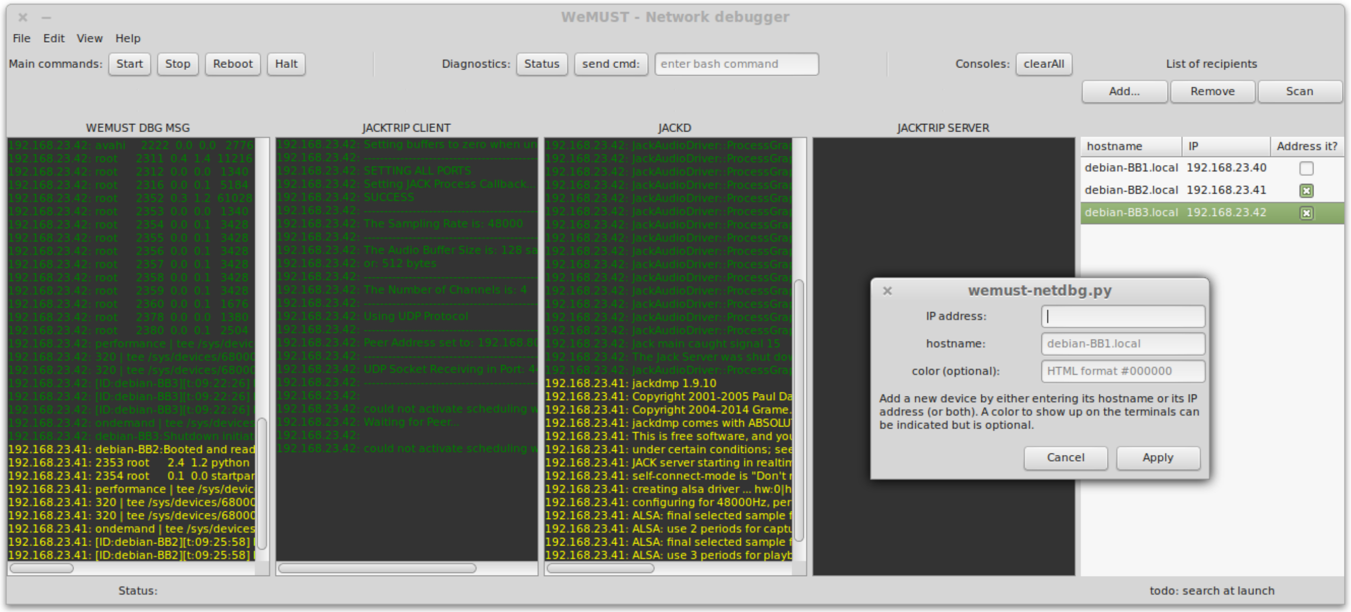


Figure 4.9: A diagram showing functional sections of the scripts running on WeMUST OS after boot.

messages from JACK and Jacktrip. Stacking these consoles horizontally allows to follow easily the time flow from top to bottom. Different colors are assigned to different peers, in order to increase readability.



```
192.168.23.42: avahi    2222  0.0  0.0   2776
192.168.23.42: root     2311  0.4  1.4  11216
192.168.23.42: root     2312  0.0  0.0   1340
192.168.23.42: root     2316  0.0  0.1   5184
192.168.23.42: root     2352  0.3  1.2  61028
192.168.23.42: root     2353  0.0  0.0   1340
192.168.23.42: root     2354  0.0  0.1   3428
192.168.23.42: root     2355  0.0  0.1   3428
192.168.23.42: root     2356  0.0  0.1   3428
192.168.23.42: root     2357  0.0  0.1   3428
192.168.23.42: root     2358  0.0  0.1   3428
192.168.23.42: root     2359  0.0  0.1   3428
192.168.23.42: root     2360  0.0  0.1   1676
192.168.23.42: root     2378  0.0  0.0   1380
192.168.23.42: root     2380  0.0  0.1   2504
192.168.23.42: performance | tee /sys/devic
192.168.23.42: 320 | tee /sys/devices/68000
192.168.23.42: 320 | tee /sys/devices/68000
192.168.23.42: [ID:debian-BB3][t:09:22:26]
192.168.23.42: [ID:debian-BB3][t:09:22:26]
192.168.23.42: [ID:debian-BB3][t:09:22:26]
192.168.23.42: ondemand | tee /sys/devices
192.168.23.42: debian-BB3:Shutdown initiat
192.168.23.41: debian-BB2:Booted and read
192.168.23.41: 2353 root     2.4  1.2 python
192.168.23.41: 2354 root     0.1  0.0 startpar
192.168.23.41: performance | tee /sys/devic
192.168.23.41: 320 | tee /sys/devices/68000
192.168.23.41: 320 | tee /sys/devices/68000
192.168.23.41: ondemand | tee /sys/devices
192.168.23.41: [ID:debian-BB2][t:09:25:58]
192.168.23.41: [ID:debian-BB2][t:09:25:58]
```

```
192.168.23.42: Setting buffers to zero when un
192.168.23.42: ----------------------------------------
192.168.23.42: SETTING ALL PORTS
192.168.23.42: Setting JACK Process Callback...
192.168.23.42: SUCCESS
192.168.23.42: ----------------------------------------
192.168.23.42: The Sampling Rate is: 48000
192.168.23.42: ----------------------------------------
192.168.23.42: The Audio Buffer Size is: 128 sa
192.168.23.42: or: 512 bytes
192.168.23.42: ----------------------------------------
192.168.23.42: The Number of Channels is: 4
192.168.23.42: ----------------------------------------
192.168.23.42: Using UDP Protocol
192.168.23.42: ----------------------------------------
192.168.23.42: Peer Address set to: 192.168.80
192.168.23.42: ----------------------------------------
192.168.23.42: UDP Socket Receiving in Port: 4
192.168.23.42: ----------------------------------------
192.168.23.42:
192.168.23.42: could not activate scheduling w
192.168.23.42: Waiting for Peer...
192.168.23.42:
192.168.23.42: could not activate scheduling w
```

```
192.168.23.42: JackAudioDriver::ProcessGra
192.168.23.42: JackAudioDriver::ProcessGra
192.168.23.42: JackAudioDriver::ProcessGra
192.168.23.42: JackAudioDriver::ProcessGra
192.168.23.42: JackAudioDriver::ProcessGra
192.168.23.42: JackAudioDriver::ProcessGra
192.168.23.42: JackAudioDriver::ProcessGra
192.168.23.42: JackAudioDriver::ProcessGra
192.168.23.42: JackAudioDriver::ProcessGra
192.168.23.42: JackAudioDriver::ProcessGra
192.168.23.42: JackAudioDriver::ProcessGra
192.168.23.42: JackAudioDriver::ProcessGra
192.168.23.42: JackAudioDriver::ProcessGra
192.168.23.42: JackAudioDriver::ProcessGra
192.168.23.42: JackAudioDriver::ProcessGra
192.168.23.42: Jack main caught signal 15
192.168.23.42: The Jack Server was shut dow
192.168.23.42: JackAudioDriver::ProcessGra
192.168.23.41: jackdmp 1.9.10
192.168.23.41: Copyright 2001-2005 Paul Da
192.168.23.41: Copyright 2004-2014 Grame.
192.168.23.41: jackdmp comes with ABSOLU
192.168.23.41: This is free software, and you
192.168.23.41: under certain conditions; see
192.168.23.41: JACK server starting in realtim
192.168.23.41: self-connect-mode is "Don't
192.168.23.41: creating alsa driver ... hw:0|h
192.168.23.41: configuring for 48000Hz, per
192.168.23.41: ALSA: final selected sample f
192.168.23.41: ALSA: use 2 periods for captu
192.168.23.41: ALSA: final selected sample f
192.168.23.41: ALSA: use 3 periods for playb
```

hostname	IP	Address it?
debian-BB1.local	192.168.23.40	☐
debian-BB2.local	192.168.23.41	☒
debian-BB3.local	192.168.23.42	☒

wemust-netdbg.py

IP address:

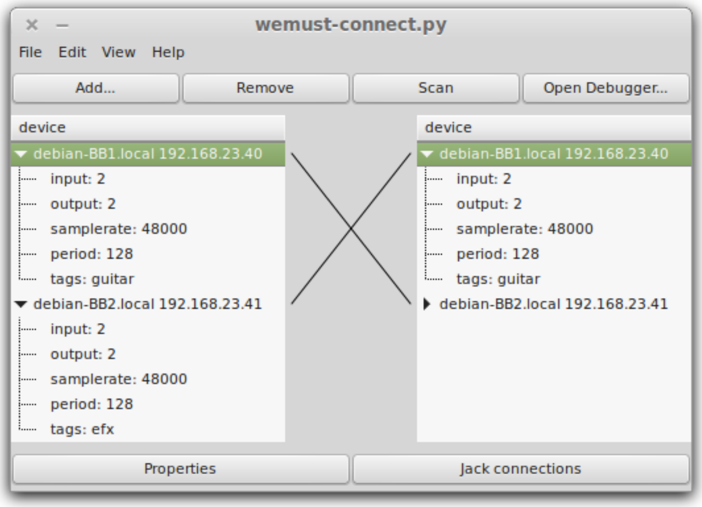
hostname: debian-BB1.local

color (optional): HTML format #000000

Add a new device by either entering its hostname or its IP address (or both). A color to show up on the terminals can be indicated but is optional.

Cancel Apply

Figure 4.10: wemust-netdbg GUI.



```
device
▼ debian-BB1.local 192.168.23.40
    input: 2
    output: 2
    samplerate: 48000
    period: 128
    tags: guitar
▼ debian-BB2.local 192.168.23.41
    input: 2
    output: 2
    samplerate: 48000
    period: 128
    tags: efx
```

```
device
▼ debian-BB1.local 192.168.23.40
    input: 2
    output: 2
    samplerate: 48000
    period: 128
    tags: guitar
▶ debian-BB2.local 192.168.23.41
```

Figure 4.11: wemust-connect GUI.

The current GUI for wemust-connect, depicted in Figure 4.11, takes inspiration from the connection utilities in QJackCtl[5], a graphic frontend for JACK. The graphic metaphores are the same, however, in this case, peers have ports that are connected with other peers. The connection takes place by sending appropriate commands to the peers.

The WeMUST-OS image for BBxM and all the software tools are available at the A3LAB research group web page[6].

4.4 Hardware

The BBxM is an open hardware platform, designed and promoted by the Beagleboard.org Foundation. It features a 1GHz ARM Cortex-A8 core (Texas Instruments DM 3730), 512MB RAM and many peripherals generally found on personal computers or mini-PCs. One important aspect is the availability of input and output line sockets, and a capable audio codec IC (Texas Instruments TPS 95650). The input analog stage is lacking a preamplifier to adapt

[5]http://qjackctl.sourceforge.net/
[6]http://a3lab.dii.univpm.it/research/wemust

to microphone signal levels.

More recent and widespread platforms, such as the Beagleboard Black by the same team, or the Raspberry Pi, are similar in the core architecture, but are targeted to different application, and either lack the onboard audio codec or lack the audio input. For this reason, some works addressed this need by adding an external audio codec or USB sound card[65, 66]. In WeMUST, portability and ease of connection is a key goal, hence the BBxM, with its on-board audio connections and codec has been preferred over other platforms.

Key components available on the platform are the Ethernet and USB controllers, the audio codec, the RS232 serial interface for debugging and the microSD slot which stores the bootloader and the operating system. The availability of the Ethernet controller is fundamental in prototyping stages, to access the system reliably via a remote command console. The USB connection is necessary to add WiFi capabilities or to attach commodity devices.

4.4.1 Power Management

Multimedia applications can often be power-constrained. Power management is required for battery-powered applications, and there are use cases, such as that described in Sec. 4.8 which require to run on battery. To avoid audio glitches and achieve maximum performance the DM3730 core must not resort to dynamic frequency scaling and for best performance it should be clocked at the highest frequency, i.e. 1GHz during audio packets exchange or processing. However, while waiting for a connection, the core can be clocked at a lower frequency. This functionality, called frequency scaling, is implemented in the `cpufreq_governor` Linux kernel module, generally referred to as *scaling governor*, and is easily accessible from the `sysfs` virtual file system interface. The scaling governor enables different operating modes, such as *ondemand*, *performance*, *powersave*. While the former dynamically scales frequency according to a load balancing algorithm, the latter two enable maximum and minimum core frequency. By taking advantage of the `sysfs` interface WeMUST-tools scripts enable to save power while not transmitting audio. As musical performances

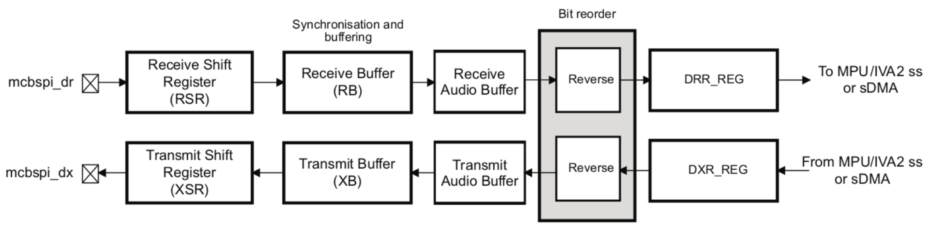


Figure 4.12: Diagram of the McBSP2 audio data path.

need to be set up some time prior to the due time (even hours before), saving power while waiting for the performance to start enables to increase battery life.

4.4.2 On-board Audio

While other embedded audio projects employ external USB audio interfaces [67] or custom tailored I2S (Inter-IC Sound) audio codecs [66], the authors purportedly decided to take advantage of the on-board audio codec, pushing the system to its full potential. As reported in previous works [64], the TPS95650 is a companion chip supporting operation of OMAP3 and other OMAP3-compatible ICs (such as the DM3730). It provides an audio codec (for high-quality multimedia streams) and a voice codec (for lower quality, low latency voice communication). The audio codec is connected to one of the DM3730 serial lines, McBSP2 (Multi-channel Buffered Serial Port). The McBSP2 (see Figure 4.12 has a large buffer, to avoid glitches in multimedia applications. At the current state, the ALSA[7] audio driver driver for TPS95650 , needs to fill the entire McBSP2 output buffer and thus, allocate several buffers. On the contrary, the input McBSP2 buffer can allocate as few as two buffers of the desired length. In [64] a solution was found to decrease the input-output latency to 9.3 ms by halving the hardware buffer queue by introducing dummy data without affecting operation. As a remark, lower latencies could be obtained with a different audio chipset allowing for shorter hardware buffering with the same period size.

The choice of the buffer size determines the computational resources devoted

[7]ALSA stands for Advanced Linux Sound Architecture. It is the Linux kernel module taking care of low-level audio function and hardware interfaces.

to buffer exchange, interrupts servicing, etc. The lower the period size, the higher the overhead. With the ARM core running at maximum speed, and the software stack later described in Sec. 4.3.1, the minimum buffer size that do not incur into glitches is 64 frames. Lowering the period size to 32 incur into audible glitches and platform instabilities even when there is no processing going on.

One important lack of the TPS95650 is the absence of a microphone preamp, forcing the user to use an external preamplifier. 9V battery-powered ones are a available for smartphones and similar devices which provide +48 V *phantom* power, accept XLR balanced inputs and adapt it to the 3.5 mm jack input of the BBxM.

4.4.3 A/D and D/A Latency and CPU Overhead

As mentioned previously, the input-output latency of the BBxM can be as low as 9.3 ms without affecting the audio quality with audible glitches. This is given by the need for 2 64-frames audio input buffers and 5 64-frames audio output buffers at 48kHz. However, when transmitting audio to other peers, some ring buffering is implemented inside Jacktrip, to avoid network jitter and allow packet reordering. This buffering is necessary to avoid glitches due to network issues, but adds some latency. Ring buffer size can be decided by the user. With good networking condition (wired, or a good wireless link), four buffers can be allocated, with an average delay of two buffers (the buffer pointer tries to keep in the middle of the ring buffer to avoid both overruns and underruns). Tests with two BBxM acting as peers, with the signal following a round trip from one to the other and back via Jacktrip, the average input-output latency is approximately 15.7 ms, with 6.4 ms given by the ring buffer (two receive ring buffer, one for each trip) and the remainder 9.3 ms by the BBxM hardware buffering. No resampling is performed and both BBxM are set to work at 48kHz.

Another important aspect is the CPU load of such a configuration. The shorter the period size, the shorter the time to perform all operations (inter-

		32kHz	44.1kHz	48kHz
128	jack	13%	19%	21%
	jtrip	16%	21%	23%
64	jack	21%	30%	46%
	jtrip	22%	33%	46%
32	jack	89%	78%	94%
	jtrip	-	-	-

Table 4.2: Average CPU load evaluated for different sample rates and period sizes. Please note that with a period size of 32 frames, Jacktrip has not been run since JACK already required most of the CPU for synchronizing to the audio card interrupts.

rupts servicing, buffer handling, data encapsulation in 802.11 or 802.3 frames, just to name a few). This results in a higher overhead and a higher average CPU load. Table 4.2 reports some average CPU load data depending on the audio parameters for a system running JACK, sending and receiving audio packets through Jacktrip. These results have been measured in best-case conditions and are subject to possible issues, such as CPU heating (depending on the housing or the outside climate conditions), link reliability, and in rare cases possible wear of the SD card hosting the whole operating system due to the high load the system is subject to. For a more reliable performance, more conservative options can be chosen. Please note that the upper bound for glitch-less audio operation is well below the 100% CPU load threshold, due to the non-optimal nature of the Linux scheduling algorithms (in terms of deadline constraining) and the presence of kernel preemption and hardware interrupts. Kernel network layer critical sections and network card interrupts servicing, that are important for audio packet transmission can affect performance. In the current case, for instance, the interrupt routine for the wireless or wired LAN controller, may steal time to the audio process interrupt routine managing the TPS95650. Thriving more into these details is out of the scope of this paper.

4.4.4 Wireless Interface

As stated above, the goal of WeMUST is assess the feasibility of low-latency audio transmission with commercial wireless technologies. Several sub-1GHz ISM bands are available and inexpensive transceivers are marketed since years, but can hardly provide enough bandwidth for high quality audio transmission. On the other hand larger bandwidth channels are available for multimedia local area networks, e.g. in the 60GHz band, but they are not yet widespread. The 802.11 family is the communication technology adopted for the WeMUST project, being the only widespread high-bandwidth wireless technology available at the time of writing. WiFi RF chips are nowadays embedded into almost any computational device. For this reason the authors focused their effort on the 802.11 family of protocols, however, at the time of writing, in the absence of reliable 802.11ac device drivers available for the Linux kernel, tests have been conducted only on 802.11a/b/g and 802.11n links. Regarding kernel driver availability, WeMUST-OS is based on a recent kernel supporting several wireless chipsets[8]. Among these, the RT5370 (driver: rt2800usb) and RTL8191SU (driver: rtl8712u) chipsets have been tested and work correctly on the BBxM. In a previous work [68] it is shown that 802.11g networks are able to sustain basic audio transfer: a bidirectional uncompressed 48kHz stereo couple does not incur in sensible packet loss. The increased data rate available in a 802.11n link reduces the loss probability to a very low value.

While driving a on-board or USB-connected wireless chipset is the most straightforward solution to connectivity within WeMUST, wired Ethernet connection facilitated the development and may be considered for alternative solutions.

[8]http://wireless.kernel.org/

4.5 Automating Connections: a Device Discovery Protocol

In the path to a seamless audio connectivity framework some effort is required in order to provide meta-data regarding the devices and the streams and allow software to automatically connect or list available devices. The UPnP device discovery protocol is taken as a reference for a protocol draft, implemented in WeMUST.

4.5.1 UPnP Device Discovery

A number of device discovery protocols already exist in the literature and in the market. One of the most used in multimedia wireless streaming is the Simple Service Discovery Protocol (SSDP), used by the UPnP Forum[9], on which DLNA-compliant products are based. The UPnP set of protocols is implemented for many multimedia devices, such as smart TVs, smartphones, PCs, NAS (Network Attached Storage) devices, etc [10]. Both open and proprietary implementations exist, ported to many platforms. The UPnP standard is very complex and targeted to a large number of applications: from video streaming to home audio distribution, from network printer access to multimedia content browsing. Audio streaming in UPnP is handled as a reliable compressed stream, with large buffering and thus, high latency. As such, it is not targeted to music production or performance use. The SSDP layer however is of interest for several reasons: based on IP standards, it provides a server-less discovery mechanism. SSDP is a HTTPU text-based protocol encapsulated in UDP datagrams which are sent to a multicast address for discovery or to unicast addresses for information exchange. The SSDP and upper UPnP layers guarantee devices not only to discover other UPnP devices but also to describe their set of features and resources and have access to them.

The SSDP device discovery mechanism includes three types of messages: ad-

[9]www.upnp.org

[10]UPnP Forum "UPnP Specifications v.1.1", available online at: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf

vertise (uses the `NOTIFY * HTTP/1.1` header), search (uses the `M-SEARCH * HTTP/1.1` header) and search response (uses the `HTTP/1.1 200 OK` header). The advertise message is issued by a root device for different purposes: announce a new root device connection or reaffirm its presence (Notification Sub Type: `ssdp:alive`), several NOTIFY messages must be sent for each resource or service), change or update information about the set of features and services available (Notification Sub Type: `ssdp:update`), announce the removal of a resource or service from the network (Notification Sub Type: `ssdp:byebye`).

The general connection scheme implies announces for a root device when it first connects to the network, one for each of its embedded devices or services. Control points can issue a search to the entire network or a unicast address in order to receive further information on its capabilities. This sort of information is described in XML format. The device description part of the UPnP is outside the scope of the device discovery and will not described further.

4.5.2 Simple Autonomous Buddying Protocol Specifications

The SSDP is a server-less protocol that provides device discovery and search preliminary to subsequent connection of devices. The protocol is quite flexible and at the moment the authors considered it even too complex for the application at hand. In alternative to SSDP the authors propose a simpler solution, partly inspired by SSDP, called Simple Autonomous Buddying (SABy). The current solution is implemented in the Python tools wemust-connect and wemust-netdbg and as a Puredata C external, for compatibility with other platforms. The Puredata external implementing SABy has been called `[netfind]` and works together with two modified versions of `[netsend~]` and `[netreceive~]` originally designed by Olaf Matthes[11]. The audio streaming is still handled by the latter two externals, while the device discovery is handled by `[netfind]`, which configures the two streaming externals for proper connection to and from a new peer, when found. In the Python implementation SABy-related classes and functions are available in the wemust.py library. However, the Puredata

[11]http://www.nullmedium.de/dev/netsend~/

implementation is less goal-specific and, thus, is taken here as example to detail the SABy specifications.

SABy is similar to SSDP, but stripped down for the application at hand. A multicast group (a multicast IPv4 address, i.e. 239.255.255.251:1991) is used, similarly to SSDP for announcement of new devices. `[netfind]` can instantiate four different child threads:

1. `netfind_listen`: listens to announces sent to the multicast group. When a valid announce is received instantiates a new `netfind_flowctrl` thread.
2. `netfind_announce`: announces the existence of the device to the multicast group at the instantiation of `[netfind]` and repeats the announce at a certain interval `ANNOUNCE_INTERVAL` of several seconds.
3. `netfind_flowctrl_listen`: waits for incoming TCP connections at the port `FLOWCTRL_TCP_PORT`. It can accept multiple connections. For each connection negotiates the audio parameters, ports for audio streaming, etc. If the peer `netfind_flowctrl` thread accepts the connection, they keep the TCP connection alive and they check on each other's state by requiring a status message and reply with a *ACK* message.
4. `netfind_flowctrl`: is started when `netfind_listen` finds a suitable peer to connect to. It tries to connect to that peer on its `FLOWCTRL_TCP_PORT` port and initiates negotiation with it. The negotiation includes a check for the match of the audio parameters (at the moment the sampling rate only) and then tries to agree with the peer for port numbers for incoming and outgoing audio streaming. If successful, it sends connection parameters to the `[netsend~]` and `[netreceive~]` externals connected to `[netfind]` outlets.

In Figure 4.13 the interaction of two devices is illustrated, with L and F being Peer1 `netfind_listen` and `netfind_flowctrl` child threads, while A and FL are Peer2 `netfind_announce` and `netfind_flowctrl_listen` child threads. Each peer have the identical set of threads, however for illustration purpose only the ones involved in a handshake are shown. For instance, Peer1 has its own `netfind_announce` and `netfind_listen` threads, but since the two peers appear at different instants, the first one to spot the other's presence is Peer1, which occurs to receive Peer2 multicast announce and starts the negotiation between the F and FL threads. On the other hand the Peer2 `netfind_listen` thread will spot the presence of Peer1 only after their connection, so that Peer2 `netfind_listen` will not start a new child thread for connection with Peer1. The `ANNOUNCE_INTERVAL` time must be high enough to statistically avoid two peers to simultaneously start a connection negotiation but must be low enough for the user experience to be responsive enough. For a simultaneous connection negotiation to happen from both sides the two devices must send their announcement messages in a time interval shorter than the connection negotiation duration (time from the creation of thread F1 to success). This time interval generally takes from 5 to 6 ms. This condition thus can only happen when two devices are both started in this short time interval. The simultaneous connection negotiation can thus avoided by using on each peer a register accessible from all its child threads to track peers with whom connection is in negotiation. On the other hand, to improve the connection responsiveness a search can be issued by `[netfind]` at start instead of waiting for announcements.

