Ethernet Real-time Audio Transmission to FPGA

Pierre Cochard
INSA Lyon, Inria, CITI, EA3720,
69621 Villeurbanne,
Pierre.Cochard@insa-lyon.fr

Jurek Weber
Fakultät IV, Technische Universität Berlin
Germany,
Jurek.Weber@rwth-aachen.de

Romain Michon
Inria, INSA Lyon, CITI, EA3720,
69621 Villeurbanne,
Romain.Michon@inria.fr

Tanguy Risset
INSA Lyon, Inria, CITI, EA3720,
69621 Villeurbanne,
Tanguy.Risset@insa-lyon.fr

Stéphane Letz
Grame, CITI, EA3720,
69621 Villeurbanne,
letz@grame.fr

Abstract—Real-time audio transmission over Ethernet has become a standard in recording studios, concert halls, etc., thanks to protocols such as Dante, AVB, and more generally AES67. Audio over Ethernet solves many issues by allowing for the transmission of a large number of digital audio streams over fairly long distances in a standardized way. On the other hand, it introduces challenging issues like jitter and distributed clock synchronicity. While the aforementioned standards deal with these kinds of problems, they may be hard to deploy when audio diffusion is carried out using dedicated/custom hardware, such as an FPGA. This paper proposes a dedicated and simple protocol for transmitting Ethernet audio streams on a LAN to an FPGA acting as an audio DSP platform. This protocol, combined with the FPGA's ability to handle a large number of audio streams, provides a cost-effective way to deploy spatial audio systems. We demonstrate this protocol on a 32-speakers WFS system controlled by an AMD/Xilinx Zybo Z7 FPGA development board.

I. Introduction

Transmitting digital audio streams over Ethernet – mostly using UDP – has been a topic of interest for many years to become a standard in the audio industry nowadays. Depending on the kind of applications: audio streaming, spatial audio rendering [1], live networked audio production [2], etc., priority might be given to audio quality (sample size, sampling rate, no packet loss), synchronicity (no sample lost), latency (fast round trip time), or cost (used by hobbyist).

The work presented in this paper is motivated by the use of audio-over-Ethernet transmission in the context of an FPGA-based system. Recently, FPGAs designed for audio applications have garnered significant attention [3]–[5] because of their unique performances for specific applications such as low latency [6] and spatial audio [7]. The compilation of DSP kernel algorithms on FPGAs has been addressed recently [8], but interfacing such system with audio diffusion software is still an open issue. Despite the inclusion of a system-on-chip (SoC) with a powerful processor capable of running an operating system on modern FPGA boards [3], [9], these systems are not intended for heavy computational tasks and are primarily oriented towards low-cost diffusion. Therefore, there is a clear need for a simple protocol to stream audio files over Ethernet to FPGA-based audio systems.

This paper describes a proposal for such a protocol, using the JACK Audio Connection Kit Server as an interface. It targets an FPGA-based system providing real-time audio DSP as well as multichannel sound diffusion capabilities. The protocol is open source and has been implemented in the online distribution of Syfala.[1] It has been used in the context of a Wave Field Synthesis (WFS) system (see Section IV-C). The solution presented here it relies only on JACK, Syfala, and the Rust compiler, all of which are open-source. It is reasonably simple to deploy and it constitutes a straightforward way to implement spatial audio systems in small/medium size room.

The system includes a server running on a "generic" computer that is able to connect to JACK and distribute audio from and to multiple FPGAs via the local Ethernet network. It addresses synchronicity issues by using a very basic resampling algorithm which is adapted to network jitter occurring on a LAN. Section II provides the precise problem formulation and the associated state of the art. The protocol is described in Section III. Performance metrics, including latency and clock drift correction, are presented in Section IV.

II. Problem Formulation and State of the Art

The development of audio networking solutions has been driven by an increasing demand, particularly in professional audio and broadcasting environments. CobraNet, [10] in the 1990s, is often credited as one of the pioneering technologies for audio-over-Ethernet. EtherSound and Dante are more recent commercial audio-over-Ethernet solutions. Dante has gained significant popularity in the professional audio industry. Simultaneously, various open standards for audio-over-IP have emerged. The most notable one is AES67 and is provided by the Audio Engineering Society (AES). AES67 aims to provide interoperability between different audio-over-IP (AoIP) systems, allowing audio streams to be shared seamlessly across different platforms and devices. Finally, the Audio Video Bridging Standard (AVB, 2011) was developed by the Institute of Electrical and Electronics Engineers (IEEE) to enable the reliable and time-synchronized delivery of audio and video over Ethernet networks. In general the problem of networked

[1] https://github.com/inria-emeraude/syfala/

music performance is quite complex, a good survey can be found here [11].

