Making Frugal Spatial Audio Systems Using Field-Programmable Gate Arrays

Romain Michon,[a] Joseph Bizien,[a] Maxime Popoff,[b] and Tanguy Risset[b]
[a]Univ Lyon, Inria, INSA Lyon, CITI, EA3720, 69621 Villeurbanne, France
[b]Univ Lyon, INSA Lyon, Inria, CITI, EA3720, 69621 Villeurbanne, France
romain.michon@inria.fr

ABSTRACT

Spatial audio systems are expensive, mostly because they usually imply the use of a wide range of speakers and hence audio outputs. Some techniques such as Wave Field Synthesis (WFS) are especially demanding in that regard making them out of reach to many individuals or even institutions. In this paper, we propose to leverage recent progress made using Field-Programmable Gate Arrays (FPGA) in the context of real-time audio signal processing to implement frugal spatial audio systems. We focus on the case of WFS and we demonstrate how to build a 32 speakers system that can manage multiple sources in parallel for less than 800 USD (including speakers). We believe that this approach contributes to making advanced spatial audio techniques more accessible.

Author Keywords

FPGA, Faust, WFS

CCS Concepts

•**Applied computing** → **Sound and music computing;** •**Hardware** → **Digital signal processing;** •**Computer systems organization** → *Real-time languages;*

1. INTRODUCTION

In the world of spatial audio, more speakers usually means "better." However, as the number of speakers grows in a system, so does the amount of computation as well as the number of digital audio outputs. In most cases, one or the other becomes a bottleneck, eventually limiting the number of speakers.

Among sound spatialization techniques, some require larger number of speakers than others. While ambisonics [5] or VBAP[1] [8] can work with just a few (at least 8), other techniques such as WFS[2] [3, 1] usually require a much larger minimum number of speakers (at least 16). WFS presents significant advantages compared to other techniques though as it typically works within a very large area (no sweet spot).

The most common approach to deal with real-time audio signal processing systems involving an extended number of speakers is to use a computer (laptop or desktop) and a multichannel audio interface. While modern CPUs[3] provide plenty of computational power to run spatial audio algorithms, few USB audio interfaces provide more than 32 audio outputs (and are usually very expensive). While some "tricks" can be used to use multiple audio interfaces of this kind in parallel [6], going above the 64 channels threshold with this kind of configuration is not easy.

An alternative approach consists of using Ethernet audio interfaces (or amplifiers) [9] which are easier to stack in parallel, allowing for a greater number of audio output channels to be used. Most of the largest WFS systems in the world such as the one available in the Lecture Hall 104 at the Technical University of Berlin featuring 2700 speakers on 832 independent channels [15] rely on this approach. However, computation can be limited by the use of desktop/laptop computers and Ethernet audio interfaces are usually very expensive making such systems completely out of reach to most people and institutions.

Field-Programmable Gate Arrays (FPGAs) are becoming an increasingly used platform for real-time audio signal processing because of the unique performances they provide in terms of audio latency, computational power, and interfacing [2, 13, 14]. Their high degree of parallelization is especially adapted to some spatial audio algorithms such as WFS where specific independent computing is carried out on each channel. Additionally, their large number of General Purpose Inputs and Outputs (GPIOs) allows for many audio chips (audio codecs[4]) to be connected to them.

The main downside of using FPGAs for real-time audio signal processing is that they're notoriously hard to program, mostly because of their inherently low-level architecture involving the use of Hardware Description Languages (HDL) such Verilog or VHDL. However, recent work carried out by our team make it possible to program FPGAs for real-time audio applications using Faust [2]. Faust[5] is a programming language for real-time audio DSP which can target a wide range of platforms and standards [7]. Faust is much more accessible than the Hardware Description Languages which are normally used to program FPGAs and it is mastered by many people in the audio community. The "Faust to FPGA" compiler is provided as part of the Syfala

[1] *Vector Based Amplitude Panning*
[2] *Wave Field Synthesis*
[3] *Central Processing Units*
[4] The term "audio codec" can be ambiguous in our field. Throughout this paper, it will refer to ADC/DAC chips adapted to audio applications.
[5] `https://faust.grame.fr` – All URLs provided in this paper were verified on Jan. 17th, 2023.