The announce message is a text packet of the form shown in the following.

```
APP:<application name>
SR:<samplerate|period size>
NID:<ID>
TAG:<optional>
```

Listing 4.2: Announce message format

The first three fields are mandatory. The `APP:` field stores the application (e.g. puredata) that implements the protocol. The `NID` field can be a user-provided name or a unique 10-characters alphanumeric ID assigned by the system at the instantiation of `[netfind]`. The `SR:` field stores audio paramenters, while the `TAG:` field is optional and can store keywords describing the device, that can be searched for. Multiple connections can be handled and

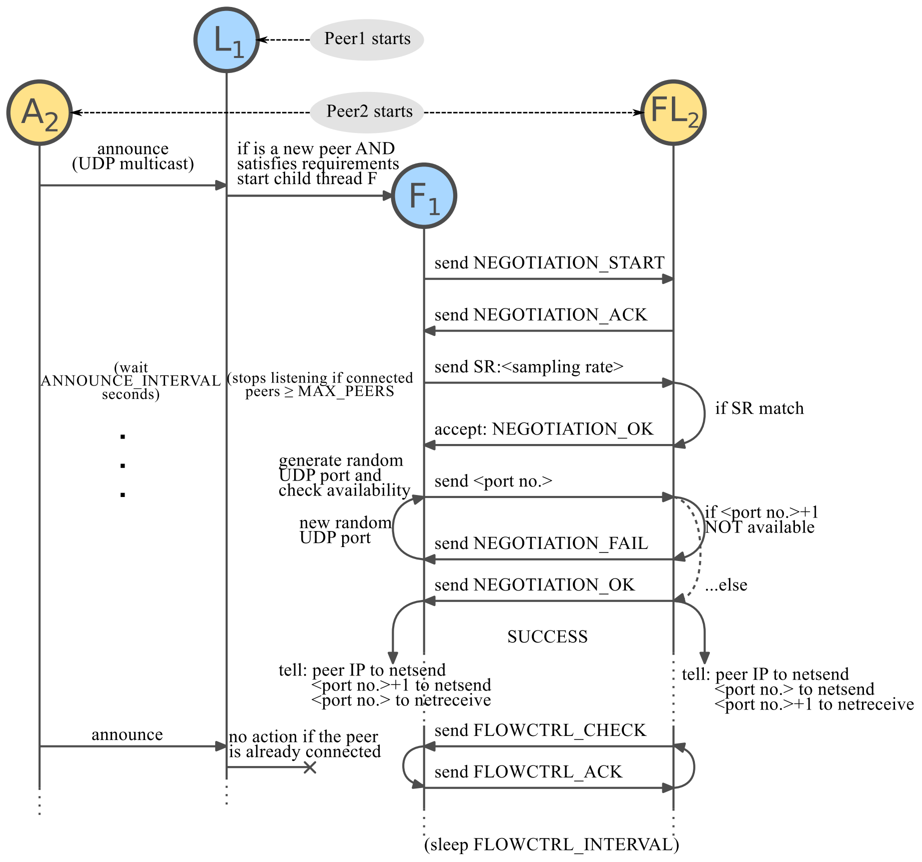

L1
Peer1 starts
A2
Peer2 starts
FL2
announce (UDP multicast)
if is a new peer AND satisfies requirements start child thread F
F1
send NEGOTIATION_START
send NEGOTIATION_ACK
(wait ANNOUNCE_INTERVAL seconds)
(stops listening if connected peers ≥ MAX_PEERS
send SR:<sampling rate>
if SR match
accept: NEGOTIATION_OK
generate random UDP port and check availability
send <port no.>
if <port no.>+1 NOT available
new random UDP port
send NEGOTIATION_FAIL
send NEGOTIATION_OK
...else
SUCCESS
tell: peer IP to netsend <port no.>+1 to netsend <port no.> to netreceive
tell: peer IP to netsend <port no.> to netsend <port no.>+1 to netreceive
announce
no action if the peer is already connected
send FLOWCTRL_CHECK
send FLOWCTRL_ACK
(sleep FLOWCTRL_INTERVAL)

Figure 4.13: Flow chart of the simple device discovery protocol between Peer1 and Peer2 child threads. L stands for Peer1 `netfind_listen` thread, A and FL stand for Peer2 `netfind_announce` and `netfind_flowctrl_listen` threads, F stands for Peer1 `netfind_flowctrl` thread.

initiated this way very easily. It is up to the user to prepare the Pure Data patch in order to handle audio from the peer platforms. An example patch is provided together with the externals source code[12] that is able to control several [netsend~] and [netreceive~] externals, by way of multiplexing netfind messages. In Figure 4.14 a 2-way communication implementation based on a single Pure Data patch running on two different machines is illustrated for clarity.

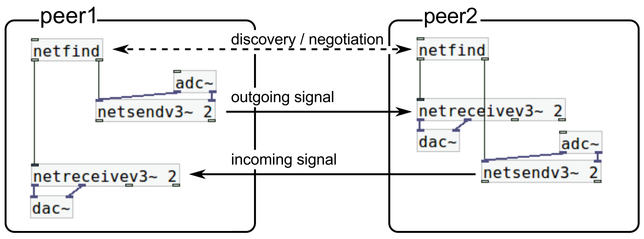


Figure 4.14: A 2-way communication implemented in Pure Data using [netfind].

Selection of peers to connect with can be also done based on a list of capabilities. When a `search <tag>` message is issued to [netfind], it performs a query to the multicast group for devices that are tagged accordingly. From that moment on it will also discard announce messages that do not contain the specified tag. Tags are added to a [netfind] external by issuing a `addtags <tag list>` message. The tagging mechanism allows for quick selection and connection of available devices. For instance, a guitar stompbox based on Pure Data patches and [netfind] can be configured to only connect the input [netreceive~] external to devices tagged as "guitar", and the output [netsend~] external to a device tagged as "mixer" without need of human intervention.

4.5.3 Comparison with SSDP

The SABy mechanism described hereby inherits some concepts from SSDP. It is however different in some respects. Two design principles have been taken

[12]http://a3lab.dii.univpm.it/research/wemust

into account: flexibility mediated by small computational footprint. A complete UPnP software stack could be employed and adapted to the application at hand and development could be based on existing open C/C++ libraries. However this would create a high dependency on rather large libraries, decreasing portability, which results unnecessary and would create large compiled binaries for such a simple task. This lightweight protocol includes all primary information in the announce message. Description of the devices is based on text tags and a simple search/query mechanism, instead of the much more complex UPnP Description layer, which in turn requires an additional XML parser library. Overall responsiveness is very high, as mentioned above and the source code is quite compact and self-contained.

4.6 Adaptive Resampling for Clock Synchronization

In Section 3.2.4 the clock synchronization issue was introduced, together with an adequate formalism. The communication technology literature provides several solutions depending on the use case constraints. As an example, the CS-MNS algorithm is provided in Section 3.2.4. A different approach has been implemented in jacktrip as part of the WeMUST project. Differently from CS-MNS, which aims at a perfect synchronization. In NMP only frequency or relative synchronization is required. The time functions of two remote ends may have a different offset (i.e. JACK started at different instants), but the audio link between the two instances of the network audio software start when the first packet is sent from one of the two instances. Referring to the formalism of Section 3.2.4, let the transmitter and receiver skew be β_1 and β_2 respectively. If the two devices run at exactly the same frequency then $\beta_1 = \beta_2$, thus the offset is

$$\rho = \xi_1(t) - \xi_2(t) + T_1(0) - T_2(0). \tag{4.2}$$

As far as the random processes $\xi_1(t)$ and $\xi_2(t)$ are bounded the offset is bounded as well. As explained in Section 3.2.4, a bounded offset can be compensated for by buffering at the receiving end.

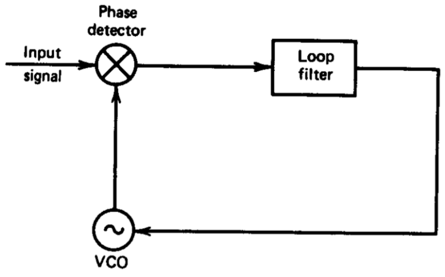


Figure 4.15: SFG of a DLL.

In general, unfortunately, the assumption $\beta_1 = \beta_2$ is not justified in reality. As a result the offset diverges with time. To avoid this from happening the two frequencies should be matched somehow. There is no access to the hardware clock of the two remote devices, thus the adjustment is done at the software level by means of resampling. Resampling has been introduced in jacktrip in two steps: a uniform resampling algorithm, allowing devices with different nominal sampling rates to exchange audio seamlessly, and adaptive resampling, allowing devices with different actual sampling rate to communicate without dropouts. By employing both, of course both advantages are achieved.

Adaptive resampling is performed by constantly adapting the resampling ratio to keep the same pace as the receiver. The data it requires to work is extracted from the JACK DLL, hereby introduced.

4.6.1 Delay-Locked Loop in JACK

The DLL in JACK is a digital implementation of a typical Phase-Locked Loop in analog electronics, i.e. a control loop employed to filter digital signals jitter and such. A PLL is composed of a phase detector, a loop filter and voltage control oscillator (VCO), see Figure 4.15.

Let $\theta_i\left(t\right)$ be the input signal phase and $\theta_o\left(t\right)$ the VCO output signal phase. The output of the phase detector is, thus,

$$v_d = K_d \cdot (\theta_i - \theta_o) \tag{4.3}$$

where K_d is the phase detector gain. Filtering the voltage v_d by the low-pass filter $F(s)$ removes noise and high-frequency components. The VCO frequency is determined by the filter output $v_c(t)$. The relation between the VCO input and its output frequency deviation from a central value is

$$\Delta\omega = K_o \cdot v_c \tag{4.4}$$

where K_o is the VCO gain. Given the differential relation between phase and frequency,

$$\frac{\partial\theta_o}{\partial t} = K_o \cdot v_c, \tag{4.5}$$

stands true. Its Laplace transform is

$$L\left[\frac{\partial\theta_o\left(t\right)}{\partial t}\right] = s \cdot \theta_o\left(s\right) = K_o \cdot V_c\left(s\right), \tag{4.6}$$

i.e. the phase of the VCO output is proportional to $\int v_c(t)dt$. By also transforming in the Laplace domain the following

$$V_d\left(s\right) = K_d \cdot \left[\theta_i\left(s\right) - \theta_o\left(s\right)\right] \tag{4.7}$$

$$V_c\left(s\right) = F\left(s\right) \cdot V_d\left(s\right) \tag{4.8}$$

the filter output is

$$V_c\left(s\right) = \frac{s \cdot K_d \cdot F\left(s\right) \cdot \theta_i\left(s\right)}{s + K_o \cdot K_d \cdot F\left(s\right)} = \frac{s \cdot \theta_i\left(s\right)}{K_o} \cdot H\left(s\right) \tag{4.9}$$

where $H\left(s\right)$ is the closed-loop transfer function. By substituting the low-pass filter Laplace transfer function

$$F\left(s\right) = \frac{1 + \tau_2 \cdot s}{1 + \tau_1 \cdot s} \tag{4.10}$$

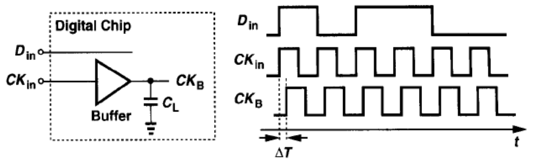


Figure 4.16: Delay introduced by buffering and tracks parasitic components in a IC require reconstructing the clock signal.

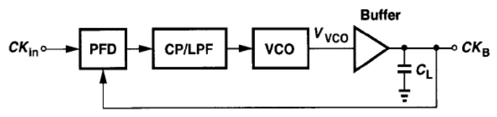


Figure 4.17: A PLL can be introduced to remove clock signal delay or jitter.

in 4.9, the closed loop transfer function can be rewritten as

$$H(s) = \frac{K_d \cdot F(s)}{\frac{s}{K_o} + K_d \cdot F(s)} = \frac{\frac{K_o \cdot K_d(\tau_2 \cdot s + 1)}{\tau_1}}{s^2 + s \cdot \left(\frac{K_o \cdot K_d \cdot \tau_2}{\tau_1}\right) + \frac{K_o \cdot K_d}{\tau_1}}. \tag{4.11}$$

The closed loop transfer function is thus a second order one, and can be simply rewritten as

$$H(s) = \frac{2 \cdot \xi \cdot \omega_n \cdot s + \omega_n^2}{s^2 + 2 \cdot \xi \cdot \omega_n \cdot s + \omega_n^2}. \tag{4.12}$$

The design of PLLs for clock skew reduction is well documented [69]. Its use is necessary to remove the delay introduced by different propagation path for the clock signal through copper tracks in ICs. In Figure 4.16 the clock signal CK_B at the input of a gate is delayed with respect from the input data D_{in} and the original clock CK_{in}. By introducing a PLL in the clock path the delay or jitter between CK_{in} and CK_b is removed by the feedback control, as shown in Figure 4.17.

Similarly, in the digital domain, a feedback filter can be employed to filter out jitter in timestamps and estimate the periodicity between those. Since the closed-loop transfer function of a PLL is a simple second-order recursive system, it can be implemented as a IIR, shown in Figure 4.18. Such a filter is employed in JACK [62] to estimate the current and next period timestamps

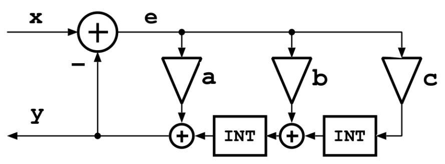


Figure 4.18: SFG of a second-order DLL.

as well as the period time. The filter coefficients are computed following the equations

$$\omega = \frac{2\pi B}{F_s} \tag{4.13}$$

$$a = 0 \tag{4.14}$$

$$b = \sqrt{2}\omega \tag{4.15}$$

$$c = \omega^2 \tag{4.16}$$

where F_s is the sampling frequency and B the desired bandwidth.

Although the acronym is spelled as Delay-Locked Loop [62], the literature suggests that DLL are implemented based on delay lines (see, e.g. [70]), while the DLL in JACK is more strictly related a PLL, employing a second order filter and a means to obtain an error signal.

4.6.2 Resampling Algorithm

Let $\hat{F}_1(t)$ and $\hat{F}_2(t)$ be the DLL estimate of clock frequencies F_1 and F_2 at instant t. If the relation $\hat{F}_1(t) < \hat{F}_2(t)$ stands true for a sufficient amount of time, the second device goes overrun after some time. To solve the problem, the receiving end must resample according to a ratio, calculated between the two estimated clock as

$$R_r = \frac{\hat{F}_1}{\hat{F}_2} \cdot \frac{\frac{P_2}{F_02}}{\frac{P_1}{F_01}} \cdot R_u \tag{4.17}$$

where P_1, P_2 are the period size at both ends, and F_{01}, F_{02} are the nominal sampling rates at both ends, while

$$R_u = F_{02}/F_{01} \tag{4.18}$$

is the uniform resampling ratio, if the two nominal frequencies differ. Some of these parameters (P_2, F_{02}) are obtained from the transmitter when negotiating the communication, while the $\hat{F_2}$ is estimated and added by the transmitter at each period that is sent with the packet header. The packet header, thus increases from 12 bytes to 16. With typical periods of 128 frames of 16 bits (i.e. 512 bytes), the overhead increase is negligible: from 2.3% to 3.1%. The $\hat{F_1}$ is estimated at the receiving end at each cycle. Both frequency estimates are calculated from JACK DLL period estimate.

Adding adaptive resampling required modifying the `ringBuffer` class in jacktrip. Referring to the ring buffer formalism introduced in 3.2.4, recovery from underruns is done in jacktrip by spinning back the read pointer of $m = 1$ slots, while with overruns the read pointer is moved forward by $m = N/2 - 1$ slots. Both solutions may not prove efficient, i.e. they depart from the ideal case of Eq. 3.2.4, however, they require no assumptions on the skew. In other words, by moving the read pointer backward or forward (depending on the case to recover from) by $m = N/2 - 1$ slots, an assumption is made, i.e. that the over-/under-run condition occurred due to clock skew. If, on the other hand, an underrun occurred because of delayed or lost packets, but the clocks are skewed so that $F_1 > F_2$, without further occurrence of packet loss or delay, placing the read pointer close to the write pointer will reduce the time to the next overrun. The approach of the original jacktrip implementation was to make no assumptions regarding clock skew, and the behavior has been retained for the modified version as well.

The adaptive resampling mechanism requires to read an arbitrary number of samples, that are fed to the resampler and then sent to JACK for playback. For this reason the original jacktrip `RingBuffer` class has been replaced by the JACK API `RingBuffer` class. Resampling has been implemented by making use of *zita-resampler*[13], an open-source library by Fons Adriaensen implementing polyphase filtering for audio resampling. This library provides notable features compared to other open-source libraries such as libsamplerate. Its computational cost, compared to libsamplerate is lower, even at the highest quality setting.

Of specific interest is the `VResampler` class in Zita-resampler, that allows arbitrary ratios $1/16 \leq r \leq 64$ for the resampling factor. The algorithm that is employed is a polyphase filter performing a constant bandwidth resampling in the spectral domain. Let

- F_{in}, F_{out} be the input and output sample rates,
- F_{min} the lower of the two,
- F_{lcm} their lowest common multiple,
- $b = F_{lcm}/F_{in}$,
- $a = F_{lcm}/F_{out}$,

While an ideal resampler would perform interpolation and decimation of integer factors, the resampler exploits b FIR filters in polyphase fashion. All these filters have the same frequency response, but different delays that correspond to the relative position in time of the input and output samples. The FIR filters are approximation of ideal anti-aliasing and anti-imaging filters, thus a trade-off between computational cost and aliasing is done by targeting the resampler at between the common audio sample rates (44.1, 48, 88.2, 96, 192 kHz), and considering that consequently frequency response errors and aliasing will occur only above the upper limit of the audible frequency range. Given this assumption, a trade-off is made by dimensioning the polyphase filter to reach an attenuation of 60dB at the Nyquist frequency. As a result, aliasing is $\leq -110dB$ for all the range $0 - 20kHz$ at sample rates of 44100 or 48000 kHz.

[13]http://kokkinizita.linuxaudio.org/linuxaudio/zita-resampler/resampler.html

4.7 Wireless Transmission in Critical Contexts

In previous works by the authors [68, 60, 64], short-range indoor transmission is taken in consideration as the target scenario. As a result, commercial USB WiFi chipsets (as documented above) have been installed and tested on WeMUST OS. These generally feature omnidirectional 2.4GHz/5GHz patch antennas or larger dipole antennas.

In Sec. 4.8 a live performance will be reported which took place on boats over the sea. The conditions imposed by this new scenario make a carefully designed wireless transmission critical for the outcome of the performance. Specifically, the longer distance to cover, the presence of water, moving obstacles and metal objects (other boats, decks, etc.) require the RF link to be conceived differently. The first option to consider is employing directional antennas to increase the range and reduce scattering and reflections. The obvious shortcoming is to keep the antennas in line-of-sight as much as possible. Additionally, to increase on robustness, redundancy can be exploited, by creating a different RF link for each BBxM, at a different frequency band. This, although increases the BOM (Bill Of Materials) for a live performance, guarantees greater reliability. Finally, it is generally suggested to use the 5 GHz ISM band instead of the 2.4 GHz one, as it is less crowded and antenna patterns are narrower compared to a 2.4GHz antenna of the same size. Obviously a field test to look for potential interferers is mandatory, in order to avoid channels occupied by competing networks with strong SNRs (Signal to Noise Ratio).

To facilitate the deployment and design of the performance, a set of wireless devices from Mikrotik [14], the SXT 5HnD [15] have been employed. These devices are configurable wireless stations/access points, comprising a 10/100 Ethernet port, a directional antenna, driven by a high-power RF amplifier, and a microprocessor. Their size is slightly bigger than that of the BBxM with the antenna surface having a diameter of 140 mm. Configuration is done by software and a large set of features are available, including routing, bridging and acting as an access point. Wired and wireless interfaces are totally configurable and

[14]http://www.mikrotik.com/
[15]http://routerboard.com/RBSXT

the RF link can be set up using standard 802.11 modes or even proprietary high-throughput protocols. The SXT can also scan an area for networks and monitor SNR and link quality during transmission. As a result, each BBxM drives an SXT by employing the Ethernet port, instead of using a local bus as it would be done with USB adapters.

To assess the viability of the solution and SNR margin for live usage, preliminary experiments have been conducted in an indoor space of 19x9 m. This space, although not ideal, is free from neighboring wireless networks, obstacles or reflective surfaces and stands as a good reference for best-case transmission, with SNR values very close to the maximum ones achievable. Tests were performed setting up three separate 802.11a links, on three adjacent frequency bands, of 20 MHz width each inside the 5 GHz ISM band. The antennas and BBxM were put on the side opposite to the mixing PC, i.e. at 19 m. The SXT can automatically set for the best data rate, however in these favorable conditions, the data rate always stays at the maximum 54Mbps and the SNR is quite stable at an average 110 $\pm$2dB with maximum Tx power. For what concerns the used bandwidth, in these conditions, when sending stereo signals in both directions with 32-bit samples at 48kHz, the SXT monitoring utility measured Tx/Rx rates of 1.7Mbps/1.7Mbps including the overhead added by Jacktrip and 802.11.

In a real-world communication scenario the SNR can drop dramatically from the 110dB figure reported above, thus to properly design the RF link parameters it is necessary to collect statistics on link quality and find SNR margins that guarantee a very low rate of audio glitches occurrence. In order to gain insight on this, a set of tests has been performed reducing the SNR by decreasing the transmission power. The interest in this context is on the rate of audio periods lost, i.e. those that did not arrive in time or were not received at all. The Period Loss Rate (PLR) index is, thus, introduced, which measures the rate of lost periods over the total periods in the time unit. To evaluate the PLR sine tones with offset were generated from the BBxM, sent to the mixing PC and recorded there for later analysis by software. When the receiver is missing a period it writes zeros to the output file, for easy detection of lost periods. The

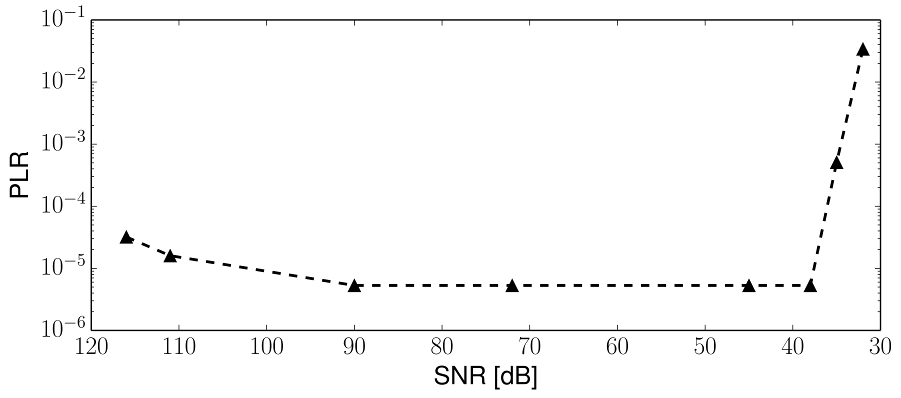


Figure 4.19: Period Loss Rate tests at different SNR. All tests have been conducted at the same Data Rate (36Mbps) but decreasing Tx power, thus decreasing the SNR.

test results are reported in Figure 4.19. As the Figure shows the salient aspect is the sudden increase of lost packets when the link SNR falls below 38 dB. Unexpectedly there is also a slight increase in the packet loss at very high SNR values due to nonlinear effects at the RF frontend of the SXT (most prominently intermodulation and input signal saturation). It is suggested, thus, that a best-case scenario for evaluating transmission performance is not the first one reported above, with a very high 110 $\pm$2dB SNR, but a lower Tx power configuration, which achieves an SNR between 90 and 40 dB.

In the attempt to reduce the SNR required to decrease the packet loss, several strategies have been implemented, including:

- varying the number of transmission attempts on a missing ACK,
- introduce redundancy in Jacktrip, i.e. sending systematically a packet more than once.

Unfortunately neither solution proves useful. A redundancy in Jacktrip even increases the PLR, due to the doubled bandwidth. Sending data over UDP devoids from the chance to ask for retransmission, but TCP is not a solution due to the large increase in latency and its unpredictability.

When collecting signals from N remote ends into one mixing PC, for the sake of reliability and robustness, it is suggested to create a link for each end. However, this N-by-N solution requires $2N$ SXT, plus one Ethernet network switch. Another feasible setup needs $N+1$ SXT only. In this case one of the SXT acts as Access Point (AP) and the other as Stations, however all communication is performed on one channel only, requiring enough bandwidth for transmission of all the signals. Furthermore there is risk of collisions and increase of transmission delays due to errors at one end. In other words, the isolation provided by the more robust solution allows the performance not to be affected by a broken link. PLR tests have been performed for a 3-to-1 case, analog to the Waterfront case study described in Section 4.8, together with a multichannel 1-to-1 case involving the same amount of traffic. Figure 4.20 provides PLR results for two data rates and three different configurations. The baseline is the first case where two bidirectional stereo channels are exchanged between two nodes. The other two cases share the same overall bandwidth handled by the node at the mixing PC end, i.e. 4.6Mbps and 9.2Mbps (excluding overhead). In the 3-by-1 scenario, these cases are implemented:

- three nodes sending stereo signals to a mixing PC and the latter sending a stereo mix back to them, for a total of six bidirectional streams (average 5.2Mbps uplink, 5.2Mbps downlink)
- three nodes sending 4-channels signals to a mixing PC and the latter sending a 4-channel mix back to them, for a total of 12 bidirectional streams (average 9.9Mbps uplink, 9.9Mbps downlink).

In the 1-to-1 scenario 6 or 12 audio channels are sent in both directions (with total 5Mbps and 9.7Mbps uplink and downlink).

As the Figure shows the PLR increases with the number of channels for both tests, however, as expected, the 1-to-1 case has a lower PLR due to the decreased chance of collisions.

For the sake of comparison, it must be considered that with the N-by-N solution the PLR stays equal to the baseline for all the connections. Reducing the PLR for multichannel or N-by-1 requires further effort which is left to future works, however viable alternatives may exploit the proprietary TDMA protocols available with the SXT.

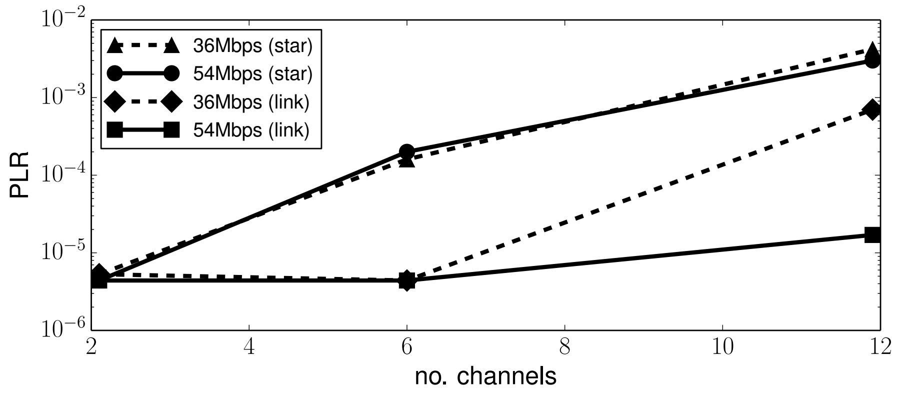


Figure 4.20: Period Loss Rate tests with different multi-channel configurations. Please note, values of $0.74 \cdot 10^{-6}$ (bottom values) mean that no period loss was observed during an interval of 60 minutes.