The main goal of the current paper is to enable audio over Ethernet transmission on FPGA-based platforms through a lightweight protocol that does not include a complete official standard. A fair amount of work has already been conducted towards simpler implementations using basic TCP/UDP protocols. The SoundWire project, initiated by the Center for Computer Research in Music and Acoustics (CCRMA, Stanford University) [12], employs TCP to minimize packet loss or UDP in other audio diffusion scenarios. The JACK Audio Connection Kit has inspired many developments [13], such as JackTrip[2] [14] for network music performance over the Internet, NetJack[3] [15], or Zita-njbridge.[4].

First contributions to audio on FPGA date back from the 2000s and have continuously occurred since then as FPGAs offer very good performances in terms of latency, throughput, and multi-channel capabilities. Many project have been proposed using dedicated hand-made FPGA designs such as: digital drum kits [16], audio effect generators [17], [18]. The advent of SoC[5] integrated on recent FPGA platforms opened the door to a complete standalone platform able to handle audio control and processing [19]–[21].

It is worth noting that FPGAs are increasingly used in the audio industry. Novation,[6] Antelope Audio,[7] UDO audio,[8] and futur3soundz[9] all offer products integrating FPGAs. These products are, of course, not meant to be integrated in an Ethernet-based open-source broadcasting system.

The main approach for deploying real-time audio applications on FPGA is based on manual hardware design [4], [18]. Few publications exhibit an automatics process to map audio DSP on FPGA: Verstraelen et. al. propose a programmable parallel machine on FPGA [22], Vannoy et al. proposed an open-source IP-based design using MathWorks' HDL Coder [5]. An interesting work recently presented by Politecnico de Torino [23] targets low latency networked music performances using an embedded core on an FPGA. None of these work resulted in a usable open-source automatic flow from audio to FPGA. Spreading the use of FPGA for audio necessitates a more systematic way of mapping audio programs on FPGA, or – to name it differently – a "real" audio to FPGA compilation flow.

Syfala is the first real compiler generating FPGA design bitstreams [8] from a high-level audio DSP language: Faust.[10] It was notably able to reach the smallest analog-to-analog latency ever reported [6]. With the addition of embedded Linux capabilities [3], it can drive a large number of audio codecs, potentially exceeding 100, from a single FPGA board. The Syfala compiler is open-source,[11] together with a set of open-hardware boards that help specialize its use to various kinds of applications [7]. However, to take full advantage of its capabilities, one must be able to stream audio signals directly to the FPGA. USB could be used to stream signals, however it cannot be use to broacast of multicast signals to several FPGA boards, hence the Ethernet port present on all FPGA boards appeared to be the natural choice for this.

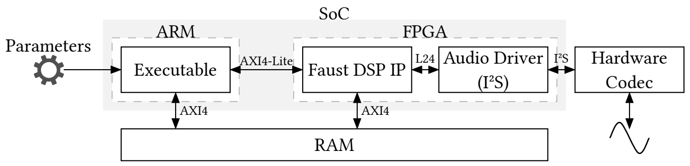

Figure 1. Original Architecture (i.e., without Ethernet audio streaming) of the audio program compiled by Syfala [8]

The system view of an FPGA design resulting from Syfala is depicted in Fig. 1. The entire DSP application operates on the FPGA SoC (Faust DSP IP on Fig. 1). The control part of the application runs on the ARM subsystem (Processing System in Xilinx SoCs) within an embedded Linux environment. The DSP kernel component of the application runs on the FPGA, it is referred to as the DSP IP (note that in the context of hardware design and throughout all this paper, IP stands for Intellectual Property and designates a dedicated hardware component). The embedded Linux distribution enables the control of the DSP IP through Ethernet using OSC (Open Sound Control) or an HTTP-based protocol, typically from the host computer or over the network.

The problem we aim to address here is the integration of Ethernet audio transmission in the Syfala toolchain. The principle of this integration is illustrated in Fig. 2: an audio server sends audio streams over the network, and these streams are directly processed by the FPGA, which runs a compiled Syfala program. To achieve this, the DSP IP must be adapted to receive streams from the network (not just from the audio codec), and a client/server protocol needs to be established to handle streams over the network, but accessing audio streams over Ethernet requires a much more optimized mechanism.