NIME'23, 31 May–2 June, 2023, Mexico City, Mexico.

tool-chain[6] which is completely open source.

In this paper, we demonstrate how to build a frugal (less than 800 USD for 32 speakers, including speakers) WFS systems entirely programmable in FAUST and targeting a large number of speakers using a Xilinx FPGA board, Adafruit i2s digital amplifiers break out boards, and DIY[7] speakers. Our approach can be used to implement WFS systems handling dozens of sources with up to 200 speakers with a medium-range FPGA board and 500 speakers with a higher-end FPGA. While we focus more on frugality here, we also show that a similar approach can be used to build professional-grade spatial audio systems.

As far as we know, there's no other example of an FPGA-based WFS system in the literature. In a paper from 2009, Theodoropoulos et al. [11] compare the theoretical performances of various processor architectures (i.e., CPUs, GPUs, FPGAs, ASICs, etc.) in the context of WFS without actually making any implementation. Later in 2011, they used an FPGA to implement a beam-forming system [12], but it was static and not programmable.

First, we present our physical/hardware set up. We then detail the various optimizations that we had to make to our WFS algorithm to efficiently run it on an FPGA board. We finally describe the performances and limits of such systems before detailing future research avenues afforded by this project. We show that our approach is not limited to WFS but can be used with any other spatial audio techniques.

2. HARDWARE

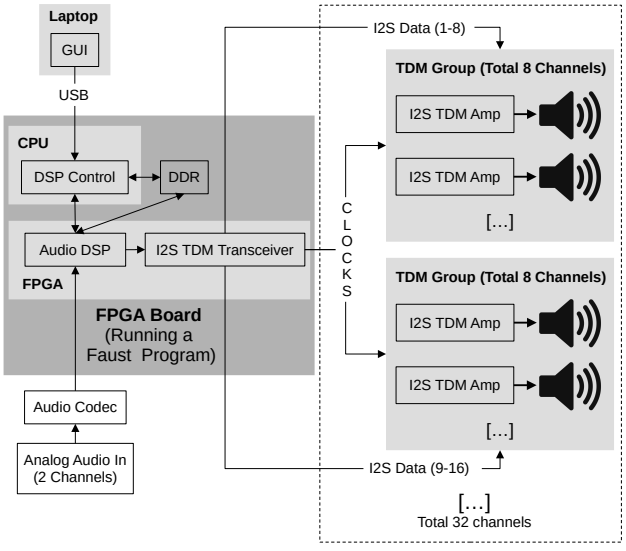

Figure 1: FPGA-Based Wave Field Synthesis system overview.

In its current configuration, the system presented in this paper is based on (see Figure 1):

- an FPGA development board, which is used to run the spatial audio algorithm, WFS, for instance (see §2.1);
- a laptop computer, which is used to run a GUI[8] to control the system;
- a series of Adafruit breakout boards for MAX98357A 3W i2s digital amplifiers[9] (see §2.2);
- a series of DIY passive speakers (see §2.4).

Figure 2 presents a picture of a prototype WFS that we built using this approach.

2.1 FPGA and Real-Time Audio DSP

The Syfala tool-chain currently supports two different FPGA development boards: the Digilent Zybo Z7 (based on a Xilinx Zynq-7000 FPGA) and Genesys (based on a Xilinx Ultrascale+ FPGA, which is much more powerful than the Zynq-7000). Hence, the approach presented in this paper is compatible with these two boards. For our prototype WFS system, we use a Zybo Z7-20 offering more than enough computational power to run a 32 channels WFS algorithm.

Thanks to Syfala, computation is balanced between the built-in CPU of the board and the FPGA. Operations related to control and that do not have an impact on the computation of audio samples happen on the CPU. All other operations happen on the FPGA. DDR[10] memory is shared between the CPU and the FPGA. Indeed, even though the FPGA itself hosts built-in memory (block RAM), it remains quite limited and hence, any buffer exceeding a certain size (which can be determined by the user) is automatically placed in the DDR RAM.