All the results presented in this Section are for single-hop networks. Should a longer signal chain be devised requiring multiple hops, further evaluation should be done following the procedures in [71].

4.8 Waterfront: a Networked Music Performance Experience

Once the viability of the approach has been assessed and the use of the SXT as a transmitter has been validated a live performance has been envisioned to showcase the framework and receive a feedback from trained musicians. The performance was conceived for *Acusmatiq*, a live electronic music festival, taking place in Ancona, Italy, in the magnificent frontage of the *Mole Vanvitelliana*, an 18th-century pentagonal building constructed on an artificial island close to the port of Ancona and other historical buildings. Nowadays, the Mole has decks for motor boats and sailboats, metal obstacles (such as metal lift docks) and wireless networks for maritime purposes which can endanger critical wireless transmissions. The goal was to have acoustic musicians play on separate boats at the frontage of the Mole, each with its own equipment to stream their signal to the land, and receive back a mix of the ensemble playing. The audience would stand on the land and be free to roam, exploring the acoustic space and observe the musicians from different vantage points. The performance has been called *Waterfront*.

Figure 4.21 reports the satellite view of the area. The maximum transmission distance is that between the land antennas and the double-bass player (2 and 3 in Figure 4.21), i.e. 90m. The shortest is that between the land antennas and the vocalist, (2 and 5), i.e. 30m. The pier (6), is 54 by 14 m wide. The antennas are placed on top of the fortified wall, at 3 m height, to leave space for the audience to rom along the pier.

After a careful selection of possible artistic proposals and performers, the choice converged on a trio performance of *Solo* by Karlheinz Stockhausen. The piece, dated 1966, explores on a solo performance (hence the name), augmented by feedback tape delays which selectively loop some of the acoustic material to enrich the scene and provide a seemingly ensemble experience. As opposed to this, with Waterfront, the three musicians are granted a moment of solo performance each, but join finally into an impromptu ensemble where each instrument gains its space competitively, reinforced by the distorted recurrence of the feedback delays. The whole performance lasts approximately 50 minutes.

The original piece required four technical assistants to manually drive the delay taps, gains and filters according to six different variations suggested by Stockhausen himself in form of notated scores. In this piece the signal processing was conducted digitally with *SOLO nr.19*, an iPad application. Each of the musicians had its own iOS device running the aforementioned application, which captured the instrument signal and supplied the BBxM with the signal. This also compensated for the lack of a microphone preamp on the BBxM, since the iPad outputs a line signal, compatible with the BBxM line input. The iOS devices add latency to the signal chain. This was discussed with the musicians which could test with different setups and latency settings and finally decided to have a slightly larger latency but perform with a less cumbersome equipment. The processed version of the signal is insensitive of the low processing delay added by the device, since the delays imposed by the Solo score can be up to several seconds long. As a side note, considered the architecture of the We-

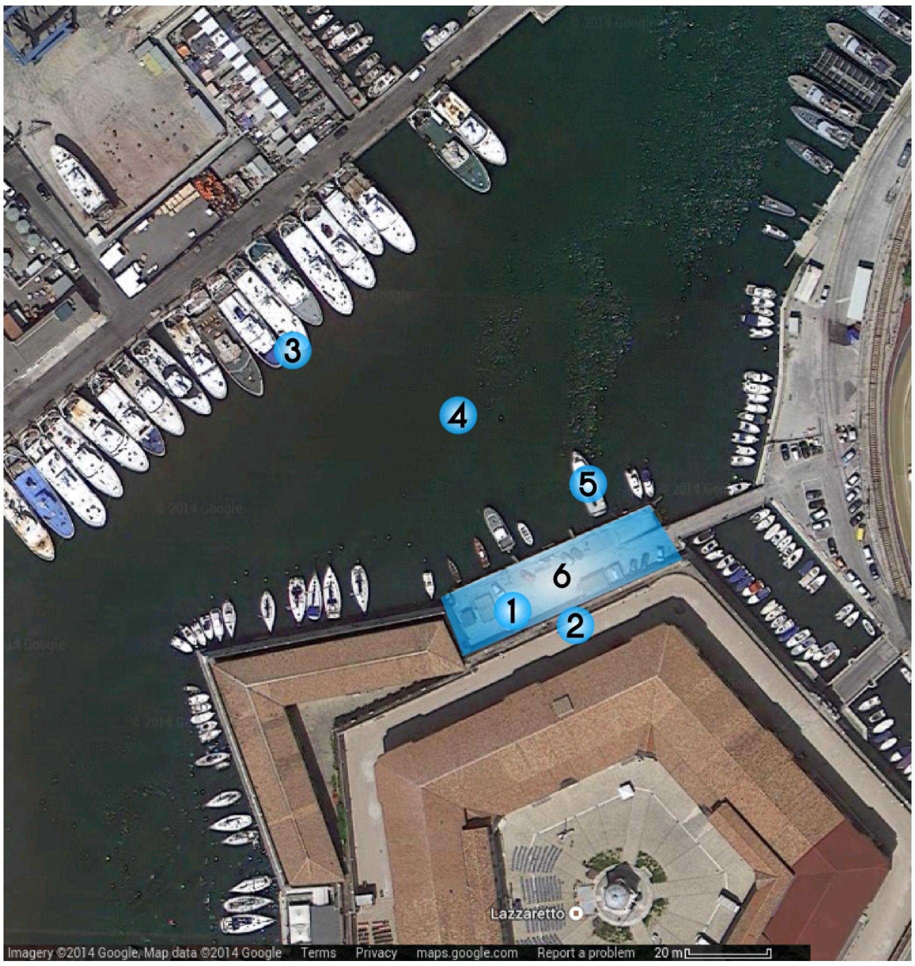


Figure 4.21: Satellite view of the Mole Vanviteliana and the sorrounding area. 1) direction console with master PC, 2) land antennas standing on the Mole fortified wall, 3) double bass, 4) saxophone, 5) vocalist, 6) audience

STATUS	TX POWER [dBm]	SXT [mA] at 24 V	BBxM [mA] at 5 V
Searching	17	115	465
Waiting	17	125	
Transmitting 24Mbps	0	143	600
	12	145	
	17	148	
Transmitting 54Mbps	0	134	
	12	136	
	17	138	

Table 4.3: Current consumption for the SXT and the BBxM under different operating modes.

MUST framework, in order to reduce the use of hardware devices a countdown and a *click* to synchronize musicians can be supplied from the MPC, together with all the signal processing required by Stockhausen's score (porting of the signal processing in Solo exist for CSound and other languages). This would also reduce the latency added by the iOS device. The musicians however felt more comfortable in carrying their own iOS device which provides also visual feedback.

A video of the performance is provided at the A3LAB research group WeMUST page[16].

4.8.1 Energy Supply

To evaluate the energy requirements of the system current consumptions have been measured for both the BBxM and the SXT. Current probes have been used to measure current consumption, while the voltage supplied by the regulated power supply is constant at 5 V for the BBxM and 12 V for the SXT independent of the output current. Power requirements have been measured during operation, and are reported at Table 4.3 as a function of the Data Rate and the Tx Power.

The musicians have been supplied with a BBxM and an SXT each. To power

[16]http://a3lab.dii.univpm.it/research/wemust

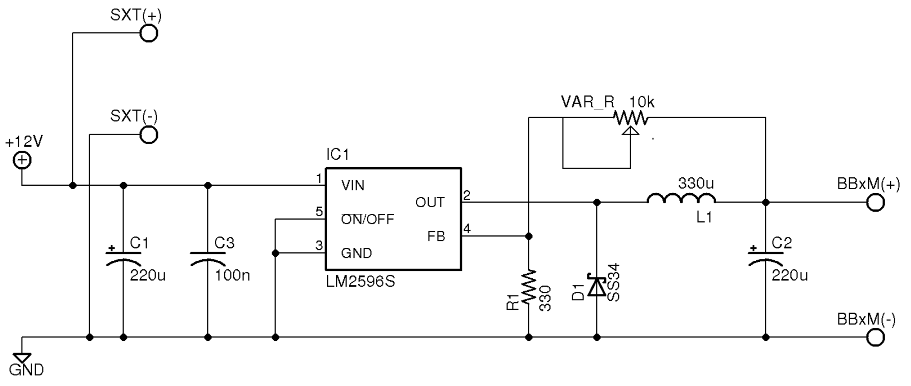


Figure 4.22: Schematic diagram of the battery supply circuit.

this equipment commercial-grade 12V 4.5Ah rechargeable Lead-Acid batteries have been used, enabling sustained operation. The voltage must be regulated to fit the requirements of the equipment. A switching voltage regulator, the LM2596 (adjustable output version), has been employed. This allows up to 2A output current (without heat sink) and input voltage in the range 4.5V to 40V. The efficiency depends on the output voltage. For the case at hand ($V_{IN} = 12V, V_{OUT} = 5V$) the efficiency is 80%. The typical quiescent current is 5 mA at 25 C. To meet more stringent power constraints, the quiescent current can be reduced to $80\mu A$, by switching the component off (toggling pin 5, shown in Figure 4.22). The SXT have an internal regulator and can be directly supplied with 12V.

During indoor transmission tests the battery life has been evaluated. As expected the batteries performed slightly different from each other, however the minimum battery life documented in the tests is of 3 hours of continuous transmission plus 1 hour of waiting. Given the figures shown in Table 4.3, the total power required by the system while transmitting is 6.3 W.

4.8.2 Audio Signal Routing and Monitoring

The signal routing is depicted in Figure 4.23. As mentioned beforehand, the BBxM acquire the line signal from the iOS device, which is used to capture the sound from the acoustic instrument, and output the original and processed version. The BBxM acquires the signal and transmits it over to the SXT, which acts as a transparent bridge to the mixing PC. As all the signals are acquired at the master PC, they are mixed exploiting a software mixing console. The direction could speak to the musicians with an open microphone connected to the MPC sound card. By proper routing (which can be usually automated by means of scripts and presets) the signals are mixed for the public address system and are selectively sent to the musicians, to allow each one to have its own earphone monitor mix, to ensure the best interaction experience. All the musicians experience, thus, the same round-trip time, which, with the final settings agreed for the premiere performance, was 27.4 ms, i.e. 15.7 ms (as documented in Section 4.4.3) plus an iOS latency of 11.7 ms (128 samples buffers, includes some proprietary microphone signal preprocessing). As stated above, the latency added by the iOS devices can be reduced by moving all the signal processing inside a JACK client in the MPC.

During the performance audio glitches have been partly masked by employing a rudimentary error concealment mechanism available in Jacktrip, i.e. buffer repeating. By offline analysis on the performance recording, three glitches were spotted in approximately 50 minutes, up to the expectations from previous transmission tests.

During the Waterfront premiere, a MacBook Pro was used as the MPC, running *JackOSX*, Jacktrip, *qjackctl* and *AULab*. Equivalently, on a Linux machine, JACK, Jacktrip, *qjackctl* and *jack-mixer* can be employed to obtain similar features. For the sake of a stable performence, a Linux machine was employed, running *wemust-netdbg* to monitor the state of the connections. This configuration requires one technician managing audio, levels and musicians' requests and another one to control the state of the connections. With the system getting more mature both audio and monitoring tools could run on the same machine.

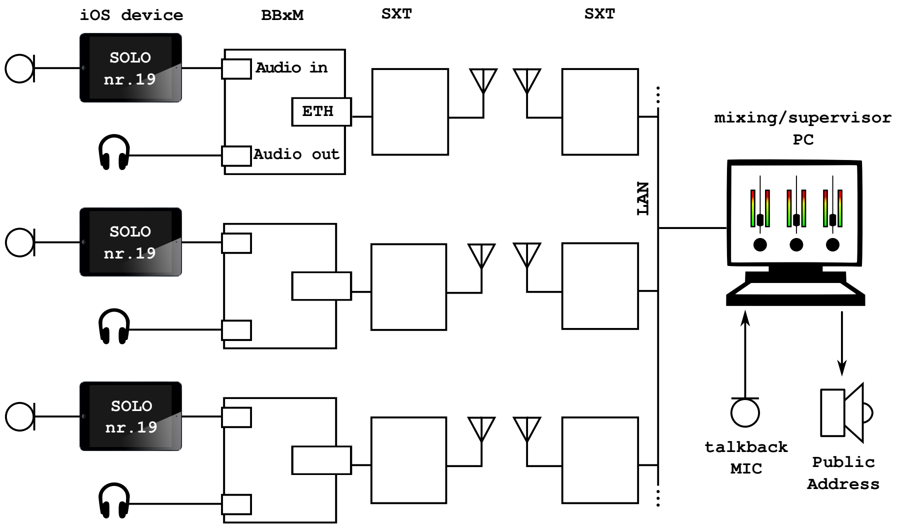


Figure 4.23: Overview of the hardware and connections employed in Waterfront.

4.9 Future Perspectives

The description of WeMUST has been targeted along the paper to introduce Waterfront and similar applications, where many endpoints acquire signals and send it to one mixing PC. The framework, though, has further features and potential.

As mentioned beforehand, the processing involved with Solo, could be moved to the mixing PC, to free the musicians from the use of a tablet device, or to sustain computation of algorithms of higher complexity that require increased computational power to be computed in real-time. Another option for Waterfront could have been to port signal processing to the BBxM, in form of C/C++ or Supercollider[17] code, Puredata[18] sketches and the likes. As long as the BBxM is the platform of reference for WeMUST, signal processing must be rather optimized to fit the computational power of the single core Cortex-A8 at low period size. One downside of employing a headless platform is the lack of visual cue or screen to provide musicians with information, a visual metronome, etc. On the other hand, modern tablets have good computational

[17]http://supercollider.sourceforge.net/
[18]http://puredata.info/

resources and large displays but lack openness of development platforms and are weighted down with general purpose software of no use in this context.

In Waterfront no control data has been exchanged. However, MIDI or OSC data could be exchanged along the network, e.g. to remotely trigger events or control one of the endpoints running some processing or synthesis algorithm. Another use for control data typically found in networked laptop orchestras [72, 73], is a broadcast of status messages, control parameters or data exchange for multi-agent or distributed architectures. All these options are allowed by WeMUST, as any device connected to the network can transparently send or receive data to and from any other device. Connections and devices on the network can dynamically change. Any device can be addressed by its IP addresses or hostname, and these can be discovered by use of the SABy protocol [60] or Avahi.

One last noteworthy feature enabled by WeMUST is to ensure the musicians the ability to synchronously play a score, by sending a metronomic *click* from the MPC or a conductor device to the endpoints.

Chapter 5

Real-Time Digital Signal Processing Algorithms for Embedded Platforms

The previous chapter provided details on the development of a platform devoted to MAs such as NMP. Other MAs suitable to digital signal processing are reported hereby from a theoretical standpoint with a focus on the porting to embedded platforms and/or scalability. All these algorithms have been tested on the WeMUST embedded platforms to various extents and different outcomes.

5.1 The Discrete Wavelet Transform for Low Computational Cost Algorithms

The DWT is a family of multi-resolution transforms, very common in many fields of signal processing and employed in applications ranging from audio to geophysics. Its importance stems from its good resolution in time and frequency. Unlike DTFT, DFT and STFT it does not barely provide a linearly spaced representation of the spectrum, which do not match the logarithmic nature of the human ear, but gathers more detail where it is most needed, i.e. at the low range of the human hearing. In the musical field its usage is motivated by its multi-resolution spectral representation, which is similar to some extent to the octave-spaced distribution of a musical scale. The DWT can be evaluated sample-by-sample or in frames[1] of data, while the STFT is computed only on a frame-by-frame basis. While the DWT presents extremely good time resolution properties, the STFT needs overlapping, i.e. redundant computation of overlapping frames, to increase its time resolution consistently. Since the STFT is based on DFT computed for successive windowed frames, it can employ FFT algorithms to speed up computation.

A brief overview on the DWT following the same notation used in [74] is provided hereby. The DWT can be implemented in the digital domain as a dyadic filter bank [75], shown in Figure 5.1

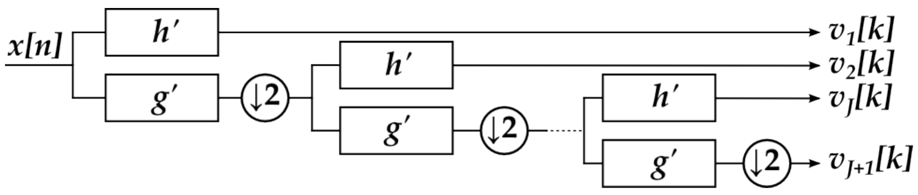


Figure 5.1: Flow graph representation of a J-level Discrete Wavelet Transform (analysis part).

Moving from the following notation equalities:

$$\begin{aligned} &Upsampling: \ (\uparrow x)[2n] = x[n], \ (\uparrow x)[2n+1] = 0, \\ &Downsampling: \ (\downarrow x)[n] = x[2n] \end{aligned} \tag{5.1}$$

$$\begin{aligned} &\mathbf{G}\,(g_n \ approximation\, filter) \quad (\mathbf{G}x)[n] = \sum_k x[k]\cdot g[n-k], \\ &\mathbf{H}\,(h_n \ details\, filter) \quad (\mathbf{H}x)[n] = \sum_k x[k]\cdot h[n-k] \end{aligned} \tag{5.2}$$

where $\tilde{g}[n] = g[-n]^*$ is the paraconjugate operator, and k is the index of the wavelet coefficients, looking at the decomposition part, it can be observed that the original signal $x[n]$ is processed through filtering and down-sampling

[1]not to be confused with the audio frame term used in Chapter 4 related to audio buffering in Linux

operations resulting in different sequences:

$$\begin{aligned} v_j[k] &= \sum_k x[n] \cdot \tilde{h}'_j[n-2^j k], \quad j = 1, ..., J, \\ v_{J+1}[k] &= \sum_k x[n] \cdot \tilde{g}'_J[n-2^J k] \end{aligned} \tag{5.3}$$

where $\tilde{g}'_j = \left(\tilde{\mathbf{G}}' \uparrow\right)^{j-1} \tilde{g}'$, $\tilde{h}'_j = \left(\tilde{\mathbf{G}}' \uparrow\right)^{j-1} \tilde{h}'$. Each of these sequences has different length than the others; in more specific words, it means that their scale and resolution values are divided by 2 at each decomposition level, consequently reducing of the same factor the sequence length and the part of spectrum they represent. This fact is directly related to the coverage of time/frequency plane existing in continuous wavelet transform (CWT). The signal can be reassembled from the coefficients through filtering and up-sampling operation:

$$x[n] = \sum_{j=1}^{J} \sum_k v_j[k] h_j[n-2^j k] + \sum_k v_{J+1}[k] g_J[n-2^J k] \tag{5.4}$$

where $g_j = \left(\mathbf{G} \uparrow\right)_{j-1} g$, $h_j = \left(\tilde{\mathbf{G}}' \uparrow\right)_{j-1} h$. However, since this filter bank is critically sampled, used filters are constrained to satisfy the following condition to achieve perfect reconstruction (without delay), here valid in case of a simple two-band filter bank:

$$x[n] - \mathbf{G} \uparrow\downarrow \mathbf{G}' x[n] = \mathbf{H} \uparrow\downarrow \mathbf{H}' x[n]. \tag{5.5}$$

This can be easily extended to the j-level decomposition case.

Of lesser interest due to a linearly-spaced spectral representation is the wavelet packet decomposition, or DWPT (Discrete Wavelet Packet Transform). This transform is similar to the DWT in that it uses quadrature mirror filters with detail and approximation wavelet filter coefficients as those used in DWT, but the bank tree is a full binary tree. The sub-bands output coefficients (called WPEC, Wavelet Packet Energy Coefficients) all subject to the same group delay and sampling rate. This property is thus of interest in some cases, as discussed later. Figure 5.2 reports the SFG of the DWPT.

The computational costs of the DWT and WPEC are roughly similar. Fol-

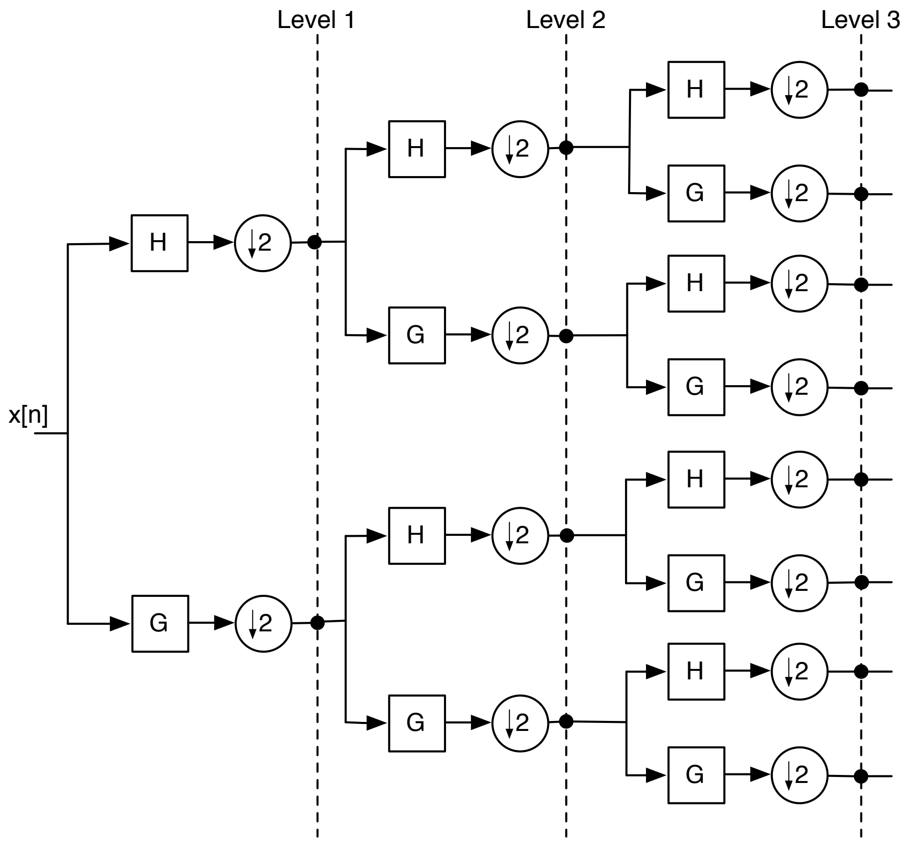


Figure 5.2: The Discrete Wavelet Packet Transfrom (analysis, or *decomposition* fitler bank).

lowing the assumption that real-valued sums and products require one CPU operation, an FIR filter of order M requires $(2M - 1)$ operations per input sample. The computational cost of an M coefficients wavelet decomposition down to level j is

$$C_D = \sum_{i=0}^{j-1} \frac{2(2M-1)}{2^i}. \tag{5.6}$$

Easily it can be shown that

$$\lim_{j\to\infty} C_D = 8M, \quad \text{for} M \gg 2, \tag{5.7}$$

and this stands true already for low values of j. The value $C_D = 8M$ represents an upper bound for computational complexity and overestimates computational cost for very short filter kernels (e.g. Haar wavelet).

Similarly, the wavelet packet decomposition down to level j is

$$C_D = 2(2M-1) + \sum_{i=1}^{j-1} \frac{4(2M-1)}{2^i} \to 12M \quad \text{for} M \gg 2. \tag{5.8}$$

Figure 5.3 reports the DWT and DWPT computational cost in terms of operations per input sample for different values of j and M. As a reference, the computational cost of a non overlapped STFT calculated by means of a M-bins FFT algorithm over a window of M samples is $C_F = 2Mlog_2(M) - 4M + 6$ [76, 77], i.e. approximately $2log_2(M) - 4$ operations per input sample. Overlapping may apply, introducing a increase inversely proportional to the overlap ratio. The choice between STFT, DWT or DWPT should not be done based only on the computational cost as they are meant for different applications and obtain very different time-frequency representations.

5.1.1 The Discrete Wavelet Transform as a Morphing Tool

The DWT is usually employed as a (perfect reconstruction) filter bank to conduct processing in the transformed domain. In this section and in the next one, however, two slightly different usages are shown.

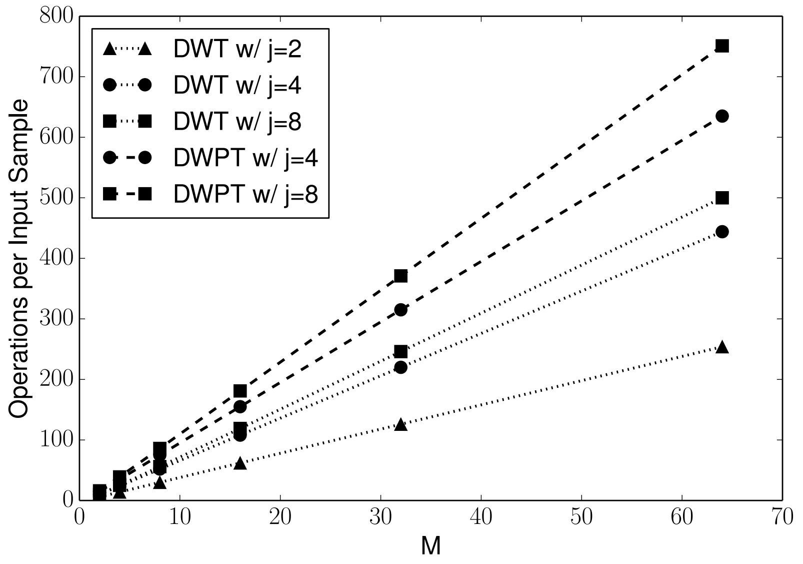


Figure 5.3: The computational cost of different DWT configurations. The computational cost of the DWT is also reported.