It is worth noting that a similar approach could be implemented using any real-time operating systems running on an embedded system [24], [25] without necessarily targeting FPGA-based platforms. In that regard, the work presented in this paper can be seen as an FPGA adaptation of [26] to which we added linear interpolation to smooth the sample loss.

In the following sections, we describe the technical implementation of this extension. Subsequently, we demonstrate its application in the context of WFS.

III. Technical Decisions and Implementation

AVB, Dante and AES67 were not chosen for our implementation because they rely on PTP (Precision Time Protocol)

[2]https://jacktrip.github.io/jacktrip/
[3]https://netjack.sourceforge.net//
[4]https://kokkinizita.linuxaudio.org/linuxaudio/
[5]SoC: System on Chip
[6]https://novationmusic.com/en/synths/summit
[7]https://en.antelopeaudio.com/
[8]https://www.udo-audio.com/#introduction
[9]https://www.futur3soundz.com/
[10]https://faust.grame.fr/
[11]https://github.com/inria-emeraude/syfala/

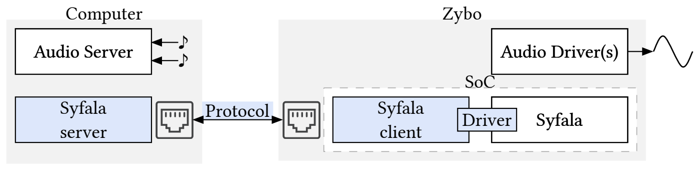

Figure 2. Targeted architecture coupling the Syfala toolchain and Ethernet transmission to the Zybo Xilinx FPGA board (blue boxes are described in this paper).

for synchronizing clocks which implies the use of specific hardware. Ideally, when using one of these protocols, the whole networking equipment chain would have to support PTP, which usually results in higher hardware cost.

Our system relies on the JACK API, which is very standard on Linux for sharing audio between different applications. The Syfala server connects to the JACK API and enables a transmission of audio streams using a protocol that encapsulates audio data in UDP packets. The audio transmission is bidirectional while supporting multiple clients (FPGAs) per server.

The Syfala client and server were developed in Rust, which is a new approach, especially in the field of embedded systems. In our use case, one of Rust's key advantages is its strong support for asynchronous programming using the Tokio runtime[12]. Rust achieves performances comparable to that of C while significantly simplifying the development of safe concurrent programs. The encoding, decoding, and transfer of data between multiple TCP and UDP protocols, as well as interactions with the operating system in a multithreaded environment, can be performed safely and with remarkable ease.

Additionally, the big variety of crates[13] increased productivity noticeably, since many ideas could be tested and debugged fast without implementing complicated algorithms (e.g., resampling) by hand.

This section explains how the proposed architecture has been implemented with two bi-directional transmissions: one between the audio server on the computer and the ARM chip on a Digilent Zybo Z7 development board (illustrated in Fig. 3), and the other between the ARM chip and the Faust DSP kernel on the FPGA (illustrated in Fig. 5).

A. Custom Transmission Protocol

An overview of our custom transmission protocol is depicted in Fig. 3. The streaming server (PC running the audio server) opens a TCP server on port 6910 and listens for incoming connections. The streaming client (on the FPGA processing system) connects to the TCP server, and transmits information about the DSP program, such as its number of input and output channels. The server responds by sending other configuration details to the client, such as latency and buffer sizes. UDP audio streaming can then start: the client sends/receives audio data in UDP packets to/from the server.

[12]https://tokio.rs/

[13]Crates are the Rust equivalent to a library

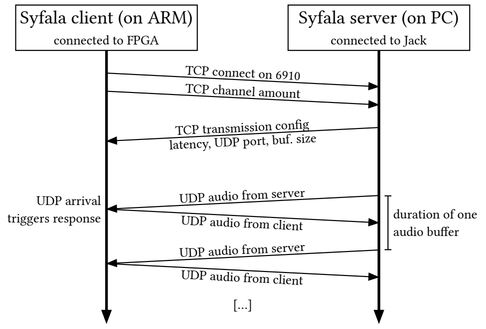

Figure 3. Proposed custom client/server protocol – exchanging audio data between the audio server and the ARM processor on the FPGA board's SoC.

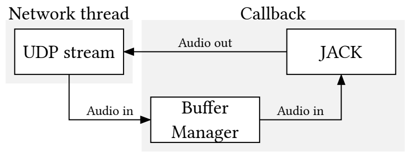

Figure 4. Server-side implementation of the protocol using two tasks.