The Syfala compiler automatically decides what portion of a given FAUST program is ran on the CPU and on the FPGA and what goes into DDR. Hence, the user only has to worry about writing a FAUST program.

2.2 Getting Audio Out of the System

To implement audio outputs, we use MAX98357A 16 bits 3W i2s digital amplifiers Adafruit breakout boards. The fact that they are distributed by Adafruit make them easily accessible in many places in the world. Their cost is very low: only 6 USD a piece, but they used to be even cheaper before the "components crisis" started. They are compatible with Time Division Multiplexing (TDM), which is very important in our case. TDM allows us to share the same i2s[11] bus between 8 amplifiers, which significantly reduces the number wires and hence used GPIOs on the FPGA. i2s TDM implies the use of three wires: bit clock, word select (which indicates the beginning of a new audio sample frame), and data. Since bit clock and word select can be shared between all the amplifiers in the system, every new block of 8 channels only requires a single wire (see Figure 3). For example, for our 32 channels prototype, only 6 GPIOs are used: 2 (bit clock + word select) + 1 + 1 + 1 + 1 (blocks of 8 channels).

Since the Zybo Z7 has about 40 GPIOs and the Genesys 80, a very large number of audio outputs can be implemented using this approach (more than 200 on the Zybo Z7 and more than 500 on the Genesys).

In order for this to work, we had to implement our own i2s TDM transceiver in VHDL which is now part of the open source Syfala tool-chain. After carrying out a wide range of optimizations, it turns out that adding new audio outputs to the system has almost no cost from a computational

[6] `https://github.com/inria-emeraude/syfala`
[7] *Do It Yourself*
[8] *Graphical User Interface*
[9] `https://www.adafruit.com/product/3006`
[10] *Double Data Rate*
[11] *i2s* is a serial communication standard used to transmit audio streams between integrated circuits. In its most fundamental form, it implies the use of three wires: a bit clock, a word select (left/right), and data. If the system has both an input and an output (rx/tx), then a fourth wire is needed.

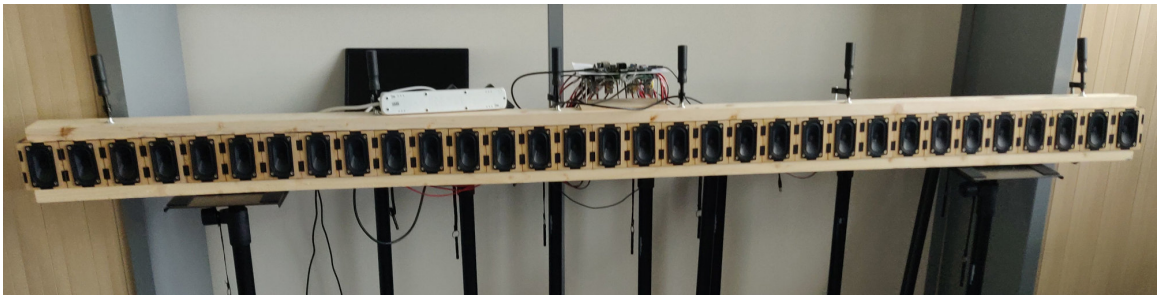

Figure 2: Prototype WFS system.

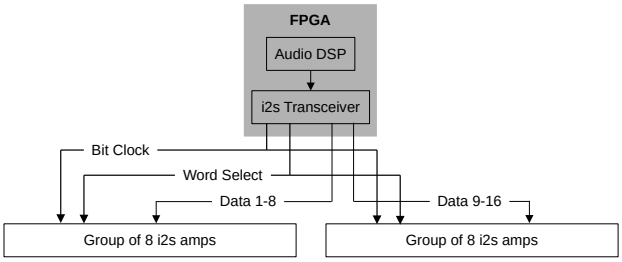

Figure 3: Wiring of TDM i2s amps.

standpoint. Indeed, dealing with a large number of parallel digital audio streams is very straightforward for an FPGA. As this is demonstrated in the following sections, the two main bottlnecks are the number of potential operations carried out on these streams as well as the number of memory accesses done by the FPGA in the DDR.