In [78], a tool to conduct signal morphing or hybridization in the Wavelet domain was reported and demonstrated. Hybridization is a family of techniques to create a tone which retains some of the features of two original source tones, while morphing is the dynamic change of a sound from a source tone to another transitioning through a series of time intervals where the produced tone retains some features of both. Ibrida, the software tool (based on Pure Data) reported in [78] allows for both. The user interface for Ibrida is reported in Figure 5.4. The algorithm reported in Figure 5.5 transforms two signals of equal sample rate in the Wavelet domain, and sums the sub-bands of the two that have equal sample rate. The result is the linear mix of the Wavelet coefficients of the two signals, which can be transformed back in the time domain by inverse DWT. Since the sum of the sub-band signals is a linear operation, the perfect reconstruction property of the DWT-IDWT filter banks is guaranteed. The effect generated by the weighted sum is a spectral mixing of the two signals, such that for each subband it can be decided how much of the two source signals to feed to the output.The algorithm provides several degrees of freedom. The weights of the linear mix can be modified for each branch to allow one of the two sources to be predominant and the wavelet family can be chosen.

"phd-thesis" — 2015/3/23 — 10:33 — page 106 — #132

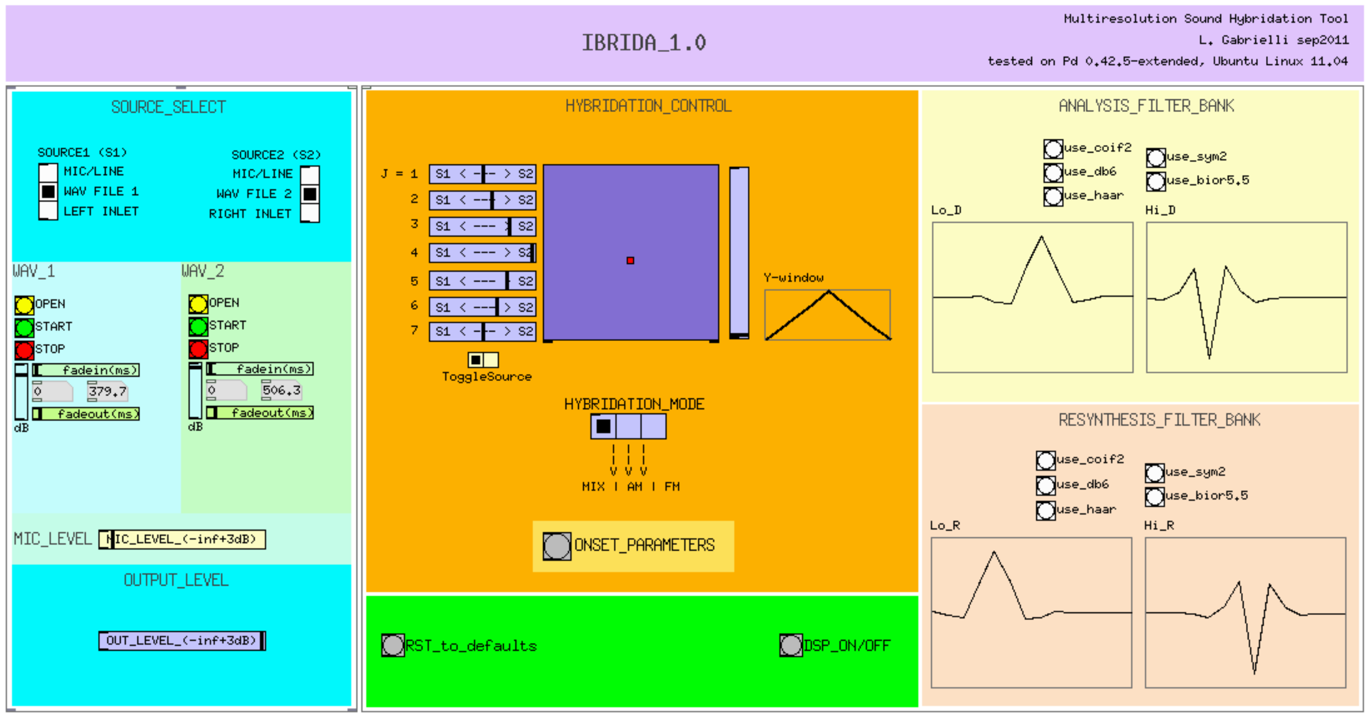


Figure 5.4: The graphic user interface of Ibrida. The weights for mixing two sources in the DWT are controlled by the purple sliders (one per sub-band). The Onset_Parameters trigger allows for morphing from Source 1 to Source 2.

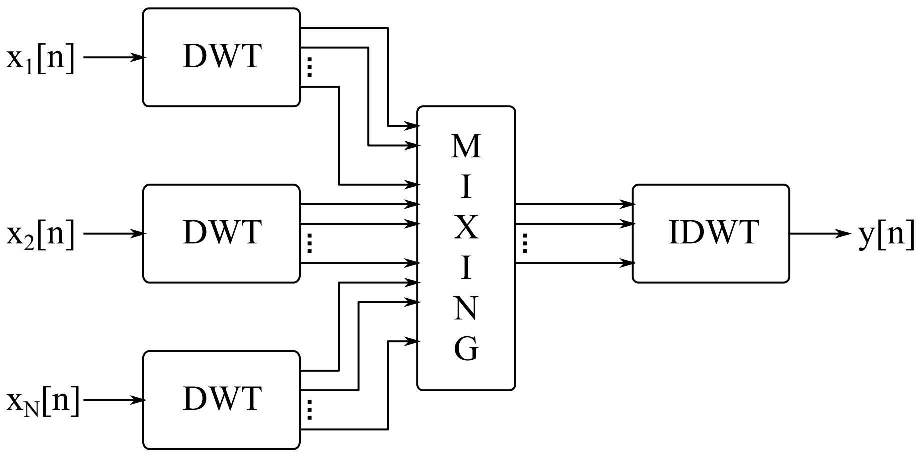


Figure 5.5: The SFG of the Ibrida algorithm.

The evaluation of hybridization and morphing techniques is left as a future work, since there is no objective method to assess the quality of a technique with respect to another. Meaningful comparison can be done subjectively, if the listeners are prepared to the topic and a set of criteria are given for the evaluation. For an interesting overview and taxonomy of signal processing techniques for morphing and hybridization refer to [79].

5.1.2 The Discrete Wavelet Coefficients as a Feature for Onset Detection

As discussed the DWT collects coefficients in the transformed domain for each subband or branch. The coefficients collected from each branch have different sample rate (exception made for the lowest two details and approximation sub-bands). For this reason practical implementation of DWT filtering or processing is not obvious and requires some care. Even less obvious is the case first introduced in [74] and further expanded in [80, 81], where the DWT coefficients are used as features for a musical note onset decision algorithm, and thus are not transformed back in the original domain. This introduces a problem of alignment of the subband signals and of linking between these and the original signal timebase to place correctly the onset instant related to the input music signal. The alignment constraint simplifies the nontrivial task of selecting the Wavelet filter design method, since it imposes the filters to be linear phase property, reducing the number of the design methods worth to take into consideration. Linear phase FIR filters have the property of imposing a constant phase delay, making it easy to align the subband signals by simple sample shifting (no pun intended). Biorthogonal wavelets have the advantage over orthogonal wavelets in that they can be designed to be linear (such as the Haar wavelet) or nearly linear phase (such as the Coiflet). The Coiflet functions [82] have been selected. The linear phase of other biorthogonal wavelets (see e.g. [83], pp. 271-280) has been traded for a higher number of vanishing points for a given order. Both contribute to increase the convergence properties of subsequent linear prediction filters (later discussed). The nearly linear phase property ensures anyway a negligible misalignment when aligning the filter according to a constant sample shift, and the better convergence properties of the Linear Prediction Error Filters (LPEF) cascaded to Coiflets DWT with respect to the linear phase biorthogonal wavelets have been shown during experiments. For the sake of clarity, the nearly linear phase property (or near simmetry) stands when

$$\phi(\omega) = k\omega + o(\omega^N) \quad for \quad \omega \longrightarrow 0. \tag{5.9}$$

Under this condition the group delay can be approximated to the one of a linear phase filter, i.e. $(N-1)/2$ samples, where N is the length of the FIR impulse response. Once the filter delay is known, a simple synchronization mechanism can be designed to align all the subsignals by simply applying different delays for every channel. Since the delay increases while descending the decomposition tree and the sampling frequency decreases, the upper subsignals must be compensated with higher delays, according to the following

$$d(j) = 2^{J+1-j} \sum_{i=1}^{J+1-j} \frac{N}{2^i} \tag{5.10}$$

where $d(j)$ is the delay to be added to channel j. By adding such a delay to each channel, after the DWT filter bank, synchronization is granted in case the subchannels are all resampled to the same sampling frequency.

An approach similar to DWT which, however, does not pose problems of alignment is the DWPT. The resulting signals are all decimated at the same sampling frequency and can thus be employed directly without any alignment or resampling issue. The output of the tree leaves are called Wavelet Packet Energy Coefficients WPEC (not to be confused with the coefficients employed by its FIR filters coefficients).

Both the DWT [80, 81] and the WPEC [84] approaches have been tested for onset detection in different works, also paired with neural networks and other spectral features. In [80, 81] the DWT has been used together with a LPEF (Linear Prediction Error Filter), following the concept in [85]. Each sub-band signal is processed by an LPEF whose coefficients are updated with each input sample by a modified version of the Normalized LMS (NLMS) algorithm [86] described below. The NLMS approach is chosen for its suitability to signals with large energy variations, such as music signals. In order to detect onsets by observing the prediction error, the step-size value for the j-th band, μ_j, is crucial,

$$\mu_j = \frac{\mu'}{|\mathbf{u}_j[k]|^2 + c} \tag{5.11}$$

where $0 < \mu' < 2$, c is a small constant to avoid division by zero, $\mathbf{u}_j[k] = (d_j[k-1], \ldots, d_j[k-p])^T$ represent the previous p input samples and $|.|$ acts as an estimate of the signal energy, which varies in time, making the step-size varying as well. However, if the convergence of the filter coefficients is too fast, the increment of the prediction error envelope at note boundary may became less evident, thus, a large value of the step-size is not desired. The modified step-size update considers silence regions (e. g., pitched non percussive), as reported in [87]:

$$\mu = \min\left(\frac{A}{rms[k] \cdot p}, \frac{1}{|\mathbf{u}_j[k]|^2}, 1000\right) \tag{5.12}$$

where $rms[k]$ is the root mean-square value of samples in a 20 ms window just after the *k-th* sample of $d_j[k]$. The constant A is empirically set to 0.5. The second term in the minimum operation ensures the convergence while the third term prevents the step-size from getting too large when the signal energy becomes very small. An issue with the varying sampling frequency in the different DWT sub-bands is that the order of the LPEF, p, should assume different values depending on the sub-band sample frequency. For the highest band $p_{max} = 24$ while for the lowest two bands $p_{min} = 16$. These parameter values are defined as result of several preliminary evaluations.

The output of the LPEF in the transformed domain gives an estimate of the *novelty* of the signal at a certain instant in a certain branch. Because of its whitening properties, the DWT is shown to increase the convergence speed of LMS adaptive filters in the transformed domain [88]. This provides a more accurate localization of error peaks and, thus, of onsets. Furthermore, with respect to a generic FIR filter bank the DWT filter have very short impulse responses.

To make a decision, it is necessary to combine signals from the sub-bands, i.e. resampling to a reference sample rate and then feeding those to a decisor. The reference sample rate has been chosen at least one order of magnitude inferior to the original signal one, since generally in note onset detection the time resolution need not be larger than 10 ms. The decisor can then combine them according to some logic or accept multiple signals. The latter case is of particular interest as complex Computational Intelligence techniques can be employed, e.g. Bidirectional Long Short-Term Memory (BLSTM) recurrent neural networks (NN) [89], which can be trained automatically on different databases to suit different genres without the need for a manual trial and error procedure to characterize coefficients. During experiments the BLSTM recurrent NN were employed to benchmark different feature extraction algorithms. It was shown that traditional feature extraction methods based on DFT-based spectral techniques are outperformed by the aforementioned Wavelet-domain LPEF coefficients and the Auditory Spectral Features (ASF), which are Mel spectrograms with two different time resolutions. As it often happens with NN adding dimensionality to the input can increase the NN performance. Employing both Wavelet-domain LPEF coefficients and ASF yields a higher accuracy of onset detection compared to the two techniques alone.

Similarly, in [84], the WPEC features are fed to BLSTM or other NN configurations to seek for the best detection accuracy. In both approaches, the DWT+LPEF and the WPEC have been tested alone, or combined with ASF. The tests have been performed for various combination of features and neural networks on the same datasets. The results, thoroughly listed in the related publications, show that the DWT+LPEF approach perform slightly better than the WPEC alone. Let the metric be

$$\text{F-measure} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.13}$$

with

$$\text{Precision} = \text{CD}/(\text{CD} + \text{FP}), \tag{5.14}$$

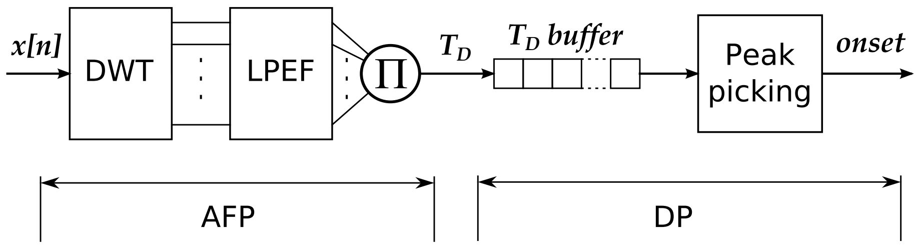
$$\text{Recall} = \text{CD}/(\text{CD} + \text{FN}) \tag{5.15}$$

where CD is the number of correctly detected onsets, FP the number of false positives and FN the number of false negatives. For a given BLSTM configuration the WPEC obtains 0.9 and 0.84 F-measure (depending on the tolerance window employed for evaluation of the results), while the DWT+LPEF reach 0.933 and 0.892 F-measure. The additional use of ASF complements the missing information to the BLSTM, providing very similar results, 0.941 and 0.906 F-measure for the WPEC+ASF, and 0.942 and 0.910 for the DWT+LPEF+ASF.

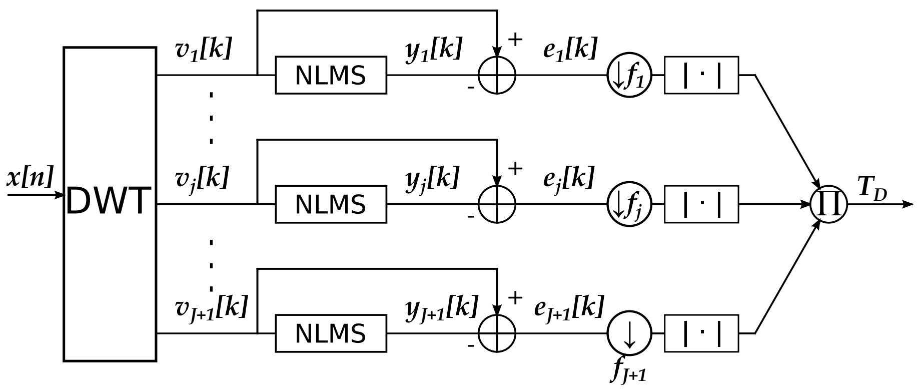
Notwithstanding the long history in NN research, NN implementation in embedded electronics is very rare. One relatively new option is to employ general purpose GPUs for NN algorithms [90]. GPUs provide a large speed up in neural network computation and can be foreseen as a remarkable improvement in the implementation of NN for daily tasks, since GPUs are available nowadays in both personal computers and mobile devices. However, computational approaches that do not employ NN are at the moment much more low on computational requirements and are better suited to embedded implementation. The DWT+LPEF approach described in [74] provided satisfying results without resorting to highly-expensive Computational Intelligence techniques. The sub-band LPEF signals were multiplied, as in multi-scale product algorithms. The onset decision was performed by comparing the product of a signal energy approximation with the signal first order difference with a given threshold. Figure 5.6 reports graphically the overall algorithm.

5.2 Novel Second Order Filter for Virtual Acoustic Feedback Emulation

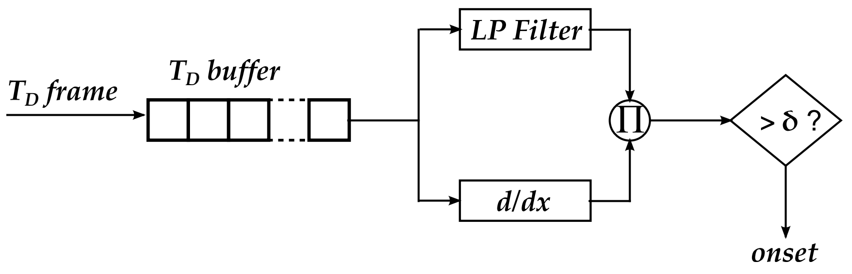
In the field of Computational Acoustics, a topic which has been scarcely addressed is that of Acoustic Feedback (AF) emulation. While acoustic feedback and echo removal is extensively treated in the Digital Signal Processing field for communication equipment, hearing aids and professional audio components, there is a handful of works regarding the emulation of AF as an expressive tool, or an inspiration for tone generation. In [91], the acoustic feedback is studied in the case of an electric guitar and an amplifying device. A database of records is discussed. The effect of interest originates from the feedback stimulation of the guitar string with the acoustic wave propagating from the amplifier and originated by the string itself. After rising, the feedback gets stable to a target amplitude, depending on several factors. This is due to the nonlinear nature of electronic amplifiers, which saturate over a certain signal threshold. There are other cases of acoustic feedback involving a guitar and an amplifying system which are annoying and are not employed for musical purposes. In these cases the coupling does not include the string. When the coupling includes the strings the effect is musical since the oscillation originating from the feedback takes place at a harmonic interval of the fundamental frequency of the string. In [91] and [92] digital implementations of the Virtual Acoustic Feedback (VAF) effect are discussed, originated from the emulation of the electric guitar case, but extending to unexplored scenarios. Being the most well known musical use of acoustic feedback, the guitar howling as been taken as a reference and inspiration. The typical setup, involves an electric guitar as sound source and a nonlinear amplifier with loudspeaker. A constructive feedback is generated



(a)



(b)



(c)

Figure 5.6: Overview of the onset detection algorithm targeted to embedded implementation (a). The adaptive filtering process (b) and the detection process (c).

when the sound pressure waves couple with one of the guitar strings. For the coupling to be remarkable the travelling waves must be in phase with the string and their amplitude high enough to overcome the string mechanical resistance and sensibly excite the string (either fretted or open). The coupled string acts as a resonator. As a corollary, it must be noted that howling can rise only at a multiples of the fundamental F_0 of the string.

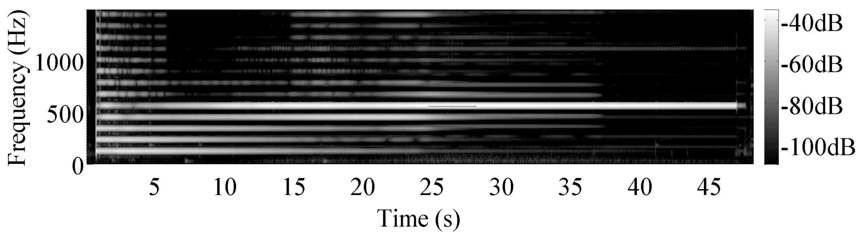
The system response from the guitar to the loudspeaker can be considered slowly time-varying (unless time-varying guitar effects are employed). The only portion of the system which can vary remarkably is the path between the musician and the loudspeaker. In a closed or even semi-anechoic environment there will be several paths able to sum in phase with a specific string. The howling will take place if at least one of the paths will violate the Barkhausen stability criterion [93], i.e. for a certain frequency ω_0 :

$$A(j\omega_0)G(j\omega_0) = 1 \Rightarrow \begin{cases} |A(j\omega_0)G(j\omega_0)| = 1 \\ \angle A(j\omega_0)G(j\omega_0) = 0 \end{cases} \tag{5.16}$$

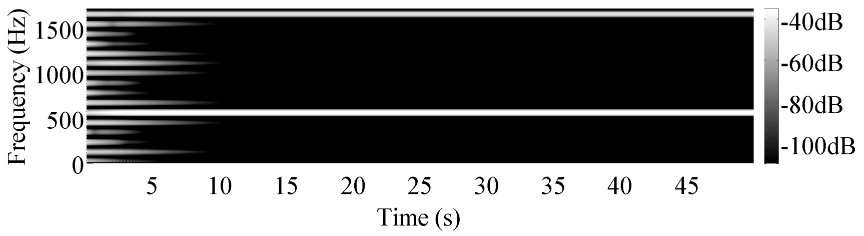
where $A(j\omega)$ is the direct path transfer function (instrument to loudspeaker) and $G(j\omega)$ the feedback path transfer function (loudspeaker to instrument).

If several paths are unstable, most likely one feedback path (i.e. one howling frequency) will prevail among the others, since, in unstable conditions, small deviations in the loop gain or phase delay will result in substantial amplitude difference between the paths after a sufficiently large amount of time. This has been experienced by computer simulations based on Sullivan's method. Tests have been conducted using Sullivan's method to gather insight on the guitar feedback phenomenon. Figure 5.7 compares a recorded A2 tone with howling rising at the 5th partial, and a simulated counterpart of similar characteristics employing a DWG guitar model as the sound source.

A good practice for digital effects design is to write down a specifications for the usability and control requirements, besides the technical specifications regarding the implementation. After evaluation of a database of tones recorded in a semi-anechoic chamber, some desired features to replicate guitar howling and introduce more practical control were:



(a) Recorded guitar howling



(b) Simulated guitar howling

Figure 5.7: Comparison of the spectrograms of a recorded guitar howling (a), and a simulation using Sullivan's method (b). Both tones are A2 (110Hz) with howling occurring at the 5th partial.

- to obtain a stable oscillation with peak amplitude limiting,
- to give a precisely tunable frequency of oscillation f_o at a multiple of the incoming fundamental frequency F_0,
- to emulate the nonlinearities (e.g. distortion, strings nonlinearities, etc.),
- to select the rise time,
- to select the amount of the effect, i.e. a dry/wet gain.

The problem under investigation can be generalized in terms of oscillators. In literature there are examples of digital oscillators mainly devoted to virtual analog modelling [94] or physical modelling of passive structures [95, 96]. While the former are generally complex and computationally expensive, the latter do not apply to the current case, which is not passive and may exhibit growth as well as decay. In order to fulfill the specifications introduced above, a positive feedback oscillator, depicted in Figure 5.8, can be employed. This oscillator consists of a selective bandpass filter in positive feedback $G(\omega)$, to select the desired harmonic and a memoryless nonlinear component $\beta(\cdot)$ to engage oscillation and prevent instabilities.

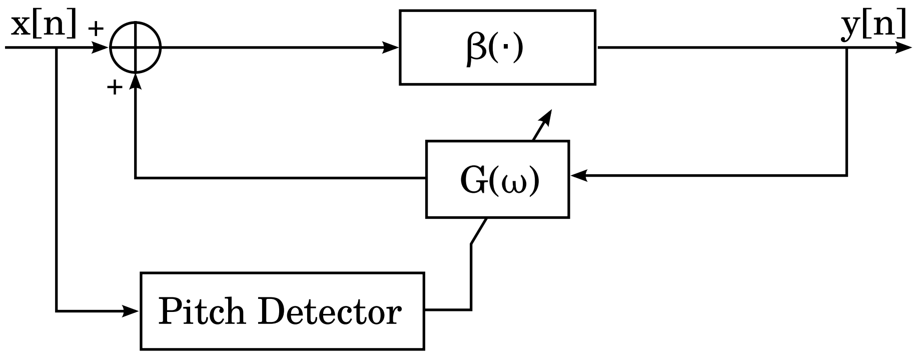


Figure 5.8: General scheme of a digital oscillator with nonlinearity $\beta(\cdot)$ and bandpass transfer function $G(z)$.

Linear Filter Design

The nonlinear component of the oscillator is neglected (i.e. $\beta = 1$) in order to design the coefficients of the linear part and obtain the overall transfer function $H(z)$ of the oscillator.

The bandpass filter must be selective enough to reject all harmonics beside the desired one. Second-order designs are sufficient for this task. However, when placed in a feedback loop, problems of computability may arise. Let $G(z) = B(z)/A(z)$ be the bandpass transfer function and $H(z) = N(z)/D(z)$ be the entire oscillator transfer function. If the bandpass direct path filter coefficient b_0 is not null, the output $y[n]$, passing through the bandpass direct path, is fed back without any delay, making the system output impossible to compute in a discrete time environment. By observing that the oscillator always has a direct signal path to the output, $n_0 = 1$, the original solution proposed here is to design the bandpass filter to have $B(z)$ with first coefficient $b_0 = 0$. This ensures computability and does not affect the oscillation frequency of the oscillator. The frequency response of the oscillator in Figure 5.8 is that of a peaking filter.

In general the oscillator poles will differ from the bandpass poles, hence the filter center frequency f_c and the oscillator frequency f_o will differ. The oscillator transfer function, is:

$$H(z) = \frac{A(z)}{A(z) - B(z)}. \tag{5.17}$$

The higher the bandpass bandwidth, the higher the $B(z)$ coefficients will be, thus the difference between f_o and f_c will increase. To prevent this drift, the $H(z)$ can be designed first, and then the $G(z)$ evaluated accordingly. The proposed method consists in the design of a peaking Butterworth design centered at the desired f_o and with desired BW, yielding numerator and denominator $N(z)$, $D(z)$. The $G(z)$ will be then evaluated as:

$$A(z) = N(z); \tag{5.18}$$

$$B(z) = N(z) - D(z). \tag{5.19}$$

The Butterworth peaking design ensures the first coefficient of the numerator $n_0 = 1$, thus, from Eqns.5.17,5.18 the first coefficient of $B(z)$ will be always $b_0 = 0$, ensuring computability of the oscillator.

The oscillator must be causal and stable. Stability is always guaranteed, provided the initial filter design is stable. Proof of this can be gathered if considering the poles of $H(z)$, given by $D(z) = A(z) - B(z)$. The bandpass poles are always inside the unit circle by design, and only approach unity with $BW \to 0$, while its zeros can always be designed to be nonnegative. The roots of $D(z)$ are, thus, always inside the unit circle.