Both TCP and UDP protocol encodings are implemented using the Rust crate Bincode[14], which enables straightforward (de-)serialization of Rust data structures. This approach simplifies cross-platform protocol development and can be easily adapted to different encoding strategies, as the encoding and decoding process requires a single line of code.

The TCP connection is also used for monitoring, as it informs the computer about clock drift, general transmission statistics, as well as protocol errors, such as packets loss.

B. Protocol Implementation: Server-Side and Client-Side

The server's implementation is shown on Fig. 4. The server-side implementation of the protocol uses multiple tasks:[15]

- A single UDP task handling the UDP packets of all clients
- One JACK task per Syfala client (acting like a virtual sound card)
- One TCP task per Syfala client (not shown, only responsible for monitoring and configuring)

Each time a Syfala client connects to the remote server, the host application creates a new (virtual) JACK client with its own audio processing callback, which is executed each time an audio buffer is ready. In this callback, audio data is passed to the UDP task, dedicated to stream the audio over the network. If the client is supposed to send audio back to the server, a Ring Buffer, as well as a Buffer Manager (for synchronization), explained further, are used.

[14]https://docs.rs/bincode/1.3.3/bincode/

[15]Lightweight thread-like unit of execution managed by Tokio: https://docs.rs/tokio/1.38.0/tokio/task/

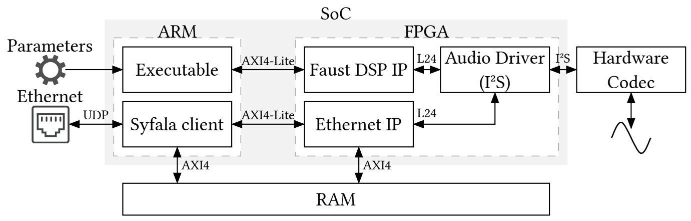

Figure 5. Client-side architecture, composed of two major components: the Syfala client and the Ethernet IP.

The client-side implementation (depicted in Fig. 5) is more intricate, as it incorporates two distinct interfaces: the network interface with the remote server, and the interface with the FPGA DSP IP. Therefore, it has effectively been divided into two components: the Syfala Network Client Application, running as an ARM executable and depicted in §III-D, and the Ethernet IP running on the FPGA and depicted in §III-C.

C. Client-Side Ethernet IP

The Syfala toolchain already supports multiple I/O audio codecs [8]. Therefore, we chose to treat the Ethernet interface for the DSP IP as if it was another audio driver, i.e., using the Integrated Interchip Sound (I^2S) protocol, as shown on the right side of Fig. 5. The communication between the Syfala client and the Ethernet IP involves two ARM proprietary on-chip communication bus protocols:

- An Advanced eXtensible Interface (AXI4) bus, used for accessing DDR memory simultaneously from the client application and the Ethernet IP, in order to share audio data using a ring-buffer. The handling of this system is explained in Section III-D1.
- An AXI4-Lite bus, a lighter protocol, subset of AXI4, used to send control information from the Syfala client to the Ethernet IP, such as the number of channels at initialization, or the ring-buffer's read and write pointer at each sample.

This Ethernet IP was developed in C++ and synthesized using High-Level Synthesis (HLS) with Vitis_HLS, allowing us to leverage the AXI4 protocol from the start, by providing a simple C abstraction where RAM can be accessed through regular pointers.

HLS made the development of the IP quite straightforward: the reading and writing of audio samples is implemented in a C entry-point function named `eth_audio`. The prototype of the `eth_audio` is shown on Fig. 6, together with the Vitis_HLS pragmas used to synthesize it as well as the access to DDR RAM via the AXI4 bus (i.e., `m_axi`). As input and output streams are stored in different memory regions, there are two accesses (`ram_in` and `ram_out`). On the other side of the Ethernet IP is the I^2S transceiver with a configurable number of input and output streams (`audio_in` and `audio_out` arrays). The other function parameters are variables shared with the ARM processor using the AXI4-

```
#define sy_ap_int ap_fixed<SYFALA_SAMPLE_WIDTH, 0, AP_RND_CONV, AP_SAT>

void eth_audio (
      float* ram_in,
      float* ram_out,
      int eth_ok,
      int audio_in_len,
      int audio_out_len,
      int* audio_in_channels,
      int* audio_out_channels,
      int* audio_in_r,
      int* audio_in_w,
      int* audio_out_r,
      int* audio_out_w,
      sy_ap_int audio_in[SYFALA_ETHERNET_NCHANNELS_FROM_I2S],
      sy_ap_int audio_out[SYFALA_ETHERNET_NCHANNELS_TO_I2S],
      int* status,
      int* read_clk,
      int* read_debug,
      int* write_debug
) {
#pragma HLS array_partition variable=audio_in type=complete
#pragma HLS array_partition variable=audio_out type=complete
#pragma HLS INTERFACE m_axi port=ram_in latency=30 bundle=ram
#pragma HLS INTERFACE m_axi port=ram_out latency=30 bundle=ram
#pragma HLS INTERFACE s_axilite port=eth_ok
#pragma HLS INTERFACE s_axilite port=audio_in_len
[...]
#pragma HLS INTERFACE s_axilite port=read_clk
[...]
```