2.3 Connecting the Amps to the FPGA Board

As explained before, we decided not to use a PCB for our design to make sure that our system would remain modular, easy to assemble, and reproducible.

When dealing with i2s TDM, the quality of wiring matters a lot. In the current configuration, sampling rate is 48kHz, audio samples are coded on 16 bits, and 8 channels are transmitted on the same i2s bus. Hence, bit clock is $16 \times 48000 \times 8 = 6.144$MHz. To prevent issues, we tried to limit the length of wires as much as possible by assembling the Adafruit amplifiers breakout boards as "towers" (see Figure 4). We also use shielded wires to connect these "amp towers" to the GPIOs of the FPGA board. Not using short shielded wires can result in a lot of potentially hard to debug problems.

Channel selection between the amplifiers is carried out by placing a combination of resistors/wires between the pins of the board (per the datasheet of the MAX98357A). These wires and resistors were directly soldered to the breakout boards for our prototype. Assembling the amplifier towers then only implies connecting the data, bit clock, and word select pins together, which is very straightforward.

Speaker wires directly connect to the board through screw terminal blocks which are themselves connected to the amps. Even though the MAX98357A are very energy efficient, an external power supply is used since the FPGA board can't deliver enough power for all of them.

The board that we developed for our prototype connects directly to the FPGA board through a PMOD port.

2.4 Speakers

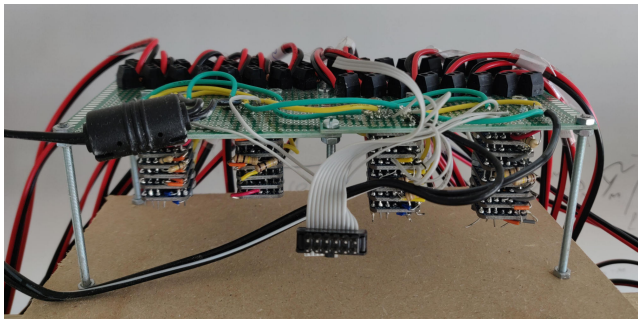

Figure 4: View of the electronic circuit of the system.

We made our own speakers for our prototype WFS system. They are based on Visaton SC 5.9 ND 8 Ohms woofers which are mounted on laser cut enclosures made out of MDF wood panels. The volume of the enclosure follows the recommendation of the speaker manufacturer and is insulated using fiber glass foam. We conducted basic frequency response measurements and decided that they were good enough for the applications that we are aiming for this prototype. The files corresponding to the speaker design can be found on the project website.[12]

Figure 5: DIY speaker developed for our prototype WFS system.

2.5 Input Sources

In our current prototype, the sources that are spatialized by the system are directly taken as analog audio inputs using

[12] https://team.inria.fr/emeraude/plasma/

the built-in audio codec of the Zybo Z7 board (which has a stereo input). Hence, our custom TDM i2s transceiver is designed to remain compatible with standard i2s and to support various devices (chips) with different standards in parallel.

Alternatively, sources can be specified directly in the Faust program as sound synthesizers, audio files pre-loaded in memory, etc.

3. WAVE FIELD SYNTHESIS ALGORITHM AND OPTIMIZATIONS

While the point of this paper is not to present new WFS algorithms, we did have to carry out a couple of optimizations on standard WFS in order for it to run more efficiently on FPGA architectures. These optimizations are presented in this section.

Few examples of WFS implementation were available in FAUST before we started working on this project. Julius O. Smith sketched the basics of a FAUST WFS program in [10] that we used as a basis for our prototype. In its most fundamental form, WFS just consists in simulating the delay of arrival between a source and the different speakers in a linear speaker array. While the theory on WFS can be pushed quite far [1], very convincing results can be obtained with some delay lines and simple gain scaling on the speakers in the array.