Nonlinear Oscillator Properties

In electronic oscillator design practice [97], a positive feedback oscillator makes use of a frequency independent amplifier, with nonlinear function $\beta(\cdot)$, typically a saturating nonlinearity, such as the hyperbolic tangent. The amplifier behaviour at small signals can be considered linear, i.e. a constant gain A_s, which saturates, i.e. reduces up to zero for increasingly large signals. It is frequent to study such a circuit as a quasi-linear system, i.e. study how the linear system transfer function is affected by the nonlinearity [98].

The nonlinearity, in fact, has an impact on the oscillator poles position in the complex plane. As a corollary, the more the signal amplitude increases, the more the oscillator behaviour deviates from the linear one. Specifically the f_o may shift from the desired value depending on the signal amplitude, i.e. the loop gain.

Let the variable gain imposed by $\beta(x)$ be considered constant for an approximately linear region, and β be that constant gain, the oscillator transfer function is

$$H_{nl}(z) = \frac{\beta A(z)}{A(z) - \beta B(z)}, \tag{5.20}$$

i.e. the poles frequency deviates from the linear case as a function of the gain β, which in turn depends on the input signal amplitude.

The roots of the denominator can be evaluated for increasing values of β. Figure 5.9 shows the skew in cents of semitone at increasing values of signal peak amplitude and increasing bandwidth. This frequency skew is not desirable, but for any input peak amplitude and BWs of up to 1/5 the frequency skew is lower or comparable to the *just-noticeable difference* [99], i.e. the minimum pitch interval that can be discriminated by the human ear. If the frequency estimation method is sufficiently reliable the bandwidth can be generally chosen large enough to compensate for the estimate tolerance.

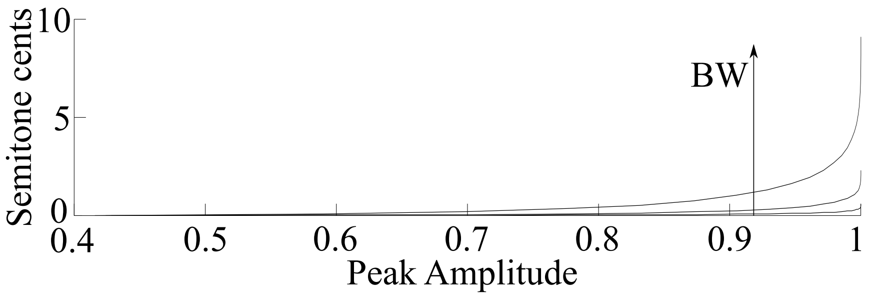


Figure 5.9: Nonlinear oscillator f_o skew in cents of semitone, with respect to a desired f_o of 500 Hz, function of the bandwidth of the $G(\omega)$ and the input signal peak amplitude. The three curves correspond to a bandwidth of 25 Hz, 50 Hz and 100 Hz.

Finally, the amplitude, rise time and slope can be set by the musician with the use of a gain G_p cascaded to the oscillator and its complimentary $1 - G_p$ cascaded to the dry signal path. The oscillator loop is closed only when an onset is triggered and a pitch detected, as the positive feedback oscillator can start oscillation with small input signals. The proposed oscillator is depicted in Figure 5.10.

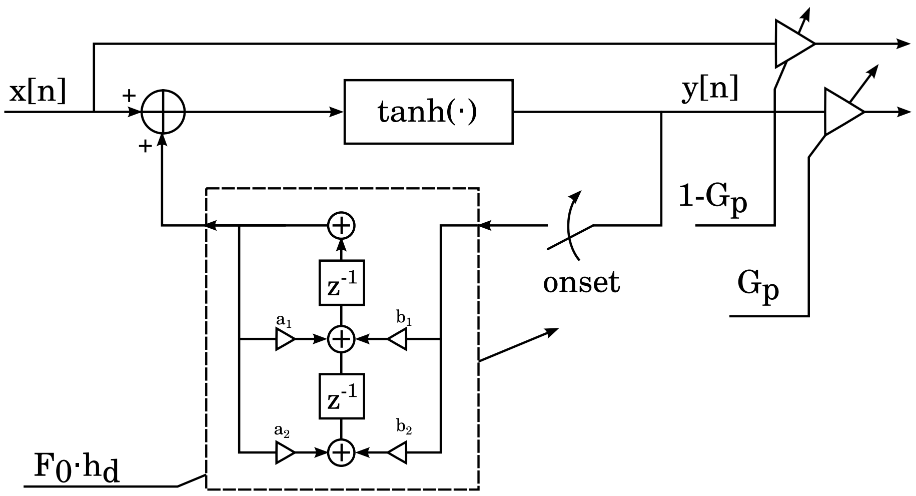


Figure 5.10: The proposed digital second-order oscillator. h_d is the desired partial for howling onset.

Since the $tanh(\cdot)$ function is odd, only odd harmonics will take place in the distorted signal. In musical applications, a distortion function should also generate even harmonics. In principle additional distortion components can be applied to the output signal $y[n]$. However, if the howling harmonic distortion is required to keep low, any memoryless waveshaping function can be used, provided it is bounded. Performed experiments show that an asymmetrical bounded function such as that from Doidic et al. ([100], Eq. (3)) poses no problems to stability and sustained oscillation. When distortion modelling blocks employing components with memory, are desired inside the loop further stability and frequency skew studies are required. One rather simple method for this is that of the describing function [101].

In a real scenario the oscillator needs note information to tune on the desired partial, thus, when the sound source is external a frequency estimator algorithm is needed to extract pitch information, often paired with a real-time onset detection algorithm. Given the low computational cost and latency of the oscillator, the choice of the pitch estimation algorithm is critical, given the real-time constraints on latency and computational cost. One approach often employed in the recent literature is an algorithm named SNAC (Specially Normalized Autocorrelation)[102]. A spectrogram obtained from simulations is shown in Figure 5.11. Virtual acoustic feedback can be triggered on a wide class of pitched instruments, including human voice. More research material and audio examples are available online [2].

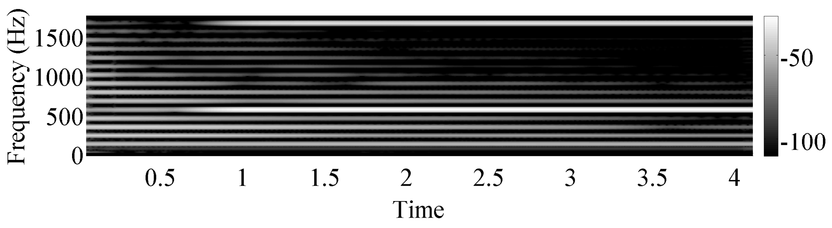


Figure 5.11: Spectrogram obtained from a simulated howling. A2 tone (110Hz) with howling at 550Hz (5th partial).

At the end of this section, it is of interest to note the requirements - system-wise - of this algorithm, which are similar to those of other algorithms for MAs in Section 5.3. First of all, the structure has a very low computational cost (a second order section and a nonlinearity, which can be implemented, e.g. as a look-up table with interpolation). Depending on the use case, however, it may be necessary to add a pitch estimation algorithm to track the current $F0$ and tune the digital oscillator accordingly. The computational cost of the pitch estimator is generally much higher than the digital oscillator itself, requires to work with with very low latency and process short buffers to keep steady as possible. Often, to obtain low tracking latency and low computational cost a trade off with the tracking accuracy must be found. To increase tracking accuracy some heuristics may be applied (e.g. filter out unexpected pitch variations) which, however often rely on branching or high computational complexity algorithms (e.g. sorting). The presence of a feedback, furthermore,

[2]a3lab.dii.univpm.it/projects/vaf

does not allow, in general, plain code parallelization as any output sample depends recursively on previous samples. Finally, for real-time implementation, one important requirement is that input-output latency is very low, meaning that processing must be executed on short buffers.

5.3 Towards a Comprehensive Musical Instrument Modelling

Relevant academic contributions of the candidate to the field of Musical Instrument Modelling are [103, 104, 105, 106, 107]. Most of these deal with stringed instrument modelling. Although this topic is covered by a large number of essays, books and papers a formalism for the string wave equation is provided for the sake of clarity, and the novel contributions of the candidate are discussed in the sections to follow.

The one-dimensional wave equation for an ideal vibrating string is:

$$K\frac{\partial^2 y}{\partial t^2} = \epsilon\frac{\partial^2 y}{\partial x^2} \tag{5.21}$$

where t is the time variable, x is the space variable, $y(t,x)$ is the string displacement function, K is the string tension, and ϵ is the mass density of the string. The solution of the wave equation can be written [108]:

$$y(x,t) = y^{+}(x-ct) + y^{-}(x+ct) \tag{5.22}$$

where $y^{+}(x-ct)$ and $y^{-}(x+ct)$ represent the progressive and regressive waves traveling with speed $c = \sqrt{K/\epsilon}$.

In the discrete-time domain the wave equation solution takes the following form:

$$y(n,m) = y^{+}(n,m) + y^{-}(n,m) \tag{5.23}$$

where the time and space variables n and m are discretized as shown below:

$$\begin{aligned} t &= nT \qquad for\ n = 0, 1, 2, ... \\ x &= mX \qquad for\ m = 0, 1, 2, ... \end{aligned} \tag{5.24}$$

where T is the time sampling period and X is the space sampling period, the two related by the following equation:

$$X = cT. \tag{5.25}$$

Such a solution of the ideal travelling wave equation is employed in Digital Waveguide synthesis together with other processing blocks that model additional features or departures from the ideal case. Introduced in the 1980s [109, 108], it has become very popular due the good trade-off between sound quality and low computational cost. For a review on DWG synthesis techniques refer to [110, 111].

Departing from the ideal string model of Eq. 5.21, stiff metal strings present dispersion and frequency dependent losses [103]. The wave equation in stiff strings with frequency dependent losses is characterized by the following [112]:

$$\frac{\partial^2 y}{\partial t^2} = c\frac{\partial^2 y}{\partial x^2} - \lambda\frac{\partial^4 y}{\partial x^4} - 2\sigma_0\frac{\partial y}{\partial t} + 2\sigma_1\frac{\partial y}{\partial t}\frac{\partial^2 y}{\partial x^2} \tag{5.26}$$

where λ is a stiffness coefficient, σ_0 the frequency-independent loss coefficient and σ_1 the frequency-dependent loss coefficient.

To directly solve such a string model, without resorting to filters or processing blocks, other synthesis methods are required, such as FDTD ones.

5.3.1 Mixed Physical Models

DWG methods for sound synthesis are based on a set of assumptions which enable to reduce the computational cost, yet allow to approximate the travelling wave equation solution. These assumptions are not always acceptable. FDTD methods, on the contrary, evaluate the wave equation at a discrete set of spatial points taking into consideration boundary conditions, physical parameters such as string density and thickness and removing some of the hypotheses of DWG methods. Boundary conditions also allow to model the exact behavior of physical variables in time, such as collision and contact.

First introduced in sound synthesis by Hiller and Ruiz [113, 114], FDTD methods find application not only for one dimensional problems but also for 2-D and 3-D meshes [112, 115].

A simple FDTD model, suited to solve the wave equation with losses Eq. 5.21, is the central differences scheme characterized by the following equation:

$$y_{k,n+1} = g_k^+ y_{k-1,n} + g_k^- y_{k+1,n} + a_k y_{k,n-1} \tag{5.27}$$

where g_k^+, g_k^- and a_k are loss parameters. Such a scheme has been referred to by the author as *first order FDTD scheme* (FOFS), to highlight the dependence of a sample from only the neighboring spatial samples one step forward and backward.

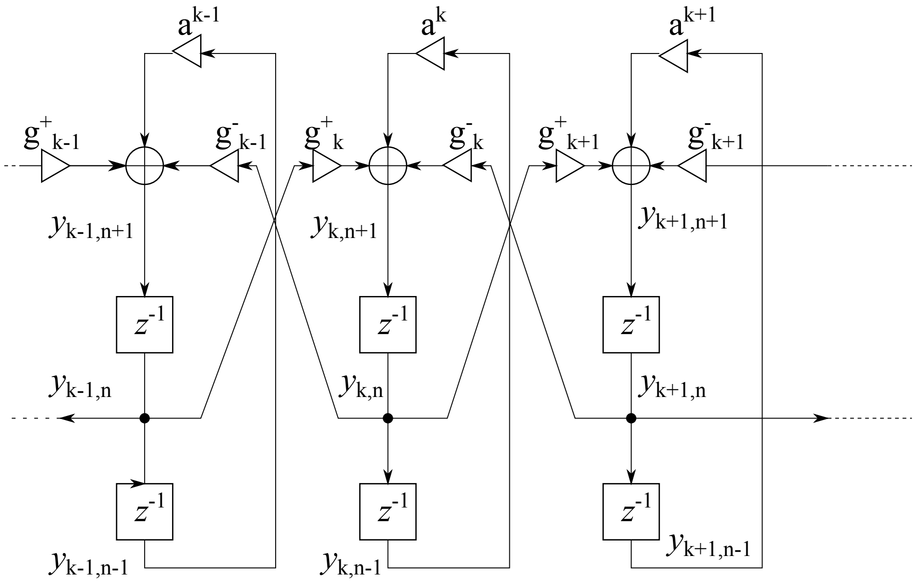


Figure 5.12: FOFS for a 1-dimensional wave equation (from [1])

Frequency independent losses can be added to the model by choosing values for the filter coefficients as in [116]:

$$g_k^+ = g_k^- = 1 - d_k \tag{5.28}$$

$$a_k = 2b_k d_k - 1 \tag{5.29}$$

where d_k and b_k are loss factor. For the stability of the waveguide $0 \leq d_k \leq 1$ and $0 \leq b_k \leq 1$.

As mentioned above, stiff metal strings follow Eq. 5.26 and need higher complexity schemes [113, 112]. A rather simple scheme showing good properties and solving the Partial Difference Equation 5.26 for low stiffness values is the following *second order FDTD scheme* (SOFS) [117]:

$$\begin{aligned} y_{k,n+1} = a_0 y_{k,n} + a_1(y_{k+1,n} + y_{k-1,n}) + \\ +a_2(y_{k+2,n} + y_{k-2,n}) + b_0 y_{k,n-1} + \\ +b_1(y_{k+1,n-1} + y_{k-1,n-1}) + \\ +J(f_h) \end{aligned} \tag{5.30}$$

where f_h is the force impressed on the string by a hammer. The time sample period k is the inverse of the sampling frequency, which also determines a grid spacing h. To ensure stability k and h must verify Equation 5.31:

$$N = floor(L/h) \tag{5.31}$$

where N is the length of the string in sample, L the length of the string in meters. It is very important to note that in a finite difference setting, h and k must be related by the numerical stability condition:

$$h \geq kc. \tag{5.32}$$

Loss coefficients in 5.30 can be evaluated according to [104]. A block scheme for the described SOFS is reported in Figure 5.13.

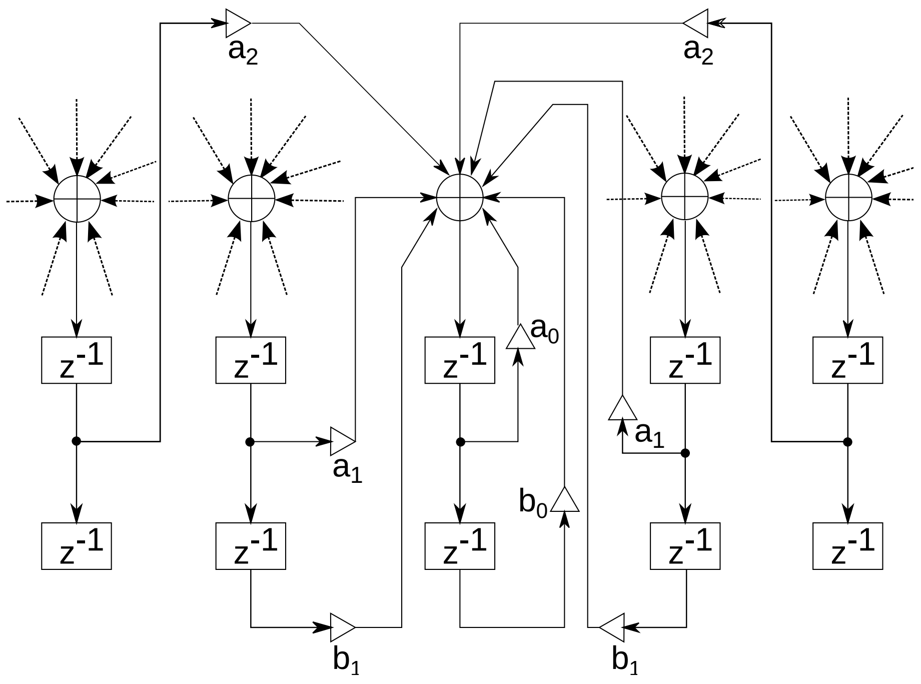


Figure 5.13: Block scheme of a SOFS model.

Mixed modeling can join the benefits of both the DWG and FDTD approaches, i.e. a good trade-off between computational cost (provided by modeling propagations with a DWG model) and sound quality (provided by modeling the string excitation mechanism with a FDTD model). A proof-of-concept of DWG and FDTD mixed modeling has been given in previous works, where a FOFS model have been considered [115, 118]. Metal strings, however are more accurately emulated with SOFS FDTD. A matching filter has been devised in [104] that is able to connect two strings parts of a string modelled with the two approaches. Let the first section of the string be emulated with FDTD and the second section in DWG, the last FDTD spatial point will be k and $k+1$ the first DWG spatial point. The right-going wave at the interface is described as:

$$y^{+}_{k+1,n} = y^{-}_{k+1,n-1} + g y_{k,n} \tag{5.33}$$

where y^{+} and y^{-} are the right-going and the left-going wave respectively, n is the temporal sample and g is the loss coefficient for the DWG model.

Similarly, the physical displacement wave on the 1-D FDTD side is related to the traveling waves inside the DWG section as follows:

$$\begin{aligned} y_{k,n+1} &= a_0 y_{k,n} + a_1(y_{k-1,n} + y_{k+1,n}) + \\ &+ b_1(y_{k-1,n-1} + y_{k+1,n-1}) + \\ &+ b_0 y_{k,n-1} + a_2(y_{k-2,n} + y_{k+2,n}) \end{aligned} \tag{5.34}$$

Equations 5.33 and 5.34 represent the interface conditions for perfect matching of traveling waves. The block scheme of the adaptor and the mixed model is represented in Figure 5.14.

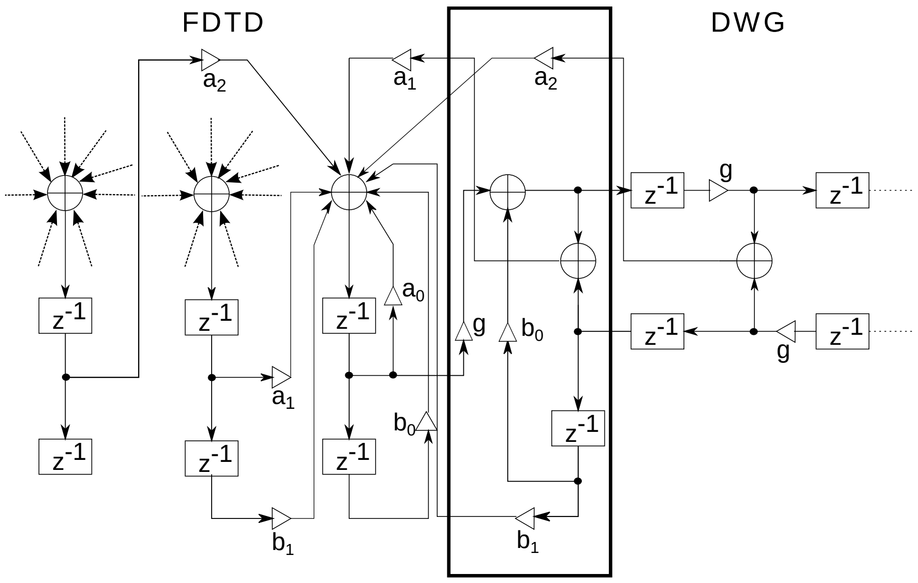
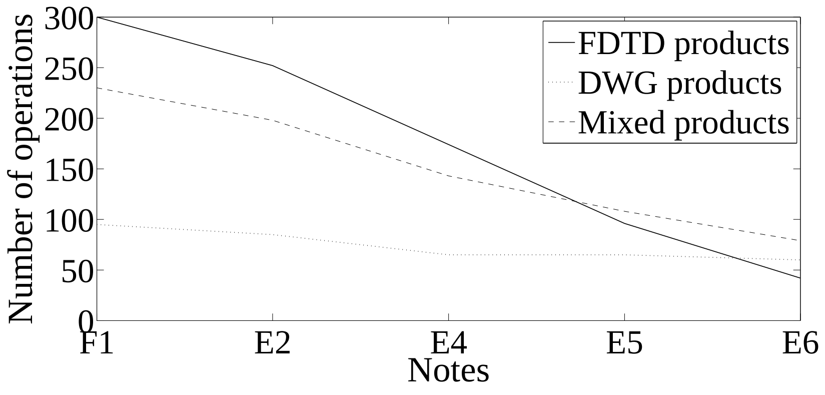
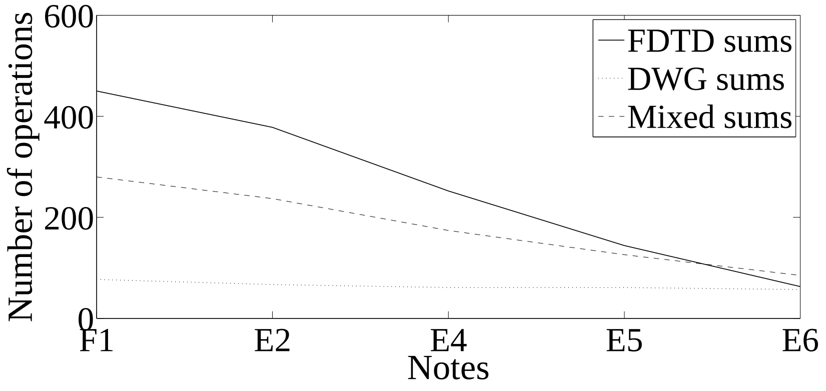


Figure 5.14: The mixed model consisting in the SOFS (left), the adaptor (center, highlighted) and the DWG (right).

Considering that each spatial sample in the SOFS (5.30) requires 6 products and 9 sums. For a string 14 cm long this leads to 42 products and 63 sums per sample. For a string of 93.4 cm this yields 300 products and 450 sums per sample. Compared, the computational cost of similar strings in a DWG model for the short string is 60 products and 57 sums per sample, while the long string requires 95 products and 77 sums per sample. In the mixed model the adaptor position is configurable, yielding to different computational costs. Increasing the FDTD string length requires to decrease the DWG string length and vice versa. Figure 5.15 compares the computational costs of the FDTD, DWG and mixed model for the strings F1, E2, E4, E5, E6 of a Hohner Clavinet with the DGW and FDTD emulating 50% of the string each.



(a)



(b)

Figure 5.15: The two figures represent in order the number of products and sums for the DWG, the FDTD and the mixed Clavinet models for the notes F1, E2, E4, E5, E6.

5.3.2 Expressive Physical Modelling: the Hohner Clavinet

Contributions to the field of sound synthesis may not only come from advancements in the basic building blocks of a system, but also from thorough analysis and parametrization of synthesis parameters. Several works from the author dealing with Hohner Clavinet instrument emulation are summarized in [103]. The results of analysis and advancements in the parametrization and synthesis of that instrument are reported. The modelling of such instrument has been covered, to date, by only few other works apart from those of the author [117].

Analysis of a stringed instrument covers many aspects, from build characteristics, to acoustic features and operating phenomena. For an electroacoustic instrument additional elements of interest are transducers and electronics. The Clavinet, for instance, shown in Figure 5.16, has metal strings transduced by magnetic pickups. It also includes an amplifier stage, with tone control and pickup switches. The pickups and amplifier are required because the Clavinet strings emit a very feeble sound and the metal keybed, differently from the piano soundboard does not efficiently radiate the sound. The pickups introduce several effects on the resulting sound [119], including linear filtering, nonlinearities [120] and comb filtering [110, 121]. Some of these effects have been studied in [107].

Prior to any investigation, a Clavinet D6 has been recorded in a quiet environment with professional equipment and a large database of tones has been created. Recordings include Clavinet tones for the whole keyboard range, with different pickup and switch settings. Automated analysis were run to obtain several acoustic features for each key.

Properties of the time-domain displacement wave and the excitation mechanism have been inferred by assuming the pickups to be time-differentiating devices [120]. The instruments, indeed is of electroacoustic nature and it is not possible to record the acoustic tone from the radiating string with a microphone without also picking the additional noise of the key mechanics which masks the attack of the tone. A very robust and reliable method to obtain the string displacement would rely on a laser transducer, which however was not available at the time the tests were conducted.