Figure 6. Prototype of the C function used for the High Level Synthesis of the Ethernet IP together with the Vitis_hls associated pragmas specifying the interface used for each parameter.

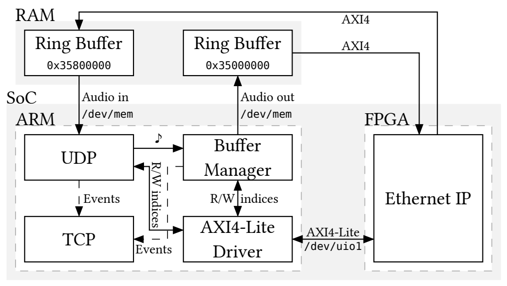

Figure 7. Syfala client application running on the ARM processor and its interfacing with DDR memory (the DSP IP and the I^2S IP shown on Fig. 5 have not been depicted in the FPGA block).

Lite bus (i.e., `s_axilite`) for coherent access to the data in DDR RAM from both sides.

This `eth_audio` function is triggered by the I^2S each time a new sample is ready. Hence it performs a sample-by-sample computation. Writing the Ethernet IP in C++ allows for a much easier integration of the AXI4 Stack without having to implement the protocol in VHDL. Once synthesized to VHDL code, this IP can be integrated to the final Syfala block-design and connected to other modules, such as the I^2S or the AXI interconnect system as shown on Fig. 5.

D. Client-Side Syfala Client

The Syfala client application is shown on Fig. 7. It serves as the interface with the remote server on the network, and also manages synchronization between the JACK Server and the Ethernet IP. Upon establishing a TCP connection, it sends and receives audio buffers to/from the server using UDP packets. It then copies/reads this data to/from the ring-buffers in DDR memory, which is accessed through a standard Linux file device (`/dev/mem`, the Linux memory mapping is explained below).

The read and write indices of the Ring Buffer are synchronized at sample-rate with the Ethernet IP using the AXI4-

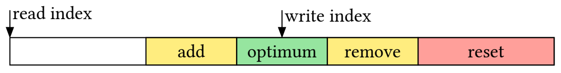

Figure 8. Read and write indices in the ring buffer used to transmit audio data, and buffer zone used to perform resampling to correct clock drift.

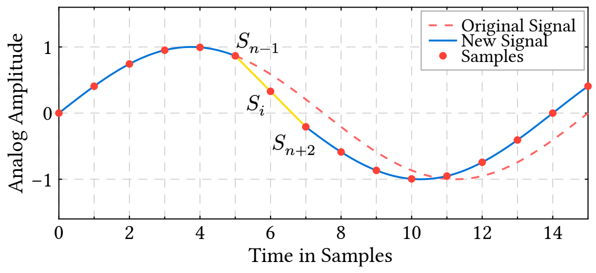

Figure 9. Linear interpolation used to correct clock drift.

Lite protocol, which is interfaced as another Linux file device (`/dev/uio1`). By comparing the read and write indices, the client can determine the fill level of the ring buffer and perform buffer management.

1) Buffer Manager and Resampling: Clock drift describes the subtle differences in clock frequencies between the sender and the receiver clocks. This can lead to an excess or shortage of samples at the receiving end. Furthermore, slight variations in the time it takes for packets to reach the receiver are called network jitter.

In our system, network jitter is handled using buffering and clock drift compensation is carried out by inserting missing or removing excessive samples. This technique is based on the approach shown in [26] and could be seen as a very basic form of resampling.

We use a Ring Buffer with two indices: read and write. The buffer manager operates on the receiving side of the transmissions, facilitating the reception of unmodified audio data and saving it to a file using the same protocol. This feature enables us to use the Syfala client application for recording input streams and debugging the protocol.