As mentioned in §2.2, the performances of an FPGA-based spatial audio system are not limited by the number of audio channels going through the system (the Zybo Z7 could probably manage thousands of digital audio streams in parallel) but rather by potential mathematical operations carried out on each individual stream as well as the number of memory accesses conducted by the FPGA in DDR, if external memory is needed.

However, WFS algorithms typically require a fair amount of memory because of the large number of delay lines they use, making it impossible to just run them on the FPGA chip built-in memory (block ram). For example, in a simple scenario where a 32 channels WFS system is used, a single sound source is virtually placed 340 meters away from the speaker array, simple delay lines are used (i.e., not fractional delay), audio samples are stored on 32 bits, and the sampling rate is 48kHz, then at least $(32 \times 48000 \times 32)/8 = 6.144$ MBytes of memory is needed, which is way more than what's available in the built-in memory memory of the FPGA: DDR has to be used in that case.

On the other hand, if individual delay lines are used for each speaker and are stored in the DDR, using the same configuration, a total of 32 memory accesses (each individual delay line implies one memory access) have to be carried out every $1/48000 = 20\mu s$, which is quite a lot. This would of course increase every time a new source is needed, and again, we're only talking about a "small" 32 channels WFS system here.

To limit the memory footprints and the number of memory accesses of our WFS algorithm, we optimized it by sharing a common long delay line (implemented in DDR) between all the speakers for each individual source. Final adjustments of the length of each individual delay on a given speaker are carried out through small fractional delay lines (second order Lagrange interpolation), which are implemented directly on the FPGA in block RAMs (see Figure 6). This allows us to significantly reduce the number of memory accesses carried out in DDR (only one access per source per sample) and to potentially expand the number of channels of our system.

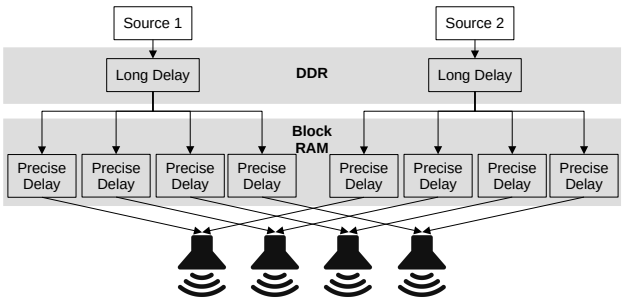

Figure 6: FPGA WFS optimization (4 speakers and 2 sources scenario) depicted as a signal flow block diagram. Each individual source only requires a single delay line to be implemented in DDR.

The current version of the FAUST WFS program that we developed as part of this project[13] uses this approach and allows the user to select an arbitrary number of sources for a system with a given number speakers (the distance between speakers can be configured, of course). The default is that sources are taken from audio inputs, which in our system are the built-in analog audio inputs of the Zybo Z7 board (see 2.5). Alternatively, synthesized or recorded sources can be played directly from the FAUST program as well (i.e., pre-recorded samples loaded directly in DDR, sound synthesizers coded in Faust, etc.). The position of each individual source can be modified in real-time using 2D Cartesian coordinates. A simple GUI (FAUST GTK UI, for instance) can be generated by the Syfala tool-chain and ran on a computer connected to the FPGA through USB. This GUI can inherit from all the features of standard Faust user interfaces (i.e., OSC/MIDI control, HTTPD server, etc.).

4. EVALUATION, PERFORMANCES, AND DISCUSSION

The WFS algorithm that we used for our system is fairly basic and standard (see §3), hence we don't believe that evaluating this aspect is necessary. However, much has to be said about the performances of our system, which is why the evaluation provided in this section is mostly technical.

The performances (i.e., computational power, number of speakers, etc.) of our FPGA-based WFS system are limited by four factors:

- the number of GPIOs available on the FPGA;
- the amount of physical resources available on the FPGA;
- the time it takes to execute a given set of operations in series on the FPGA;
- the number of memory accesses carried out in DDR within one sample cycle.

In this section, we discuss how these impact the design of our system.