Another peculiar feature of the Clavinet with respect to more well-known instruments, is the way the *tangent* action strikes the string, separating it into two sections, in jargon the *speaking* part of the string (i.e. the one transduced by the pickup) and the *non-speaking* one, which is damped by yarn. When the

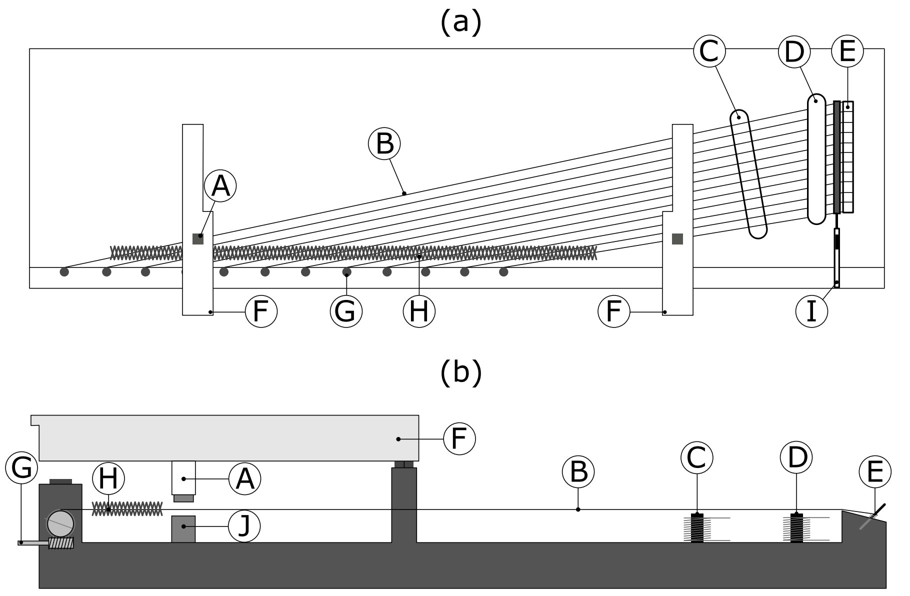


Figure 5.16: A schematic view of the Clavinet, (a) top; (b) side. Parts: A) tangent, B) string, C) center pickup, D) bridge pickup, E) tailpiece, F) key, G) tuning pin, H) yarn winding, I) mute bar slider and mechanism, and J) anvil.

key is released, the speaking and non-speaking parts of the string are unified, giving rise to a change in pitch. This transient is of short duration thanks to the yarn damping. Given the geometry of the instrument the pitch decrease after release is 3 semitones for the whole keyboard. A spectrogram of the tone before and after release is shown in Figure 5.17.

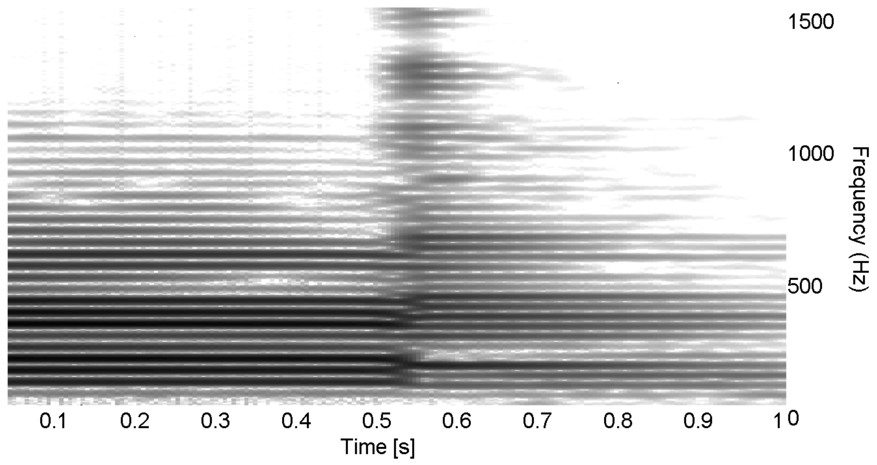


Figure 5.17: Spectrogram of the tone before and after key release. The key release instant is located at 0.5s

Another perceptually important feature of a string sound is the inharmonicity [2], due to the lightly dispersive character of wave propagation in strings. The strings are metal strings similar to those seen in electric guitars and the likes. Several methods exist for inharmonicity estimation [122]. In order to quantify the effect of string dispersion, the inharmonicity coefficient B must be estimated for the whole instrument range. Theoretically the exact pitch of the partials should be related only to the fundamental frequency and the B coefficient[123] by the following equation:

$$f_n = nf_0\sqrt{(1 + Bn^2)} \tag{5.35}$$

where f_n is the frequency of the nth partial and f_0 is the fundamental frequency. However, a different estimate method based on a mean value was chosen since empirical analysis of real tones shows a slight deviation between the measured partial frequency and the theoretical f_n, and thus a deviation value B_n can be calculated for each partial related to the fundamental frequency by the following:

$$B_n = \frac{f_n^2 - n^2 f_0^2}{n^4 f_0^2} \tag{5.36}$$

obtained by reworking Equation 5.35 and replacing the overall B with a separate B_n for every nth partial. For practical use a number of B_n values measured from the same tone are combined to obtain an estimate of the overall inharmonicity. A way to obtain this estimate is to use a criterion based on the loudness of the first N partials (excluding the fundamental frequency), as described below.

The partial frequencies are evaluated by use of a high-resolution FFT (Fast Fourier Transform) on a small segment of the recorded tone. The FFT coefficients are interpolated to obtain a more precise location of partial peaks at low frequencies. The peaks are automatically retrieved by a maximum finding algorithm at the neighborhood of the expected partial locations for the first N partials and their magnitudes in dB are also measured. The fundamental frequency and its magnitude are estimated as well. The B_n coefficients are estimated for each of the N partials using the measured value for f_0 to take a possible slight detuning into account. For a perceptually motivated B estimate the B_n estimated values are averaged with a weighting according to their relative amplitude.

The B coefficient has been estimated for eight Clavinet tones spanning the whole key range by evaluating the B_n coefficients for $N = 6$, i.e. using all the partials from the 2^{nd} to the 7^{th}. Linear interpolation has been used for the remaining keys. The estimate of the B coefficient for the whole keyboard is shown in Figure 5.18 and plotted against inharmonicity audibility thresholds as reported in [2]. From this comparison it is clear that inharmonicity in the low keyboard range exceeds the audibility threshold and its confidence curve, meaning that its effect should be clearly audible by any average listener. For high notes, the inharmonicity crosses this threshold, making it unnoticeable to the average listener, and hence may be excluded from the computational model.

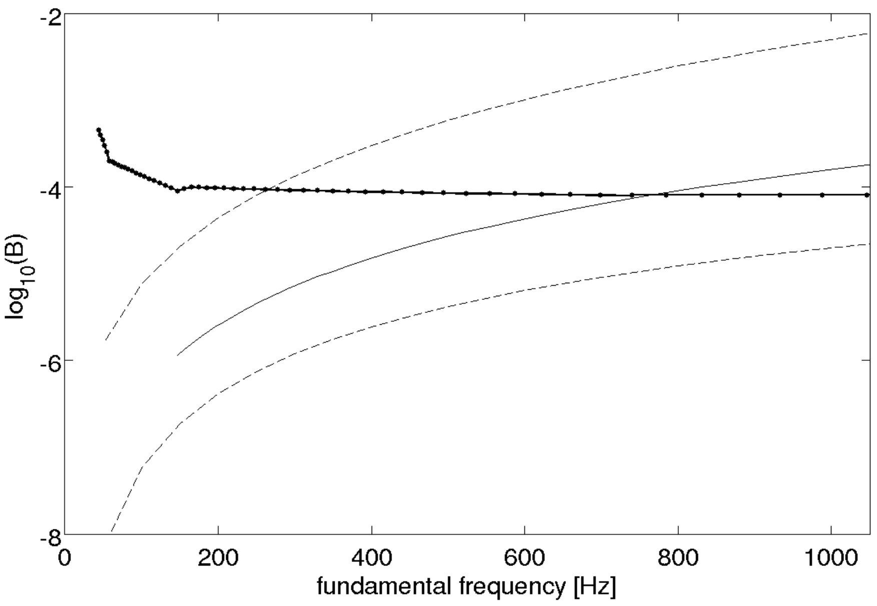


Figure 5.18: Estimated inharmonicity coefficients (bold solid lines with dots) for the whole Clavinet keyboard range against audibility threshold (solid line) and confidence bounds (dotted line) evaluated in [2]. The discontinuity between the 23rd and the 24th keys is noticeable at approximately 150 Hz.

* * *

The computational model for the Clavinet devised in [103] is rather complex, as many building blocks have been added to the simple DWG string model introduced in Section 5.3. An high-level SFG is depicted in Figure 5.19.

The DWG model consists of the delay line, which is split into two sections ($z^{-(L_S-R)}$ and z^{-R}) in order to add a comb filter embedded in the delay line, called *ripple filter*. The DWG loop includes the one-pole loss filter [124] $H_{loss}(z)$ which adds frequency dependent damping, and the dispersion filter $H_d(z)$ which adds the inharmonicity characteristic to metal strings. The fractional delay filter $F(z)$ accounts for the fractional part of L_S which cannot be reproduced

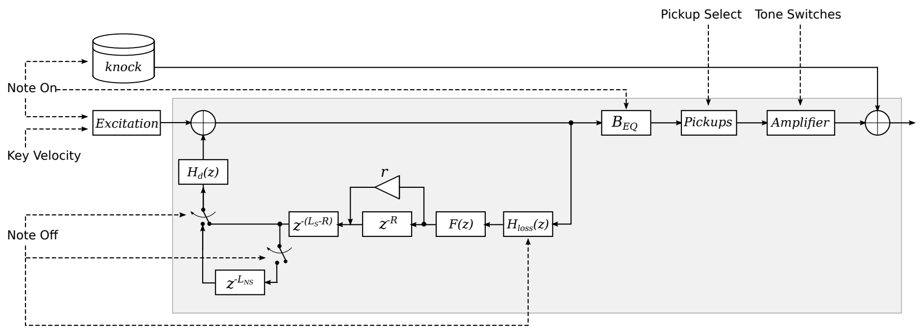


Figure 5.19: Signal flow for the complete Clavinet model.

by the delay line. While the Clavinet pitch during sustain is very stable, and thus there is no need for changing the delay length, a secondary delay line, representing the non-speaking part of the string, is needed to model the pitch drop at release. This delay line $z^{-L_{NS}}$ is connected to the DWG loop at release time, to model the key release mechanism. To excite the DWG loop there is the excitation generator block, named *Excitation*, which makes use of an algorithm described in [105] to generate the excitation signal related to key velocity and data on the tangent to string distance. This is triggered just once at attack time.

Several blocks are cascaded in the DWG loop. Since the DWG model described above only models the longitudinal string vibration mode, a *beating equalizer* [125] (B_{EQ}), composed of a cascade of selective bandpass filters with modulated gain, emulates the beating of the partials and completes the string model. Then, the *Pickup* block emulates the effect of pickups, while the *Amplifier* emulates the amplifier frequency response, including the effect of the tone switches. The study of the pickup nonlinearity is of interest and has been reported in [107].

Coil pickups, such as those used in guitars, have been addressed first in [126] and [120]. Paiva et al. conducted thorough studies on the magnetic nonlinearity of the pickup and how to model it in [119]. Specifically, it adds a combination of linear and nonlinear effects. The effect of their position is that of linear comb-filtering due to the reflection of the signal at the string termi-

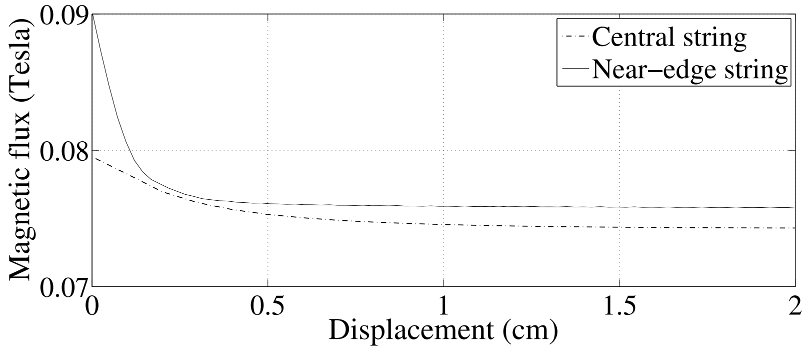


Figure 5.20: Simulation of the magnetic flux against vertical displacement: magnetic flux variation against vertical displacement for a string passing over the pickup center (dashed line) and vertical displacement for a string passing over the edge of the pickup (solid line).

nation. Pickups also have their own frequency response given by their electric impedance and the input impedance they are connected to [127]. The relation between the string displacement and the voltage generated by the pickup induction mechanism is nonlinear due to factors such as the nonlinear decay law of the magnetic dipole field. Nonlinearities in the displacement to voltage ratio have been evaluated by means of a software simulation in Vizimag, a commercial electromagnetic simulator. Simulations have been carried out for different string gauges, string to pickup distance and horizontal position of the string with respect to the pickup. Vibration in the horizontal and vertical polarizations has been measured separately, resulting in a negligible voltage generated by the horizontal displacement (25 dB lower than the vertical displacement). The string oscillation was 1mm peak-to-peak wide, which is the maximum measured oscillation amplitude. Simulations show that the magnetic flux variation in response to vertical displacement has a negative exponential shape (Figure 5.20), in accord to previous works [126, 120, 119]. Finally the frequency response of the displacement to voltage ratio is that of a perfect derivative.

To conclude the overview of the Clavinet model, a *knocking* sample of the soundboard hit by the tangent is triggered at each note attack to reproduce that feature of the Clavinet tone. This is similar to what has been done for the emulation of the clavichord [128], an instrument that shows some similarities with the Clavinet.

The theoretical computational cost of the complete model can be estimated for the worst case conditions and is reported in Table 5.1. The worst case conditions occur for the lowest tone (F1), which needs the longest delay line and the highest order for the dispersion filter. The latter depends on the estimate of the B coefficients made during the analysis phase and the parameters used to design the filter. With the current data the maximum order of the dispersion filter is eight.

Table 5.1: Computational cost of the Clavinet model per sample per string.

Block	Multiplications	Additions
Fractional Delay Filter	2	2
Dispersion filter	4×5	4×4
Loss filter	2	1
Ripple filter	1	1
Soundboard knock w/ LPF	3	2
Beating equalizers	2×7	2×7
Pickups	9	5
Amplifier and Tone switches	21	15
Total per string	82	64
Total FLOPS per string	6.4 MFLOPS at 44100 Hz	

How computational cost translates to Real Time factor or CPU load on an embedded platform is discussed in Section 2.3.

5.3.3 Parametrization and Subjective Evaluation

Most of the computational acoustics field academic works cover DSP techniques and their advancements. In order for research to sustain the practical realization of physical modelling products other topics need be addressed properly, in the author's opinion. Subjective evaluation of synthesis techniques, architectural design and low-level optimization, design and presentation of parameter ontology and sound organization and most of all digital platforms for digital music instruments. As to the last point, the author sought a partial answer along the lines of this essay and more is discussed in Chapter 6. However, the other topics require much attention. Low-level optimization is a good topic for research. Although to the "consumed" DSP programmer it may seem a matter of practice, nothing related to compiling and optimizing is out of the scope of research. Just the opposite: microarchitectures and compilers evolved thanks to a great deal of research effort, and programmer often lack a scientific approach to code optimization. Not to mention the illiteracy of some researchers. In the process of providing comments to a physical modelling paper, a reviewer harshly stated

"do people still care about memory occupation nowadays?"

Of course they do. Implementing full-polyphony DWG models on an embedded DSP, with 32KB level 1 cache and 64KB level 2 cache, considering that each voice may require more than 1KB to be accessed at each iteration may not be trivial and may require some optimization or careful design. Specifically, the main issue, to allow for a full-polyphony, is having all the relevant data at hand, i.e. in L1 cache, including filter coefficients and the delay line content. Moving data back and forth between different cache levels or even external storage (this is still the case with sampling synthesis too) can lead to wasting clock cycles without data processing, and badly affects polyphony. Parametrization is a rarely explored topic too, at least in applications related to musical instruments emulation, which do not gather the interest of very active communities such as the computer music one. In essence, this chapter provides some more introspection related to subjective listening tests and industrial implementation of physical modelling synthesis, without aspiring to have the last word on any of these.

The effort of providing physical modelling techniques to the end user spans several project cycles and professional competences. A rather general abstraction of such a process is provided in Figure 5.21. The diagram lumps the development of the computational model and its parametrization in a few steps and highlights how it intersects some of the hardware and software development steps. All cycles of software and hardware engineering remain unchanged with respect to other industrial applications.

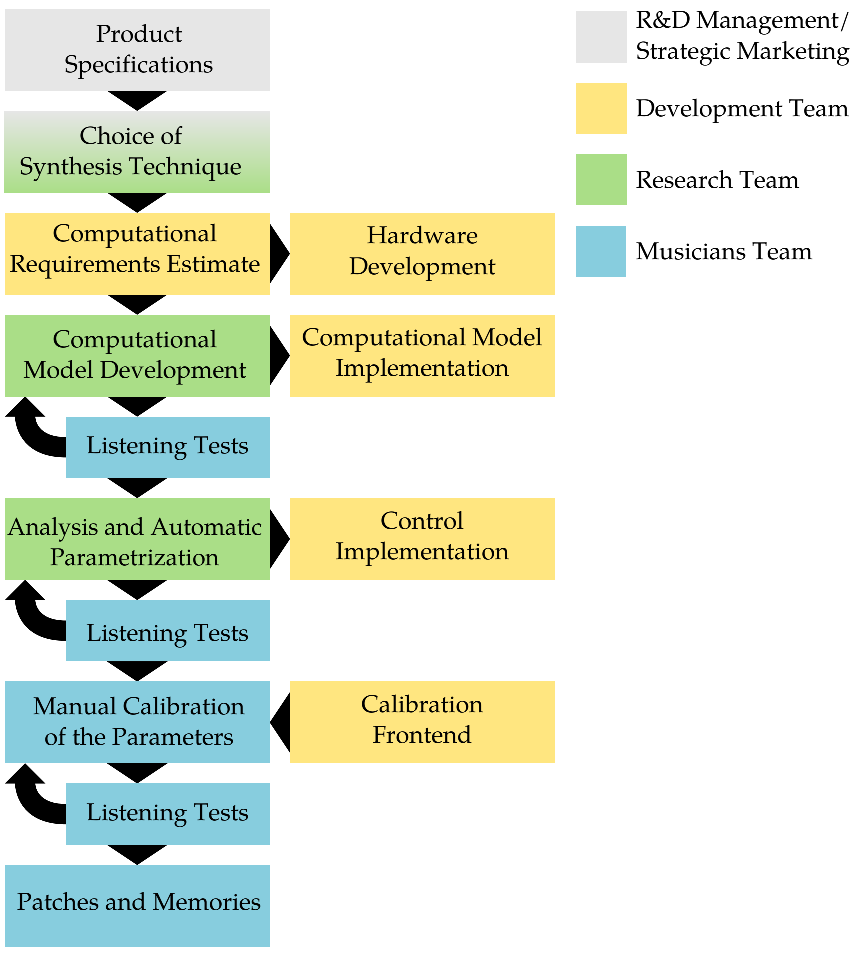

Product Specifications
Choice of Synthesis Technique
Computational Requirements Estimate
Hardware Development
Computational Model Development
Computational Model Implementation
Listening Tests
Analysis and Automatic Parametrization
Control Implementation
Listening Tests
Manual Calibration of the Parameters
Calibration Frontend
Listening Tests
Patches and Memories
R&D Management/ Strategic Marketing
Development Team
Research Team
Musicians Team

Figure 5.21: A schematic diagram representing the typical development cycles of a computational model for sound synthesis.

A brief list of the development phases follows:

- Product specifications are agreed and between the strategic marketing team, the company management and the R&D (Research and Development) management.
- From the constraints and the goals reported by the product specifications, a solution is agreed with the research team about the synthesis technique to use.
- Given a specific set of features and the synthesis technique(s) the computational requirements are estimated by the development team that also starts to develop the hardware following the computational requirements.
- The model starts being developed by the research team. This procedure draws from a thorough analysis of the physical behavior of the real instrument or prior academic knowledge. The computational model is then translated to an algorithm, passed on to the development team for implementation on the target platform. To assess the timbre quality of the model it may be developed in a prototyping platform such as Matlab, C/C++, or directly on the target platform. Listening tests and audio analysis are of the highest importance to assess whether the computational model devised by the research team is able to reproduce the salient features of the real instrument, at least (at this early moment) from a qualitative point of view. Several cycles of development of the computational model and listening tests may be required to refine it. The musicians team helps the researchers assess the quality.
- After the computational model has been agreed upon, it must be populated with parameters and coefficients. Some of them can be computed or measured (e.g. physics constants), some other must be estimated, e.g. from acoustic recordings, feature extraction and data fitting, while some other must be computed in real-time according to some law depending on user input. Results are evaluated through listening tests by the musicians team. Theoretically speaking, scientific methods should yield very

good parameter estimation if the hypotheses underlying physics modelling are correct. However, given the number of simplifying hypothesis that are generally applied to current computationally-feasible synthesis techniques, a data fit from acoustic recordings may not yield a good estimate of the model parameters. Although results may not sound yet sufficiently convincing a reason not to skip this step is the complexity of the parameter space. The number of parameters and coefficients is often very large, depending on the synthesis technique. The parameters vary in pitch and dynamic, and, thus a first approximation aided by the computer is necessary.

- The parameters calibration is conducted by the musicians to fine-tune the automatic parameters estimate in a computer-assisted fashion. In this process they are helped by some software front-end for calibration. The musicians are often not technical experts, thus, some complexity may be hidden behind the calibration front-end to help them focus on their (often trial and error-based) refinement process.
- Once the computational model is defined, calibrated and implemented in real-time, a last task for the musicians is to create patches and presets employing the computational model. This step requires creating different variations of the model according to different tastes and musical genres by slight modification of the parameters, supplementary effects, control or performance modifications (e.g. making a synth preset monophonic, or relating a control wheel to a certain parameter).

An evaluation and assessment is performed at the end of each cycle, which can be iterated several times, possibly leading to a refinement of the model, of the parameters, of the control strategy, or to an improvement or optimization of the implementation. As already stated in Section 5.2, algorithms for real-time MAs can often have very low computational cost, but are non trivial to implement. In the case of real-time synthesis, the synthesis engine must be able to allocate a large number of *voices*. While for some synthesis engine a single voice may be computationally inexpensive, the contemporary allocation of a large number of voices with the same computational cost sums up at least linearly in terms of system load (unless some factorization technique can be employed to optimize). Often, however, some system resources are wasted due to computational overhead, sub-optimal memory bus management, and such, resulting in a higher than linear increase of system load with an increasing number of voices. It must be noted, furthermore, that, differently to the assumptions made in papers and tutorials written by unexperienced developers, the polyphony required by a keyboard instrument must be much larger than the number of fingers human have, due to:

- the sustain pedal retaining many more voices active,
- voices that are being released but still are not silent,
- layers of voices (multitimbral instruments, unison, etc.).

Although trivial this may be to the accustomed reader, much low quality literature does not take such basics into account.

In [129], a brief introduction to physics-based synthesis in a commercial digital piano is provided. The instrument[3] implements different algorithms to emulate acoustic pianos, electric pianos and chromatic percussions. The SP hardware has been designed to fulfill the worst case computing requirements, which are those of an acoustic piano model featuring emulation of the soundboard and resonant strings with full polyphony. The piano features a mesh of 6 interconnected DSPs of the TI OMAPL137 family, featuring a C674x DSP and a ARM9 microcontroller unit. Another single ARM9 core coordinates the workload, handles peripherals, I/O, the user interface and transmits synthesis parameters to the 6 OMAP chips. These share the workload. The voices are computed in parallel and distributed among 4 C674x, while the soundboard and the additional effects are computed by the last 2 C674x respectively. In terms of both parallelization and sound quality, modal synthesis provides a good approach. While more computationally expensive compared to digital waveguides, this technique allows for high flexibility of calibration and permits

[3]http://physispiano.com/

a more accurate modeling of the nonlinear longitudinal string vibrations which characterize the timbre in the low acoustic piano register. Additionally, as already mentioned, it is particularly well suited to code parallelization.

The acoustic piano algorithm is based on the decomposition of the string displacement $y(x, t)$ into its orthogonal normal modes [130]

$$y(x,t) = \sum_{n=1}^{N} y_n(t) \sin\left(\frac{n\pi x}{L}\right), \tag{5.37}$$

where $y_n(t)$ are the instantaneous amplitudes of the modes, or *partials*. Substituting Eq. 5.37 into the partial differential equation of the string (Eq. 5.26) motion results in an ordinary second-order differential equation for each partial, and thus the impulse response of the string becomes a sum of exponentially decaying sinusoidal functions.

After discretization, the input-output relation of the string block is realized as a parallel connection of N second-order all-pole resonators:

$$\begin{aligned} F_{\text{string}}(z) &= H_{\text{string}}(z)\, F_{\text{h}}(z) \\ H_{\text{string}}(z) &= \sum_{k=1}^{N} W_{\text{in},k}\, H_{\text{mode},k}(z)\, W_{\text{out},k} \\ H_{\text{mode},k} &= \frac{b_{1,k} z^{-1}}{1 + a_{1,k} z^{-1} + a_{2,k} z^{-2}}, \end{aligned} \tag{5.38}$$

where $F_{\text{string}}(z)$ is the transverse force at the bridge, $F_{\text{h}}(z)$ is the force coming from the hammer and $H_{\text{mode},k}(z)$ are the transfer functions of the normal modes. The conversion between the physical variables (i.e. force, displacement) and the modal variables is regulated by a set of input and output weights $W_{\text{in},k}$, $W_{\text{out},k}$.

Sound synthesis parameters are organized in a hierarchical way. The user is presented with a short set of Macro Parameters (hammer, tuning, string type, resonance, size) which control a larger set of 15 Micro Parameters (e.g. hit position, string stiffness). These in turn can modify the internal parameters used by the sound designers, which are around 100 and in some cases can be controlled note-by-note. Finally, the last layer of transformation generates the approximately 300'000 synthesis microparameters which are used by the real-time engine. With such a large set of parameters, automatic parameter estimation is necessary, as reported above.

Modal synthesis is particularly useful in this case, as duplex resonators, sympathetic string resonance and soundboard can all be modelled by sets of oscillators. Beating can also be modelled by slightly detuned oscillators, differently from what was done in the DWG synthesis of the Clavinet, where beating was implemented by modulated Regalia band-pass filters (although roughly speaking the computational cost can be considered equivalent, the modal synthesis paradigm increases code scalability and parallelization).

The acoustic piano model, is a good example of a development cycle evolution. A previous model documented in [131], included a feedback hammer model to excite the resonators. However faithful in sound and close to the physics of the piano action, after informal evaluation a feedforward approximation of the original hammer model has been preferred for its improved usability and a more deterministic response to touch dynamics.