At the beginning of the reception, the algorithm uses the first packets to evaluate the median of the packet's arrival times. This step is necessary as the initial packets typically take longer to arrive than the subsequent ones. Once the algorithm establishes a stable median, it begins adjusting the buffer by removing or adding samples.

The buffer manager evaluates the number of samples in the ring buffer as shown in Fig. 8. If the buffer is not filled enough, a sample is inserted. If the buffer is too full, a sample is dropped.

An example of sample dropping is illustrated in Fig. 9. Suppose sample S_n needs to be removed. The resampling algorithm calculates the interpolation between S_{n-1} and S_{n+2} and uses this result as the new sample S_i instead of S_{n+1}. It's worth noting that this interpolation algorithm could be further improved in the future and replaced by a proper resampling

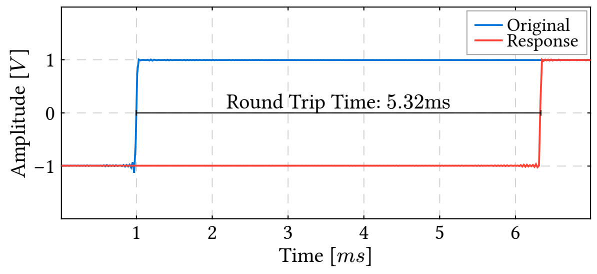

Figure 10. Round-trip latency measurements for 64 samples buffer.

strategy.

2) Memory Management in Embedded Linux: As mentioned earlier, accessing memory from the Ethernet IP is straightforward, thanks to Vitis_HLS. However, from a Linux point of view, this memory must not be included in the virtual memory manager, since memory addresses used in the Syfala client must be physical addresses, not virtual addresses. This is achieved by configuring the device-tree, as explained in [3]. This configuration allows for the sharing of the aforementioned ring-buffers. One of the ring-buffers is used for audio transmission from the Ethernet IP to the ARM chip, and the other is used for the reverse direction.

The Linux AXI4-Lite device (mounted on `/dev/uio1`) is used to transmit the base pointer of the ring buffers at initialization and also to exchange read-and-write pointers between the FPGA and the ARM at each sample.

IV. Results and Performances

The system presented above is now implemented in the Syfala toolchain and is available on the Syfala GitHub.[16] It is currently used by two research team in the world for prototyping accessible spatial audio systems. This section presents the first results obtained in a simple setup: the server (host) and the FPGA board being connected to the same layer 2 switch. The server operated on a laptop running a Linux distribution and employed JACK as the audio server, configured with a buffer size of 64 samples. The audio signals were connected using a Qt frontend for JACK. The size of the buffers (here 64) is used to compensate for Jitter, which was very low after initialisation, as we operate on a simple LAN.

A. Round-Trip Latency

For live performances, one of the most important factors is latency. We measured the round-trip latency using a simple bypass DSP program on the FPGA. A simple signal switching from -1 to 1 is sent from the PC to the FPGA through Ethernet, the DSP IP is simple loopback and the signal is again sent on Ethernet to the PC. Fig. 10 shows the round-trip latency time: approximately 5.32 ms.

[16]https://github.com/inria-emeraude/syfala

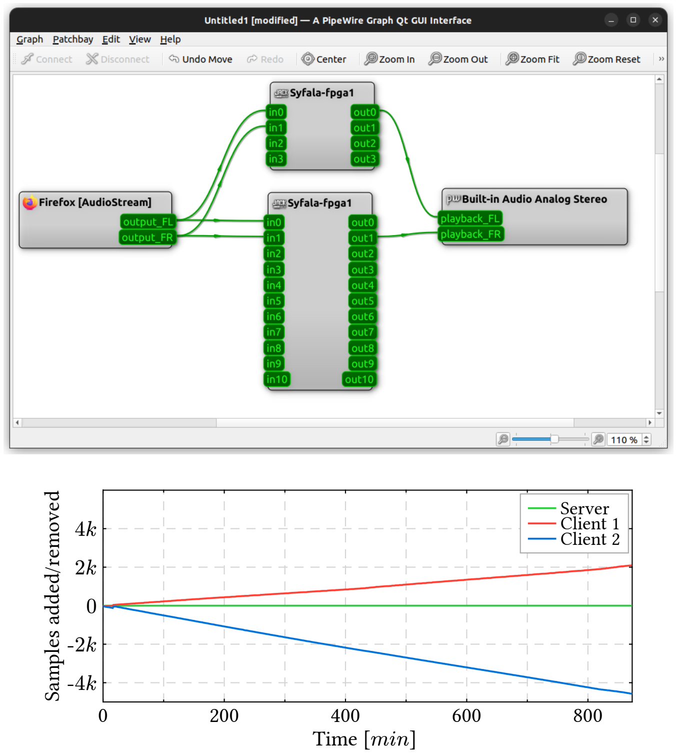

Figure 11. The audio server connection graph (top) depicting two FPGAs connected to each other to measure clock drift (bottom) between the server and the two FPGAs.