4.1 Theoretical Hardware Limit for the Number of Speaker

[13]The source code is available on the Syfala project repository: https://github.com/inria-emeraude/syfala

If only passthroughs (which again, have almost no impact on computation) were implemented, using MAX98357A, **a maximum of about 200 speakers on the Zybo Z7 and more than 500 on the Genesys can be used. These figures could be doubled using more advanced audio codecs** allowing for up to 16 channels to be used with TDM (such as the Analog Devices ADAU1787). In the case where even more speakers would be needed, multiple FPGA boards could be used in parallel and synchronized. We havent't tried this option yet but there's no reason why this wouldn't work.

4.2 Memory Accesses

The optimizations presented in §3 allowed us to significantly reduce the number of memory accesses in DDR required by the WFS algorithm, which at first was the main bottleneck of our system. In the current version of our algorithm, increasing the number of outputs (speakers) doesn't have an impact on the number of memory accesses. On the other hand, every new source added to the system increases the number of memory accesses. Assuming that sources are provided as analog inputs (i.e., not synthesized on the FPGA) and that the system has 32 speakers, **a total of approximately 50 sources can be processed on the Zybo Z7 and about 100 on the Genesys**. These figures remain purely theoretical though as we didn't have the hardware resources (only a stereo audio input) to actually test them in a real-world context.

4.3 Computational Complexity

The WFS algorithm that we developed for our prototype remains extremely primitive. Each speaker output is only processed by one multiplier (gain scaling) and a second order fractional delay line. Implementing more advanced WFS algorithms will ultimately impact computation and hence the potential number of sources and speakers afforded by the system. We're currently exploring a wide range of avenues to improve the performances of the Intellectual Properties[14] generated with the Syfala tool-chain. Hence, future versions will likely allow us to run even more advanced algorithms using the same hardware.

5. FUTURE WORK

We believe that this project has a lot of potential, and we would like to keep working on it in various directions.

First, even though using the Adafruit i2s amp breakout board was a good idea for our initial prototype, we think that developing our own PCB based on the same chips would be way more practical to assemble than making the "amp towers" described in §2.2. This PCB would have a TDM i2s interface and provide 8 amplified speaker outputs. It would be stackable so that systems with an arbitrary number of speakers could be easily implemented.

We also would like to be able to stream more sources to the system without necessarily using analog audio inputs. Hence, we're planning on exploiting the built-in Ethernet port of the FPGA board to send digital audio streams to the system. Since open source is important to us, we're not planning on using proprietary protocols such as Dante for that, and we already started experimenting with open standards such as JackTrip [4].

We're in the process of developing a high-end ultra-low latency (less than $10\mu s$ round-trip) 32 inputs/32 outputs sister board for the Genesys FPGA board. We're very close to have a working prototype. While it will be definitely less "frugal" than the system presented in this paper, it will target more professional applications for spatial audio, especially the ones involving active control. This sister board is based on multiple Analog Devices ADAU1787 audio codecs. We developed a specific breakout board for this codec, hence the sister board is just an assembly of multiple custom ADAU1787 breakout boards plus some electronics, of course. Analog Devices does provide a development board for this codec but it is very expensive: more than 700 USD whereas the codec itself only costs about 10 USD. Since these audio codecs are TDM-compatible up to 16 channels, it would be possible to assemble a board based on one of these breakouts targeting a large number of outputs at a fairly reduced cost.

We think that a similar approach as the one described in this paper could be used to implement a frugal ambisonics microphone based on an FPGA and a series of low-cost (5 USD) Adafruit PDM MEMS Microphone Breakout.[15] The mics would connect directly to the FPGA GPIOs and encoded ambisonics streams would be sent through Ethernet. The built-in ambisonics encoder would be written in FAUST.

Finally, we'd like to offer more advanced WFS algorithms in FAUST through a library.

6. CONCLUSIONS

For less than 800 USD (including speakers), the system presented in this paper provides a 32 channels WFS system programmable in FAUST that can handle multiple sources in parallel. It is open source (except for the Xilinx FPGA toolchain which is proprietary) and easily reproducible. While the target application in this paper is WFS, it can also be used to implement any other spatial audio technique thanks to FAUST which already provides libraries for ambisonics, VBAP, etc.