* * *

The development flow diagram in Figure 5.21 includes several iterations including listening tests. Every musical instrument manufacturer has its own procedures and experts to conduct those. However, as the advances in the fields of acoustics and music computing enable the creation of close-to-complete models for the emulation of real musical instruments it is worth asking whether a standard and broadly accepted subjective evaluation method can be adopted by researchers and the industry for comparison and communication. One of the current trends in research is very close to that adopted by instruments manufacturers: allowing the interested user or reader to listen and judge by himself. Following this trend, the author published a number of *compagnion pages* to published articles dealing with synthesis. It is the case of the aforementioned

Clavinet emulation[4] [5] [6] [7], or the VAF[8]. Together with audio samples other materials can be supplied to the reader, e.g. software or data tables to test, implement and modify the algorithm.

Other researchers, along their works, reported results from listening tests of many sorts, to evaluate the level of realism of their emulation algorithms. Some of these works [132, 133] are in the field of sampling technology, while others [134, 135, 136] are in the field of digital waveguide synthesis. Except from these works, there has been no effort in the academic literature to propose a uniform procedure for the evaluation of an emulation synthesis algorithm results. As the works in the field grow in number, a standard approach to assess the timbre quality of synthesis algorithms is needed, to allow also a comparison between different works.

Some more common audio fields have standard procedures to assess the quality of a work, be it a device, an algorithm, an ambience or the perceived quality of a product related to another one, in the field of sound synthesis a standard lacks. In [106], the author proposed a simple method based on subjective listening tests called R-S. The subjects need to discern real from synthesized tones of a certain instrument. If the subjects are not able to distinguish between real and synthesized tone their *Accuracy* [137] tends towards the perfect guessing value, i.e. 50%. Other ways to present results are proposed, such as the F-measure (related to Accuracy), or the ROC (Receiver Operating Characteristic [138]), which can be used to evaluate results against different synthesis parameters. To the researcher or the developer, it is of high importance to evaluate the ability of an algorithm to be convincing over several dimensions, e.g. pitch, dynamic, etc. A meaningful way to evaluate results is presenting these in a multidimensional plot, as shown in Figure 5.22. A 3D plot improves on readability of the results, and quickly shows potentially weak areas of the computational model and the employed parametrization. In the example, the surface shows that the rate of false negatives (synthetic tones classified as recorded tones)

[4]http://research.spa.aalto.fi/publications/papers/icmc11-clavinet/
[5]http://a3lab.dii.univpm.it/projects/jasp-clavinet
[6]http://a3lab.dii.univpm.it/projects/fdtd-adaptor
[7]http://research.spa.aalto.fi/publications/papers/dafx12-pickup/
[8]http://a3lab.dii.univpm.it/projects/vaf

increases for the higher tones and for low and high dynamics, suggesting e.g. that the model succeeds better in emulating the higher tones and the extremes of the dynamic range.

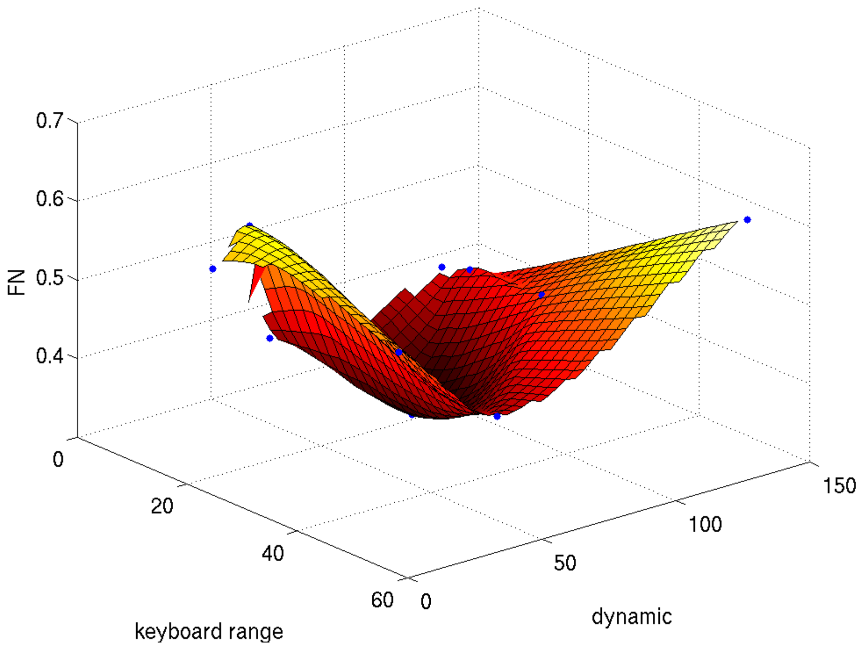


Figure 5.22: 3D plot and fitting surface of the rate of false negatives from a listening test; the plot allows for a quick assessment on the salient features of the test with respect to two parameters, keyboard range and dynamic.

While conducting the experiments, it was found that the more questionable decisions are not related to the metric but to the methodology. The requirements of the subjects are critical, as different categories of users obtained very different results. One assumption often done in some of the aforementioned academic works, is that subjects with a musical training are sufficient for the tests. Musical training is generally meant as some years of experience in playing an instrument, or being enrolled in music classes at high school or higher degree. According the experimental experience, I suggest the subjects must have years of practice of the instrument to be evaluated, and, if professional musicians, they must have also technical experience, in order to be able to discern artifacts and provide an informed feedback to the developers. In any case the results should not be averaged over different classes of subjects. Finally, it is suggested that the test interface, the evaluation procedure, the selection of sound samples and other aural or visual cues, may have effects on the evaluation procedure. In a comparison between a high-quality recorded tone and a synthesized tone which has been added with background noise, the subject may be biased and believe the synthesized tone to be real due to the characteristic background noise typical of audio recordings.

At the moment a very interesting outlook on conductive informed subjective evaluation is [139].

Chapter 6

Future perspectives in Music Computing and Networked Music Performance

The scope of this PhD essay is not to obtain a superior technical achievement in one specific area, or provide a complete overview of a certain computing topic, but to provide an overview of different topics related to wireless embedded musical applications, put the acquired knowledge under test, report technical achievements and development experience, evaluate the outcome to learn and highlight issues, problems yet to solve and possible future outcomes. This Chapter is, thus, intended as a relevant part of the essay, and not just a graceful way to conclude before the final *The End* screen.

6.1 "What We Must": Future Works

From the beginning of the PhD studies to the end a great deal of research and development has been conducted. As usual, however, it is a *little-o* of what was planned or desired, especially for what concerns WeMUST as a whole. As of WeMUST-OS v.0.3 and we-tools RC2, a large number of features are still missing or have to be fixed. Some items worth pointing out will be illustrated.

No effort has been done yet to evaluate the overhead of the control protocol in wemust-netdbg or the SABy protocol. As those protocols are extremely simple it was not of interest, yet they might be optimized in the near future. The SABy protocol has been designed taking pace from UPnP and made simpler. It is a custom protocol and it serves well the intended goals but it is worth asking whether the goals originally specified are sufficient, and whether it will gain some spread in other platforms or projects. As to the first question, it probably needs some cycles of evaluation. The original SABy specifications require four fields: APP, SR, NAME and TAG. It was targeted to a low-level C implementation using Puredata and UNIX BSD sockets. While developing we-tools, other issues did arise, e.g. providing also a period size together with the sample rate and the number of channels. The period has been added to the SR item in form of SR:<rate>:<period>. Another question is whether other features of a node must be included in SABy, such as the IP address and port.

There are other attempts in the music field to provide a protocol for music networking. A few people have proposed to implement further features in OSC for service discovery [140], networking and sharing musical objects [141] and queries[1]. As long as these efforts are sparse and no clear perspective on the future of OSC is available it is not suggested to spend more effort on a device discovery and negotiation protocol.

Besides the networking/interoperability layer, the actual software providing for connectivity has some limits at the moment. Jacktrip includes resampling only in the custom version described in the previous chapters, and it will take time until the changes get to the trunk and the binaries will be accessible to Linux or MacOsX users. In the meanwhile a promising software, zita-njbridge from Fons Adriaensen [2], has been released. This software includes a resampling algorithm slightly different from the one implemented in Jacktrip and supports IPv6. Its computational cost, being based on the same resampling library is comparable to that of Jacktrip with resampler but it performs better (at least on the BBxM) due to a lower number of calls and SW layers, not relying on the QT library. Zita-njbridge is not bidirectional, and to adapt WeMUST to employ it requires reworking in order to ensure a bidirectional connection. It must

[1]Schmeder-Wright, MrRay, OSNIP, Minuit, QLAB, libmapper, oscit, to name a few

[2]http://kokkinizita.linuxaudio.org/linuxaudio/index.html

be noted, however that some bandwidth is spared when just a unidirectional connection is required, allowing for more flexibility.

WeMUST-OS only supports the BBxM as embedded platform. As the time passes by, new more performing development boards will come out, which will draw the attention of the users. Some of these platforms should be supported by WeMUST-OS, in order for more users to experiment with WeMUST.

Finally wireless communication has been tested for a small subset of use cases. Much more data must be collected in order have a last word on the real feasibility of 802.11 for wireless NMP. Moreover, since AES67 [49] seems to be the future reference for digital audio networking, a 802.11 link or multiple nodes scenario should be tested to conclude on the feasibility of wireless NMP on 802.11 networks. It is probably needed to configure the network properly at the physical level (e.g. data rate) and the medium access level to avoid collisions.

6.2 Looking Forward: Trends and Technologies

6.2.1 DSP Architectures

In the last three or four years the mobile revolution has shaped the consumer market in many regards. It suffices to say that, at least in Italy, many TV commercials convey slogans and key information on the screen of a tablet, or feature touch-screen gestures to scroll between items. Besides the paradox of watching a screen in a screen, and the fact that this proves not only *trendy*, but even *attractive* [3] , it also produces money. The mobile market has an impact on the aesthetics of advertisement, and perhaps very subtly also on the minds of the commercial representatives of many silicon companies. As a matter of fact, devices that in the past were *industrial-grade*, such as a home thermostat or a key lock, are now getting *smart* (or more precisely *smartphone-ized*), i.e. of a shiny design, social, and interconnected[4]. Or, another example, products

[3]in an erotic way as the Italian philosopher Umberto Galimberti put it in "I vizi capitali e i nuovi vizi", Feltrinelli, 2007.
[4]see e.g. https://nest.com/ or http://august.com/

that are based and revolve around a hardware component, such as an electric energy monitor, base all their website advertisement on a smartphone, the cloud services and the fancy graphical aspects of its interface, leaving just one picture of the actual device and without even mentioning hardware technical specifications [5]. Economic cars, that were made to transport people (Fordism, anyone?) are nowadays made to entertain people. And many more example could be devised. It remains unknown whether this social trend is emerging because technical requirements are met at lower costs and there is an extra available to invest on the fun part, or either because quality is such an unknown to consumers that it does not really matter whether technical requirements are met.

The impact of the mobile market does not only reflect on society and economy, but also on the silicon industry. In the last few years, new ICT products emerged for the mobile market, which are adopted by millions of users. This means that by adopting these components, some advantage can be gathered... provided one has a sufficient reputation in the market to obtain a quote. Regarding embedded processors, some SoCs are nowadays manufactured which are apt to signal processing and are meant for the multimedia and the mobile market. The question is: are these SoC the future platforms for DSP in MAs? Or even more bold: are MAs migrating from embedded hardware to mobile platforms?

There are signs in the market of a shift from discrete DSPs to integrated SoCs based on RISC processors powered by floating-point arithmetic units or co-processors for audio, video, multimedia, wireless baseband DSP and such. This change is triggered by the impact of mobile market pressure on the silicon industry to increase performances and reduce costs. The same pressure brought to a standardization of the architectures and the operating systems, to facilitate reuse of HW and SW IPs. The reuse of SW and code is of interest, as PC software for MA could be ported to embedded instruments, simplifying the development, as with the piano model in [131] implemented as x86 code and later ported to TI OMAP DSP [129].

[5]http://www.smappee.com/

Mobile SoCs however feature a set of peripherals and cores that are not of interest for most MAs, such as LTE, NFC connectivity, GPS, security accelerators, graphic accelerators, camera input, etc. All these units increase the cost, the silicon area and are ever improved, meaning that the manufacturer will charge for the cost of their development.

While digital instruments require an Operating System, a nice GUI and many digital I/O, including - as in WeMUST - wireless connectivity, it is questionable whether shifting all the DSP to a general purpose unit with extra unused features represents a valid choice and whether computational requirements are met just by these processing cores, while in the past they were barely met by dedicated DSPs. It must be noted that, on the other hand, ARM IPs can be throttled up over the 1-GHz clock speed, while discrete DSP chips of same price[6] are still running at around 300-400MHz in many cases. A SIMD engine is available on all the current Cortex-A cores, which allow parallelism, and a floating point instructions are included to speed-up floating point calculus. Benchmarks reported in Section 2.3 suggests that ARM platforms could be viable for challenging DSP tasks, provided that the solution is robust enough to real-time critical tasks.

Other concerns for the use of ARM SoCs in the music industry are not technical in nature. First of all: support. SoCs targeted at the mobile world are intended to live short, as the handsets that host them, thus, support is guaranteed for very short periods, generally 2 years, while musical instruments can and should last for decades and be able to be serviced or repaired years later. The other concern regards pricing and availability. As the mobile ARM SoC market is at the moment dominated by Qualcomm[7], and other IP buyers such as Samsung, Broadcom, Apple, Marvel, Allwinner, Mediatek, etc., that

[6]as a reference: as of January 2015 a previous generation 450 MHz C674x chip is officially quoted $13 to $17 per 1,000 units depending on the specific product and package, while the least performing 1250 MHz C667x is quoted $79 to $118 per 1,000 units. Mobile ARM-based SoC can cost around $20 for a 1.5GHz Qualcomm Snapdragon 600 APQ8064T Quad-Core to $30 for a 1.6GHz Samsung Exynos 5 Octa 5410 (according to an iSuppli research conducted in March 2013, https://technology.ihs.com/430692/samsung-galaxy-s4-carries-236-bill-of-materials-ihs-isuppli-virtual-teardown-reveals)

[7]in H1 2013 it was ramping over to surpass Imagination Technologies PowerVR IP http://gfxspeak.com/2013/09/25/qualcomm-leads-mobile-gpu-market/

sell only for large numbers, the availability of high-performance SoC for non-mobile applications is at stakes.

As pointed out by some market researches the DSP industry is favoring hybrid SoC solutions mixing RISC with specialized DSP cores, instead of discrete DSP IC. This is surely a question of both market trends and technical and production aspects. First it is questionable whether FLOPS promises have been maintained. While in an IEEE report [142] dated 2000, Gene Frantz, of Texas Instruments, forecasted DSPs by 2010 to have 50,000 MIPS[8], at the time of writing TI has a family of DSPs (C66x) capable of 22.4GFLOPS per core at 1.4GHz (or double the number of MACs). However, TI Keystone II SoC include up to 8 DSP cores (and up to 4 Cortex A15), hence parallelism can drive performances to theoretical 198 GFLOPS (up to TI documentation). All this performance come at a cost: 160 US$ per 1,000 units, at least one order of magnitude higher than previous generation C67x DSPs (which is rated at 2.7 GFLOPS, i.e. one order of magnitude lower performance). Although the Keystone II generation of SoC also include additional accelerators for security and network packets to improve connectivity functions it does not include all the feature of mobile SoCs, probably due to TI decision to leave the mobile market in September 2012[9] (which in 2006 was 80% of the DSP sales, i.e. 35% of the whole company revenues [10]). TI is one of the leading companies in discrete DSP. Other companies being Analog Devices (which according to the internal company Annual Investors Reports has a stable 9% revenue from the DSP market[11]) and Freescale Semiconductors, formerly Motorola. Despite number crunching is as useful as ever, DSP chips are only a small portion of the DSP silicon market: 10% (as of Nov. 2012) [12]. Indeed, a number of DSP intel-

[8]In an article from electronicdesign.com Frantz is quoted having promised 3 Trillion Instructions per Seconds by 2010, which however is in strong contrast with the 50,000 MIPS figure reported in the IEEE report. The article can be found online at http://electronicdesign.com/dsps/tis-dsp-roadmap-promises-3-trillion-instructions-second-2010

[9]according to Forbes http://www.forbes.com/sites/greatspeculations/2013/03/28/heres-why-texas-instruments-stock-is-worth-35/

[10]according to Plunkett Research Ltd. "Plunkett's Outsourcing and Offshoring Industry Almanac 2008"

[11]http://investor.analog.com/annuals.cfm

[12]https://fwdconcepts.com/dsp-market-bulletin-111212/

lectual properties (IP) are nowadays embedded in SoC in form of accelerators or supplementary cores. This is the case of CEVA[13], an established DSP IP company. CEVA cores are specific to baseband communication, and since 2011 it seems to be the most employed architecture for mobile connectivity, being shipped with a rate of 1 Billion units per year (in year 2013[14]). Hexagon cores, developed and produced by Qualcomm, ship with Qualcomm Snapdragon SoCs since 2006 to assist either the Modem subsystem or the Multimedia subsystem of the SoC. It is estimated that 1.2 billions Hexagon cores were shipped in 2011 inside Qualcomm SoCs.

Despite the mobile market being the most important field for DSP application (in terms of sales), and despite its large growth, in the past there has been a large deal of retreats from the market, selling and fusion of DSP chips manufacturers [15]. As the discrete DSP ICs market is shrinking, new solutions for DSP may be worth evaluating. A last, radical question needs be expressed: should musical applications migrate to custom embedded hardware based on ARM or similar SoCs, or should the mobile platforms be the target for embedding musical applications and make out into the tangible world?

6.2.2 Wireless Networked Music Performance

Issues with wireless NMP are twofold: at the lower level, access to the medium, range, bandwidth and interferences of existing standards need to be modified to specifically target this application; at the top level, a data presentation standard, session negotiation and control need a standardization. The first aspect requires a large effort from developers, manufacturers and users groups, and the question is whether these different actors will ever join to introduce new features in the current standards for NMP and similar applications. In MAs the solutions are often simple to implement, as it is the case with audio processes scheduling: for concurrent transmission of multiple audio packets in a

[13] http://www.ceva-dsp.com/

[14] EE Times Asia http://www.eetasia.com/articleLogin.do?artId=8800667242

[15] see e.g. years 2007-2009, with Agere cellphone chips sold to Infineon, Analog Devices cellphone chips sold to Mediatek, TI and Freescale dumping their 2G/2.5G products, https://fwdconcepts.com/dsp-market-bulletin-5409/

short time frame a time-slotting mechanism could be employed. Several TDMA (Time Domain Multiplexing) algorithms are implemented in the Mikrotik devices described in Section 4.7 as an alternative to standard 802.11 medium access policies. Besides the medium access, some other problematic aspects of 802.11 are:

- the high frequency of operation has a reduced range. The sub-1 GHz range would be suited to longer transmission range. Currently available transmission standards in the 860-930 MHz range, unlicensed in most countries, have channels with narrow bandwidth, limiting the information that can be coded on a channel
- the number of potential interferers in the 2.4 GHz and 5 GHz ISM bands.
- the need for integrated antenna with a wide range, potentially improved by adaptive beamforming,

At the application level, an effort is required from many software developers to follow one standard. The main aspects that need be shared between different software implementations are:

- a metadata structure that includes all the relevant audio parameters, to assess feasibility of the audio link, instantiate a resampler if needed, allocate the proper audio buffer size, etc.;
- a set of header and payload structures for audio exchange: most audio software (netsend, jacktrip, AUnetsend, netjack, etc.) exchange audio in similar ways, but the packet header, numeric data representation and interleaving are slightly different;
- a negotiation protocol to exchange the metadata, negotiate audio link features and establish a connection;
- a session control protocol; it can be drawn from existing protocols but it needs adoption by all NMP audio softwares.

At the moment, all these aspects are covered by the AES67 standard [49]. Implementation in existing software could solve the aforementioned issues.

The AES67 standard also provides for a minimum set of requirements in terms of latency and audio packet size. Those requirements are 48kHz, 24-bit sampling, 48 samples per period and 3 periods per buffer. This means 3 ms buffer time. With network delay $\leq 3ms$ an AES67-compatible transmission could be done.

6.2.3 Digital Signal Processing for Musical Applications

More than 40 years have passed since the first proposal for a physics-based digital synthesis technique [113]. The computational models evolved and many formulations have been devised that yield a vast computational cost reduction, making it feasible to implement these into personal computer software and commercial musical instruments. The complexity of these models is ever increasing, to match the need of the experienced musicians, and thus the number of parameters and the accuracy of their tuning gets more critical. Perhaps other traditional techniques, more faithful to the physics of the system, such as Finite Differences and Wave Digital formulations, may ease this problem. While, in fact, techniques such as DWT or modal synthesis require a psychoacoustic match of the emulated tones to the real instrument ones, the FDTD and WDF adhere to the physics of the system and therefore can provide faithful emulation by matching the physical constants and properties of the material and components involved in producing sound. At the moment such techniques are relegated to off-line usage. The FDTD is successfully experimented by Stefano Bilbao and his colleagues on high performance general purpose GPUs [11], and it is possible that in future real-time FDTD synthesis will be viable, thanks to the availability of GPUs on personal computers and mobile devices.

Besides the mere computational aspects, parametrization and faithfulness are fundamental aspects in sound synthesis. The palette of synthesis techniques is so large at the moment, that a sound designer or a musician can meet a goal with ease, given valid tools. He, or she, can generate a desired sound drawing from a great deal of techniques: subtractive, additive, frequency or phase modulation, waaveshaping, wavetable, sampling, granular, vector phase shaping [143], and of course all the aforementioned physics-related ones. The general interest, thus, tends towards the use and the control of the available tools. With the processing power available nowadays there are also little to none literature contributions in terms of optimization, while optimization is a key factor in implementing ever more complex physical models on available processors to increase the level of realism in the emulation. One point emerged from informal discussion with academic colleagues at computer music conferences on the topic. Generally, the interest of these academic communities is not related to realistic emulation. The professional music market, on the other side, is not anymore in search of bright inventions, on the opposite the revival of a *vintage* trend in popular culture can be clearly seen. A recent music magazine article emphasizes on the steady growth of the analog modular synthesizers[16]: *"There's a huge new corner of the upstairs main hall that's been taken over by 10-15 independent modular synthesis companies, many of whom we've never seen showing gear at NAMM before. They banded together to rent out a large swath of the convention floor, setting up right next to each other with stations demoing their latest innovations. [...] We'll see how much modular gear starts to make it into the mainstream music production culture in the next year, but this was a clear indicator that there's an analog storm coming.* The increasing attention towards analog electronics or digital reproduction of vintage instruments pushes the attention towards virtual analog modelling, a very mature field, both industrially and from a research standpoint.

It is hard, thus, to point out what the upcoming trends will be. There are many open issues in understanding the acoustics of stringed instruments, percussion instruments, especially related to nonlinear effects. Some mathematical tools could be needed to handle large parameter spaces for model tuning. Perhaps sparse coding methods will compress the parameters space, also reducing the cost of calculating these and the bandwidth necessary to communicate these during real-time synthesis to parallel processors. Finally, the quest for a rational assessment method for physical models is open, and is of the highest

[16] from http://www.djtechtools.com/2015/01/26/the-djproducer-nammies-best-of-namm-2015

interest, since it is related to aspects of lesser technical nature, which are fundamental for the steady development of the music market and the definition of contemporary music culture.

6.3 Conclusions

This thesis is meant as a personal milestone in music technology research and development. One goal of the works conducted even before the beginning of the PhD studies was to bridge a gap between academic research and industrial development, with the secret hope to see one day the zealous local research community foster a renaissance of the once-prolific instrument manufacturing industry of my region. While we won't hear anymore the sound of housewives tuning accordion reeds at home, we could still see a number of SMEs creating and crafting novel musical instruments for the digital age exploiting global technologies and local expertise and taste. For this reason, many different topics, issues and arguments are conveyed in such a - relatively - short text. Much more remained in my notebooks, anyway. This thesis dealt mainly with embedded architectures for digital musical instruments and wireless networked performance. Both topics have needs and issues in common. A part of the development has been oriented to providing a viable solution for showing that *(a)* new SoCs are capable of real-time DSP for MAs and *(b)* that IEEE 802.11 wireless networking is able to satisfy the needs of NMP. Many issues must be addressed, still, to increase robustness and obtain compliance with existing software and protocols. The acceptance of wireless networking in music performance is not easily foreseeable in the near future. Of further concern, besides technical and usability issues there are also public health concerns related to radiated EM fields. The academic literature still fails to provide a clear answer.

As a final remark: Alexander Carôt, in his PhD thesis, expressed a concept very clearly: interdisciplinary research is a double-edged sword. Trying to summarize and convey all the research works conducted in more than three years of music technology research is a hard task. The risk of being generalist or unclear is high. Both the open-source developers and the academic communities tends to underestimate the importance of taking into consideration different issues at a time. Science is progressing slowly, unable to suggest the intellectuals (do we have these anymore, or is society mainstream culture lead by starlettes' Instagram selfies?), to shed light on currently debated issues or to provide a mainstream cultural outcome. It is hard to tell whether this is due to a critical mass of analytic knowledge and the lack of comprehensive interdisciplinary approaches to knowledge and research, or whether, in the rush of quantifying the research outcome, the academic institutions are losing their ability to produce highly relevant scientific research. It is a fact that the academic literature is now a flourishing business for low quality journals and some far east countries established a very good reputation in assembly line paper writing.

To conclude, in the hope to justify some lacks in the work conducted in the last three years, part of the day-time (night-time at times[17]) has been devoted to other topics not covered here, which were funded by a research grant from Telecom Italia, and were of help in the development of my engineering abilities.

[17]intended pun

Bibliography

[1] C. Erkut and M. Karjalainen, “Finite difference method vs. digital waveguide method in string instrument modeling and synthesis,” *AES 22nd Int. Conf. on Virtual, Synthetic and Entertainment Audio*, pp. 317–323, Espoo, Finland, 2002.

[2] H. Järveläinen, V. Välimäki, and M. Karjalainen, “Audibility of the timbral effects of inharmonicity in stringed instrument tones,” *Acoustics Research Letters Online, ASA*, vol. 2, no. 3, pp. 79 – 84, April 2001.

[3] M. V. Mathews, “Computer program to generate acoustic signals,” *The Journal of the Acoustical Society of America*, vol. 32, no. 11, pp. 1493–1493, 1960.