The 64 sample buffers used in both direction introduce a latency of T_B:

$$T_B = \frac{64}{48000} \simeq 1.33ms$$

Transmitting one frame from the server to the FPGA uses two such buffers (one on the server, one on the client), hence a round trip uses four of them. As $4 \cdot T_B \simeq 5.32,$ms, this highlights again the very low latency added by the hardware executing the DSP computation (exactly $20.8\mu s$, one sample at 48kHz).

As expected, our approach does not introduce a smaller or bigger latency than other approaches: the latency is in general linked to the network jitter on which we have no control. The advantage of our approach is to rely on an FPGA which can easily handle a large number of channels and has important computing capabilities.

B. Clock Drift Correction and Scalability

It is important to assess the scalability of our system. If a very large number of speakers is necessary, we may need several FPGA boards. We tested the system with two Zybo boards connected to the same server, allowing for up to 24 channels in both directions (see the top of Fig. 11 for a specific configuration). Two Zybo boards running the same DSP IP were connected to the same server to measure the clock drift. The same audio content was streamed to both. The system was tested for continuous transmission for 13 hours without

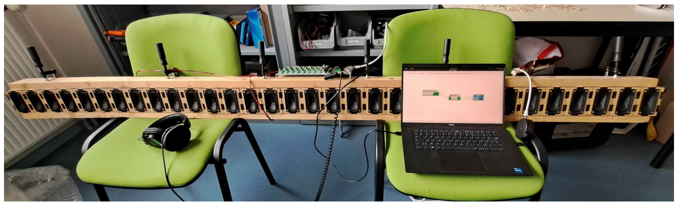

Figure 12. The WFS diffusion system [27] connected to the FPGA Zybo board.

crashes. During this test, we closely monitored the number of samples that were removed or added (which corresponds exactly to the evaluation of the number of audio samples received versus the number of audio samples consumed). The results in Fig. 11 show the expected clock drift of the Zybo.

The clock drift observed was approximately 0.812 ppm (Part Per Million) for one Zybo, and -1.82 ppm for the other Zybo. The clock drift was regularly compensated, by inserting and deleting samples: the latency remained constant. The ARM CPU usage stayed constant and low enough for other programs to run simultaneously.

The only noticeable flaw concerns synchronization, where the use of linear interpolation between removed or added samples distorts the produced sound. A soft clicking is sometimes audible when playing back a pure sine wave at a higher frequency (+400Hz), though it is not noticeable on non-sinusoidal streams (e.g., music, speech). Switching to a higher degree of interpolation could reduce this issue further. The incorporation of the Rubato[17] resampling library resulted in slight improvements of the audio quality, but the marginal benefits were insufficient to justify the associated increase in CPU usage. The addition of a more complex resampling strategy would, of course, limit the number of channels that could be transferred.

C. Full 32 Channel WFS System

The system was demonstrated at a public meeting: Tech and Fest in Grenoble.[18] The application used the “frugal” 32 loudspeakers WFS system presented in [27] for rendering a music band performance involving 4 instruments. The position of each instrument was sent to the DSP IP using the Open Sound Control (OSC) protocol through Ethernet, using a web interface which is not shown here. Depending on the position on the instruments, these 4 channels were processed by the FPGA differently for each of the 32 speakers to render the spatial audio effect.

One of the main strengths of this system is its cost. If speakers are assembled “by hand” as presented in Fig. 12, the whole 32 speakers WFS system (excluding the client-side PC) costs less than 100 euros. Additionally, thanks to the stackable digital amplifiers PCB boards introduced in [7], more speakers can be added with a simple linear cost increase. This would

[17]https://github.com/HEnquist/rubato
[18]https://www.tech-fest.fr/

not be the case with a more standard system where cost would increase exponentially as more speakers are added.

Multiple iterations of this system were built, including one at the Center for Computer Research in Music and Acoustics at Stanford University as mentioned above.