We believe that the use of FPGAs for real-time audio signal processing applications is reaching a level of maturity and accessibility allowing us to use them for a wide range of DIY applications in the fields of computer music and NIME. Their unmatched performances will likely enable a wide range of new applications in the future.

7. ACKNOWLEDGMENTS

This project has been partially funded by the French ANR (Agence Nationale de la Recherche) through the FAST project[16] (ANR-20-CE38-0001) and the Inria/Stanford Plasma Associate Team.[17]

8. ETHICAL STANDARDS

This project has been entirely funded by French public money through our respective institutions (i.e., Inria and INSA Lyon) as well as the French National Research Agency. It doesn't involve human participants and it has been entirely open-sourced.

9. REFERENCES

[1] J. Ahrens. *Analytic methods of sound field synthesis.* Springer Science & Business Media, 2012.

[2] Popoff, Maxime and Michon, Romain and Risset, Tanguy and Orlarey, Yann and Letz, Stéphane.

[14] In the "world of FPGA," an *Intellectual Property* is the name given to a synthesized (compiled) block on an FPGA.

[15] `https://www.adafruit.com/product/3492`

[16] `https://fast.grame.fr`

[17] `https://team.inria.fr/emeraude/plasma/`

Towards an FPGA-Based Compilation Flow for Ultra-Low Latency Audio Signal Processing. In *SMC-22 - Sound and Music Computing*, Saint-Étienne, France, June 2022.

[3] A. J. Berkhout, D. de Vries, and P. Vogel. Acoustic control by wave field synthesis. *The Journal of the Acoustical Society of America*, 93(5):2764–2778, 1993.

[4] J.-P. Cáceres and C. Chafe. Jacktrip: Under the hood of an engine for network audio. *Journal of New Music Research*, 39(3):183–187, 2010.

[5] M. A. Gerzon. Ambisonics in multichannel broadcasting and video. *Journal of the Audio Engineering Society*, 33(11):859–871, 1985.

[6] F. Lopez-Lezcano. Searching for the grail. *Computer Music Journal*, 40(4):91–103, 2016.

[7] Y. Orlarey, S. Letz, and D. Fober. *New Computational Paradigms for Computer Music*, chapter "Faust: an Efficient Functional Approach to DSP Programming". Delatour, Paris, France, 2009.

[8] V. Pulkki. Virtual sound source positioning using vector base amplitude panning. *Journal of the audio engineering society*, 45(6):456–466, 1997.

[9] T. Reussner, C. Sladeczek, M. Rath, S. Brix, K. Preidl, and H. Scheck. Audio network-based massive multichannel loudspeaker system for flexible use in spatial audio research. *Journal of the Audio Engineering Society*, 61(4):235–245, april 2013.

[10] J. O. Smith. A spatial sampling approach to wave field synthesis: PBAP and huygens arrays. *CoRR*, abs/1911.07575, 2019.

[11] D. Theodoropoulos, C. B. Ciobanu, and G. Kuzmanov. Wave field synthesis for 3d audio: Architectural prospectives. CF '09, page 127–136, New York, NY, USA, 2009. Association for Computing Machinery.

[12] D. Theodoropoulos, G. Kuzmanov, and G. Gaydadjiev. Multi-core platforms for beamforming and wave field synthesis. *IEEE Transactions on Multimedia*, 13(2):235–245, 2011.

[13] T. C. Vannoy. Enabling rapid prototyping of audio signal processing systems using system-on-chip field programmable gate arrays. Master PhD, 2020.

[14] M. Verstraelen, J. Kuper, and G. J. Smit. Declaratively Programmable Ultra Low-Latency Audio Effects Processing on FPGA. In *DAFx*, pages 263–270, 2014.

[15] T. Ziemer. *Wave Field Synthesis*, pages 329–347. Springer Berlin Heidelberg, Berlin, Heidelberg, 2018.