[4] M. V. Mathews, “The digital computer as a musical instrument,” *Science*, vol. 142, no. 3592, pp. 553–557, Nov 1963.

[5] S. Cass, “An elf remade,” *IEEE Spectrum*, vol. 3, pp. 19–20, March 2015.

[6] J. M. Chowning, “Method of synthesizing a musical sound,” Apr. 19 1977, US Patent 4,018,121.

[7] K. Karplus and A. Strong, “Digital synthesis of plucked-string and drum timbres,” *Computer Music Journal 7(2): 43-55*, 1983.

[8] D. A. Jaffe and J. O. Smith, “Extensions of the Karplus-Strong plucked-string algorithm,” *Computer Music Journal*, vol. 7, no. 2, pp. 56 – 69, 1983.

[9] J. Pekonen and V. Välimäki, "The brief history of virtual analog synthesis," in *Proc. 6th Forum Acusticum. Aalborg, Denmark: European Acoustics Association*, 2011, pp. 461–466.

[10] J. O. Smith, "Physical modeling synthesis update," *Computer Music Journal*, pp. 44–56, 1996.

[11] S. Bilbao, A. Torin, P. Graham, J. Perry, and G. Delap, "Modular physical modeling synthesis environments on GPU," in *Proc. 2014 International Computer Music Conference*, 2014.

[12] R. Mehra, N. Raghuvanshi, L. Savioja, M. C. Lin, and D. Manocha, "An efficient GPU-based time domain solver for the acoustic wave equation," *Applied Acoustics*, vol. 73, no. 2, pp. 83–94, 2012.

[13] H. Chamberlin, *Musical applications of microprocessors, Second Edition*, Hayden Book Company, New Jersey, 1985.

[14] D. K. Wise, "The modified Chamberlin and Zölzer filter structures," in *Digital Audio Effects (DAFX) 2006*, 2006.

[15] R. Weidenaar, *Magic music from the Telharmonium*, The Scarecrow Press, Inc., 1995.

[16] A. Carôt, P. Rebelo, and A. Renaud, "Networked music performance: State of the art," in *Audio Engineering Society Conference: 30th International Conference: Intelligent Audio Environments*, Mar 2007.

[17] J. Pritchett, *The Music of John Cage*, vol. 5, Cambridge University Press, 1996.

[18] J. Bischoff, R. Gold, and J. Horton, "Music for an interactive network of microcomputers," *Computer Music Journal*, pp. 24–29, 1978.

[19] Á. Barbosa, "Displaced soundscapes: A survey of network systems for music and sonic art creation," *Leonardo Music Journal*, vol. 13, pp. 53–59, 2003.

[20] S. Gresham-Lancaster, "The aesthetics and history of the Hub: The effects of changing technology on network computer music," *Leonardo Music Journal*, pp. 39–44, 1998.

[21] N. Collins, *Introduction to computer music*, John Wiley & Sons, 2010.

[22] P. Rebelo and A. B. Renaud, "The frequencyliator: distributing structures for networked laptop improvisation," in *Proceedings of the 2006 conference on New interfaces for musical expression.* IRCAM—Centre Pompidou, 2006, pp. 53–56.

[23] A. B. Renaud, "Cueing and composing for long distance network music collaborations," in *Audio Engineering Society Conference: 44th International Conference: Audio Networking.* Audio Engineering Society, 2011.

[24] C. Chafe, S. Wilson, R. Leistikow, D. Chisholm, and G. Scavone, "A simplified approach to high quality music and sound over IP," in *COST-G6 Conference on Digital Audio Effects*, 2000, pp. 159–164.

[25] A. Xu, W. Woszczyk, Z. Settel, B. Pennycook, R. Rowe, P. Galanter, and J. Bary, "Real-time streaming of multichannel audio data over Internet," *Journal of the Audio Engineering Society*, vol. 48, no. 7/8, pp. 627–641, 2000.

[26] C. Drioli and C. Allocchio, "LOLA: a low-latency high quality A/V streaming system for networked performance and interaction," in *Colloqui Informatica Musicale, Trieste*, 2012.

[27] C. Alexandraki and D. Akoumianakis, "Exploring new perspectives in network music performance: The DIAMOUSES framework," *Computer Music Journal*, vol. 34, no. 2, pp. 66–83, 2010.

[28] F. L. Schiavoni, M. Queiroz, and F. Iazzetta, "Medusa-a distributed sound environment," in *Linux Audio Conference*, 2011, pp. 149–156.

[29] J. Allison, "Distributed performances systems using HTML5 and Rails," in *26th Annual Conference of the Society for Electro-Acoustic Music in the United States (SEAMUS)*, 2011.

[30] S. Holland, K. Wilkie, P. Mulholland, and A. Seago, *Music Interaction: Understanding Music and Human-Computer Interaction*, Springer, 2013.

[31] N. Bowen, *Mobile phones, group improvisation, and music: Trends in digital socialized music-making*, Ph.D. thesis, The City University of New York, 2013.

[32] G. Weinberg, "Interconnected musical networks: Toward a theoretical framework," *Computer Music Journal*, vol. 29, no. 2, pp. 23–39, 2005.

[33] C. Chafe and M. Gurevich, "Network time delay and ensemble accuracy: Effects of latency, asymmetry," in *117th Audio Engineering Society Convention*. Audio Engineering Society, 2004.

[34] C. Chafe, J.-P. Caceres, and M. Gurevich, "Effect of temporal separation on synchronization in rhythmic performance," *Perception*, vol. 39, no. 7, pp. 982, 2010.

[35] P. F. Driessen, T. E. Darcie, and B. Pillay, "The effects of network delay on tempo in musical performance," *Computer Music Journal*, vol. 35, no. 1, pp. 76–89, 2011.

[36] Y. Kobayashi and Y. Miyake, "Analysis of network ensemble between humans with time lag," in *SICE 2003 Annual Conference*, Aug 2003, vol. 1, pp. 1069–1074 Vol.1.

[37] G. Baltas and G. Xylomenos, "Ultra low delay switching for networked music performance," in *Information, Intelligence, Systems and Applications, IISA 2014, The 5th International Conference on*, July 2014, pp. 70–74.

[38] M. Fink and U. Zölzer, "Low-delay error concealment with low computational overhead for audio over IP applications," in *Digital Audio Effects Conference (DAFx-14), Erlangen, Germany*, September 2014.

[39] A. Carôt and C. Werner, "Network music performance-problems, approaches and perspectives," in *Proceedings of the "Music in the Global Village"-Conference, Budapest, Hungary*, 2007.

[40] N. Schuett, "The effects of latency on ensemble performance," 2002.

[41] J.-M. Valin, T. B. Terriberry, C. Montgomery, and G. Maxwell, "A high-quality speech and audio codec with less than 10-ms delay," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 18, no. 1, pp. 58–67, 2010.

[42] S. Farner, A. Solvang, A. Sæbo, and P. U. Svensson, "Ensemble hand-clapping experiments under the influence of delay and various acoustic environments," *J. Audio Eng. Soc*, vol. 57, no. 12, pp. 1028–1041, 2009.

[43] M. Gurevich, C. Chafe, G. Leslie, and S. Tyan, "Simulation of networked ensemble performance with varying time delays: Characterization of ensemble accuracy," in *Proceedings of the 2004 International Computer Music Conference, Miami, USA*, 2004.

[44] A. Carôt, U. Krämer, and G. Schuller, "Network music performance (NMP) in narrow band networks," in *Audio Engineering Society Convention 120*. Audio Engineering Society, 2006.

[45] S. Moon, P. Skelly, and D. Towsley, "Estimation and removal of clock skew from network delay measurements," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, Mar 1999, vol. 1, pp. 227–234 vol.1.

[46] C. Rentel and T. Kunz, "A mutual network synchronization method for wireless ad hoc and sensor networks," *Mobile Computing, IEEE Transactions on*, vol. 7, no. 5, pp. 633–646, May 2008.

[47] A. Carôt and C. Werner, "External latency-optimized soundcard synchronization for applications in wide-area networks," in *AES 14th Regional Convention, Tokio, Japan*, 2009, vol. 7, p. 10.

[48] F. Rumsey, "Audio in the age of digital networks," *J. Audio Eng. Soc*, vol. 59, no. 4, pp. 244–253, 2011.

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

[50] "Tim Shuttleworth (editor), AES Technical Committee on Network Audio Systems, "Emerging Technology Trends Report"," November 2011.

[51] B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Multimedia streaming via TCP: An analytic performance study," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, vol. 4, no. 2, pp. 16, 2008.

[52] A. Aurelius, C. Lagerstedt, and M. Kihl, "Streaming media over the Internet: Flow based analysis in live access networks," in *Broadband Multimedia Systems and Broadcasting (BMSB), 2011 IEEE International Symposium on*, June 2011, pp. 1–6.

[53] G. Kreitz and F. Niemela, "Spotify–large scale, low latency, P2P music-on-demand streaming," in *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*. IEEE, 2010, pp. 1–10.

[54] R. Mills, "Dislocated sound: A survey of improvisation in networked audio platforms," in *New Interfaces for Musical Expression, (NIME 2010, Sidney, Australia*. 2010, University of Technology, Sydney.

[55] R. Bargar, S. Church, A. Fukuda, J. Grunke, D. Keislar, B. Moses, B. Novak, B. Pennycook, Z. Settel, J. Strawn, P. Wiser, and W. Woszczyk, "Networking audio and music using Internet2 and next-generation internet capabilities," 1998.

[56] J.-P. Cáceres, R. Hamilton, D. Iyer, C. Chafe, and G. Wang, "To the edge with China: Explorations in network performance," in *4th International Conference on Digital Arts, ARTECH 2008, Porto, Portugal*, 2008, pp. 61–66.

[57] J. Reuter, "Case study: Building an out of the box Raspberry Pi modular synthesizer," in *Linux Audio Conference (LAC2014), Karlsruhe, Germany*, 2014.

[58] S. Nikkilä, "Introducing wireless organic digital audio: a multichannel streaming audio network based on IEEE 802.11 standards," in *AES 44th International Conference, San Diego, CA, USA*, November 2011.

[59] S. Letz, S. Denoux, and Y. Orlarey, "Audio rendering/processing and control ubiquity? a solution built using faust dynamic compiler and JACK/NetJack," in *Joint Internation Computer Music Conference and Sound and Music Computing (ICMC+SMC14), Athens, Greece*, September 2014, p. 1518.

[60] L. Gabrielli, S. Squartini, and F. Piazza, "Advancements and performance analysis on the wireless music studio (WeMUST) framework," in *AES 134th Convention*, May 2013.

[61] J.-P. Cáceres and C. Chafe, "Jacktrip: Under the hood of an engine for network audio," *Journal of New Music Research*, vol. 39, no. 3, pp. 183–187, 2010.

[62] F. Adriaensen, "Using a DLL to filter time," in *Linux Audio Conference*, 2005.

[63] J.-P. Cáceres and C. Chafe, "Jacktrip: Under the hood of an engine for network audio," *Journal of New Music Research*, vol. 39, no. 3, pp. 183–187, 2010.

[64] L. Gabrielli, M. Bussolotto, and S. Squartini, "Reducing the latency in live music transmission with the BeagleBoard xM through resampling," in *Proceedings of the European Embedded Design in Education and Research Conference, Milan, Italy*, 2014.

[65] J. Topliss, V. Zappi, and A. McPherson, "Latency performance for real-time audio on Beaglebone Black," in *Linux Audio Conference (LAC2014), Karlsruhe, Germany*, 2014.

[66] F. Meier, M. Fink, and U. Zölzer, "The Jamberry-a stand-alone device for networked music performance based on the Raspberry Pi," in *Linux Audio Conference, Karlsruhe*, 2014.

[67] E. Berdahl and W. Ju, "Satellite CCRMA: A musical interaction and sound synthesis platform," in *Int. Conf. on New Interfaces for Musical Expression (NIME), Oslo, Norway*, 30 May - 1 June 2011.

[68] L. Gabrielli, S. Squartini, E. Principi, and F. Piazza, "Networked beagleboards for wireless music applications," in *EDERC 2012*, September 2012.

[69] B. Razavi, *Design of analog CMOS integrated circuits*, Tata McGraw-Hill Education, 2002.

[70] C.-K. K. Yang, "Delay-locked loops-an overview," *Phase-Locking in High-Peformance Systems, Wiley-IEEE Press, New York, NY*, pp. 13–22, 2003.

[71] M. Rahman Siddique, J. Kamruzzaman, and M. Hossain, "An analytical approach for voice capacity estimation over WiFi network using ITU-T E-model," *Multimedia, IEEE Transactions on*, vol. 16, no. 2, pp. 360–372, Feb 2014.

[72] M. Blain, "Issues in instrumental design: the ontological problem (opportunity?) of 'liveness' for a laptop ensemble," *Journal of Music, Technology & Education*, vol. 6, no. 2, pp. 191–206, 2013.

[73] L. Dahl, "Wicked problems and design considerations in composing for laptop orchestra," in *International Conference on New Interfaces for Musical Expression (NIME)*, 2012.

[74] L. Gabrielli, F. Piazza, and S. Squartini, "Adaptive linear prediction filtering in DWT domain for real-time musical onset detection," *EURASIP Journal on Advances in Signal Processing*, vol. 2011, 2011.

[75] P. P. Vaidyanathan, *Multirate systems and filter banks*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[76] F. Faccenda, S. Squartini, E. Principi, L. Gabrielli, and F. Piazza, "A real-time dual-channel speech reinforcement system for intra-cabin communication," *J. Audio Eng. Soc*, vol. 61, no. 11, pp. 889–910, 2013.

[77] K. Eneman and M. Moonen, "Iterated partitioned block frequency-domain adaptive filtering for acoustic echo cancellation," *Speech and Audio Processing, IEEE Transactions on*, vol. 11, no. 2, pp. 143–158, Mar 2003.

[78] L. Gabrielli and S. Squartini, "Ibrida: A new DWT-domain sound hybridization tool," in *45th AES International Conference*. Audio Engineering Society, 2012.

[79] M. Caetano and N. Osaka, "A formal evaluation framework for sound morphing," in *International Computer Music Conference, ICMC2012, Lubljana, Slovenia*, 2012.

[80] G. Ferroni, E. Marchi, F. Eyben, L. Gabrielli, S. Squartini, and B. Schuller, "Onset detection exploiting adaptive linear prediction filtering in DWT domain with bidirectional long short-term memory neural networks," in *ISMIR 2013, Annual Meeting of the MIREX 2013 community as part of the 14th International Conference on Music Information Retrieval*, 2013.

[81] E. Marchi, G. Ferroni, F. Eyben, L. Gabrielli, S. Squartini, and B. Schuller, "Multi-resolution linear prediction based features for audio onset detection with bidirectional LSTM neural networks," in *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2014.

[82] R. R. Coifman, Y. Meyer, S. Quake, and M. V. Wickerhauser, "Signal processing and compression with wavelet packets," in *Wavelets and their applications*, pp. 363–379. Springer, 1994.

[83] I. Daubechies, *Ten lectures on wavelets*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[84] E. Marchi, G. Ferroni, F. Eyben, S. Squartini, and B. Schuller, "Audio onset detection: A wavelet packet based approach with recurrent neural networks," in *Neural Networks (IJCNN), 2014 International Joint Conference on.* IEEE, 2014, pp. 3585–3591.

[85] W. Lee and C. Kuo, "Musical onset detection based on adaptive linear prediction," in *IEEE International Conference on Multimedia and Expo*, 2006, pp. 957–960.

[86] S. Haykin, *Adaptive Filter Theory (4th Edition)*, Prentice Hall, September 2001.

[87] W.-C. Lee and C.-C. J. Kuo, "Improved linear prediction technique for musical onset detection," *International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, vol. 0, pp. 533–536, 2006.

[88] N. Erdol and F. Basbug, "Wavelet transform based adaptive filters: analysis and new results," *IEEE Transactions on Signal Processing*, vol. 44, no. 9, pp. 2163 –2171, sep. 1996.

[89] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[90] F. Weninger, J. Bergmann, and B. Schuller, "Introducing CURRENNT–the munich open-source cuda recurrent neural network toolkit," *Journal of Machine Learning Research*, vol. 15, 2014.

[91] L. Gabrielli, M. Giobbi, S. Squartini, and V. Välimäki, "Adaptive digital oscillator for virtual acoustic feedback," in *Audio Engineering Society 136th Convention.* Audio Engineering Society, 2014.

[92] L. Gabrielli, M. Giobbi, S. Squartini, and V. Välimäki, "A nonlinear second-order digital oscillator for virtual acoustic feedback," in *ICASSP,*

IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, 2014.

[93] V. Singh, "Discussion on Barkhausen and Nyquist stability criteria," *Analog Integrated Circuits and Signal Processing*, vol. 62, no. 3, pp. 327–332, 2010.

[94] G. De Sanctis and A. Sarti, "Virtual analog modeling in the wave-digital domain," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 18, no. 4, pp. 715–727, 2010.

[95] B. Bank, S. Zambon, and F. Fontana, "A modal-based real-time piano synthesizer," *IEEE Trans. Audio Speech Lang. Processing*, vol. 18, no. 4, pp. 809–821, 2010.

[96] B. Bank, *Physics-based Sound Synthesis of String Instruments Including Geometric Nonlinearities*, Ph.D. thesis, Budapest University of Technology and Economics, Hungary, 2006.

[97] A. S. Sedra and K. C. Smith, *Microelectronic Circuits, 5th Ed.*, Oxford University Press, Inc., Oxford, UK, 2004.

[98] N. Hanmeng and P. Pranayanuntana, "A sinusoidal nonlinear oscillator with adjustable frequency," *Kasetsart Journal: Natural Science*, vol. 43, no. 5, pp. 372–378, December 2009.

[99] B. Kollmeier, T. Brand, and B. Meyer, "Perception of speech and sound," in *Springer Handbook of Speech Processing*, J. Benesty, M. M. Sondhi, and Y. Huang, Eds., p. 65. Springer, 2008.

[100] J. Pakarinen and D. T. Yeh, "A review of digital techniques for modeling vacuum-tube guitar amplifiers," *Computer Music Journal*, vol. 33, no. 2, pp. 85–100, 2009.

[101] J.-J. E. Slotine and W. Li, *Applied Nonlinear Control*, Prentice-Hall, 1991.

[102] P. McLeod, *Fast, accurate pitch detection tools for music analysis*, Ph.D. thesis, University of Otago. Department of Computer Science, 2009.

[103] L. Gabrielli, V. Välimäki, H. Penttinen, S. Squartini, and S. Bilbao, "A digital waveguide-based approach for Clavinet modeling and synthesis," *EURASIP Journal on Advances in Signal Processing*, vol. 2013, no. 1, pp. 103, 2013.

[104] L. Gabrielli, L. Remaggi, S. Squartini, and V. Välimäki, "A finite difference method for the excitation of a digital waveguide string model," in *AES 134th Convention*, may 2013.

[105] L. Gabrielli, V. Välimäki, and S. Bilbao, "Real-time emulation of the Clavinet," in *International Computer Music Conference.* ICMA, 2011.

[106] L. Gabrielli, S. Squartini, and V. Välimäki, "A subjective validation method for musical instrument emulation," in *131st Audio Eng. Soc. Convention, New York*, 2011.

[107] L. Remaggi, L. Gabrielli, R. de Paiva, V. Välimäki, and S. Squartini, "A pickup model for the Clavinet," in *DAFx-2012 (Digital Audio Effects), York, UK*, 2012.

[108] J. O. Smith, "Physical modeling using digital waveguides," *Computer Music Journal*, vol. 16, no. 4, pp. 74–91, 1992.

[109] J. O. Smith, "Music application of digital waveguides," *Tech. Report STAN-M-39, Stanford University, Department of Music*, Stanford, CA, 1987.

[110] J. O. Smith, *Physical Audio Signal Processing for Virtual Musical Instruments and Digital Audio Effects*, W3K Publishing, http://www.w3k.org/books/, 2010.

[111] V. Välimäki, J. Pakarinen, C. Erkut, and M. Karjalainen, "Discrete-time modelling of musical instruments," *Reports on Progress in Physics*, vol. 69, pp. 1–78, 2006.

[112] S. Bilbao, *Numerical Sound Synthesis: Finite Difference Schemes and Simulation in Musical Acoustics*, Wiley and Sons, Chichester, UK, 2009.

[113] L. Hiller and P. Ruiz, “Synthesizing musical sounds by solving the wave equation for vibrating objects: 1st part,” *J. Audio Engineering Society*, pp. 462–470, 1971.

[114] L. Hiller and P. Ruiz, “Synthesizing musical sounds by solving the wave equation for vibrating objects: 2nd part,” *J. Audio Engineering Society*, pp. 542–551, 1971.

[115] M. Karjalainen and C. Erkut, “Digital Waveguides versus Finite Difference Structures: Equivalence and Mixed Modeling,” *EURASIP Journal on Advances in Signal Processing*, , no. 7, pp. 978–989, 2004.

[116] M. Karjalainen, “1-D digital waveguide modeling for improved sound synthesis,” *Proc. Int. Conference on Acoustics, Speech and Signal Processing (IEEE)*, vol. 2, pp. 1869–1872, Orland, USA, 2002.

[117] S. Bilbao and M. Rath, “Time domain emulation of the Clavinet,” in *AES 128th Convention, London, UK*, May 2010.

[118] M. Karjalainen, C. Erkut, and L. Savioja, “Compilation of unified physical models for efficient sound synthesis,” in *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP’03). 2003 IEEE International Conference on.* IEEE, 2003, vol. 5, pp. V–433.

[119] R. C. de Paiva, J. Pakarinen, and V. Välimäki, “Acoustics and modeling of pickups,” *J. Audio Engineering Society*, vol. 60, no. 10, October 2012.

[120] N. G. Horton and T. R. Moore, “Modeling the magnetic pickup of an electric guitar,” *American Journal of Physics*, vol. 77, pp. 144–150, 2009.

[121] A. Nackaerts, B. De Moor, and R. Lauwereins, “Measurement of guitar string coupling,” in *Proc. Int. Computer Music Conf.*, 2002, pp. 321–324.

[122] J. Rauhala, H. Lehtonen, and V. Välimäki, "Fast automatic inharmonicity estimation algorithm," *J. Acoustical Society of America*, vol. 121, no. 5, pp. 184–189, 2007.

[123] H. Fletcher, E. D. Blackham, and R. Straton, "Quality of piano tones," *J. Acoustical Society of America*, vol. 34(6), pp. 749–761, 1962.

[124] B. Bank, *Physics-based sound synthesis of the piano*, Ph.D. thesis, Helsinki University of Technology, 2000.

[125] J. Rauhala, "The beating equalizer and its application to the synthesis and modification of piano tones," in *Proc. Int. Conference on Digital Audio Effects (DAFx), Bordeaux, France*, 2007, pp. 181–187.

[126] G. Lemarquand and V. Lemarquand, "Calculation method of permanent-magnet pickups for electric guitars," *IEEE Transactions on Magnetics*, vol. 43, no. 9, pp. 3573–3578, 2007.

[127] R. C. D. de Paiva, J. Pakarinen, V. Välimäki, and M. Tikander, "Real-time audio transformer emulation for virtual tube amplifiers," *EURASIP Journal on Advances in Signal Processing*, vol. 2011, no. 1, pp. 347–645, 2011.

[128] V. Välimäki, M. Laurson, and C. Erkut, "Commuted waveguide synthesis of the clavichord," *Computer Music Journal*, vol. 27, no. 1, pp. 71–82, Spring 2003.

[129] S. Zambon, L. Gabrielli, and B. Bank, "Expressive physical modeling of keyboard instruments: From theory to implementation," in *Audio Engineering Society Convention 134*. Audio Engineering Society, 2013.

[130] N. Fletcher and T. Rossing, *The Physics of Musical Instruments*, Springer, 1998.

[131] B. Bank, S. Zambon, and F. Fontana, "A modal-based real-time piano synthesizer," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 18, no. 4, pp. 809–821, 2010.

[132] C.-W. Wun and A. Horner, "Perceptual wavetable matching for synthesis of musical instrument tones," *J. Audio Engineering Society*, vol. 49, pp. 250–262, 2001.

[133] B. Hamadicharef and E. Ifeachor, "Objective prediction of sound synthesis quality," *115th Convention of the AES, New York, USA*, p. 8, October 2003.

[134] H. M. Lehtonen, H. Penttinen, J. Rauhala, and V. Välimäki, "Analysis and modeling of piano sustain-pedal effects," *J. Acoustical Society of America*, vol. 122, pp. 1787–1797, 2007.

[135] S. G. Wood, "Objective test methods for waveguide audio synthesis," MSc. thesis, Brigham Young University, UT, USA, 2007.

[136] S. Cho, U. Chong, and S. Cho, "Synthesis of the Dan Tranh based on a parameter extraction system," *J. Audio Engineering Society*, vol. 58, no. 6, pp. 498–507, 2010.

[137] M. Kubat, R. C. Holte, and S. Matwin, "Machine learning for the detection of oil spills in satellite radar images," *Machine Learning*, vol. 30, no. 2-3, pp. 195–215, 1998.

[138] J. A. Swets, *Signal detection theory and ROC analysis in psychology and diagnostics: Collected papers*, Lawrence Erlbaum Associates, 1995.

[139] B. Swanson, "Encouraging students towards meaningful subjective comparisons," in *Audio Engineering Society Conference: 50th International Conference: Audio Education.* Audio Engineering Society, 2013.

[140] A. Eales and R. Foss, "Service discovery using open sound control," in *Audio Engineering Society Convention 133.* Audio Engineering Society, 2012.

[141] N. Bowen and D. Reeder, "Mobile phones as ubiquitous instruments: Towards standardizing performance data on the network," in *Joint International Computer Music Conference and Sound and Music Computing (ICMC+SMC2014), Athens, Greece*, 2014.

[142] G. Frantz, “Digital signal processor trends,” *Micro, IEEE*, vol. 20, no. 6, pp. 52–59, Nov 2000.

[143] J. Kleimola, V. Lazzarini, J. Timoney, and V. Valimaki, “Vector phase shaping synthesis,” in *Proc. of the 14th Int. Conference on Digital Audio Effects (DAFx-11), Paris, France, September 19-23, 2011*, 2011.