V. Conclusion and Future Works

We presented the first prototype of an Ethernet real-time audio transmission protocol for an FPGA acting as an audio processor/interface. The main advantage of this prototype compared to existing commercial systems is its very low cost and its open-source nature. The low latency of the FPGA routing might also be a decisive advantage but this has to be confirmed on specific applications.

The round trip latency measured on a simple local area network was about 5ms, while the clock drift measured was approximately 2ppm between two different FPGA boards. The system has been demonstrated in many setups and places and can be built from a complete open-source distribution (except, of course, for Xilinx tools which are not open-source).

This prototype paves the way to many applications in the domain of spatial audio, in particular. Whether it would be using ambisonic or WFS, the "frugal" spatial audio diffusion could be easily deployed in various contexts, such as museums, virtual reality rooms, cars, homes and participate in the deployment of virtual acoustics in the society.

In the near future, we shall deploy an industrial application which is targeted towards real-time noise canceling using the FXLMS algorithm such as the one presented in [28]. The proposed prototype is open source and fully accessible on GitHub.[19]

[19]https://github.com/inria-emeraude/syfala/

References

[1] S. Spors, R. Rabenstein, and J. Ahrens, "The theory of wave field synthesis revisited," in 124th Audio Engineering Society Convention 2008, vol. 1, 2008, pp. 413–431.

[2] C. Chafe, "I am streaming in a room," Frontiers in Digital Humanities, vol. 5, 2018.

[3] P. Cochard, M. Popoff, A. Fraboulet, T. Risset, S. Letz, and R. Michon, "A programmable linux-based FPGA platform for audio DSP," in Proceedings of the 2023 Sound and Music Computing Conference (SMC-23), 2023, pp. 110–116.

[4] C. Wegener, S. Stang, and M. Neupert, "FPGA-accelerated real-time audio in pure data," in Proc. Int. Conf. in Sound and Music Computing, SMC-22, 2022.

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

[6] M. Popoff, R. Michon, T. Risset, Y. Orlarey, and S. Letz, "Towards an FPGA-Based Compilation Flow for Ultra-Low Latency Audio Signal Processing," in Proceeding of the 2022 Sound and Music Computing, (SMC-22), Saint-Étienne, France, Jun. 2022.

[7] M. Popoff, R. Michon, and T. Risset, "Enabling Affordable and Scalable Audio Spatialization With Multichannel Audio Expansion Boards for FPGA," in Proceedings of the 2024 Sound and Music Computing Conference, (SMC-24), Porto (Portugal), July 2024.

[8] M. Popoff, R. Michon, T. Risset, P. Cochard, S. Letz, Y. Orlarey, and F. de Dinechin, "Audio DSP to FPGA Compilation: The Syfala Toolchain Approach," Univ Lyon, INSA Lyon, Inria, CITI, Grame, Emeraude, Tech. Rep. RR-9507, May 2023. [Online]. Available: https://inria.hal.science/hal-04099135

[9] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stewart, The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc. Glasgow, GBR: Strathclyde Academic Media, 2014.

[10] B. Klinkradt and R. Foss, "A comparative study of mLAN and CobraNet technologies and their use in the sound installation industry," in Audio Engineering Society Convention 114, Mar 2003.

[11] C. Rottondi, C. Chafe, C. Allocchio, and A. Sarti, "An overview on networked music performance technologies," IEEE Access, vol. 4, pp. 8823–8843, 2016.

[12] C. Chafe, S. Wilson, R. Leistikow, D. Chisholm, and G. Scavone, "A Simplified Approach to High Quality Music and Sound Over IP," in Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00), 2000.

[13] F. Lopez-Lezcano, "From Jack to UDP packets to sound, and back," in Linux Audio Conference 2012, CCRMA, Stanford University. CCRMA, Stanford University: CCRMA, Stanford University, 04/2012 2012.

[14] 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, Sep. 2010.

[15] A. Carôt, T. Hohn, and C. Werner, "Netjack – Remote music collaboration with electronic sequencers on the Internet," in Linux Audio Conference, Jan. 2009.

[16] K. M. Jadhao and G. Singh Patel, "Hardware Architecture of Digital Drum Kit Using FPGA," in 2020 IEEE International Conference on Advent Trends in Multidisciplinary Research and Innovation (ICATMRI). Buldhana, India: IEEE, Dec. 2020, pp. 1–4.

[17] S. R. Chhetri, B. Poudel, S. Ghimire, S. Shresthamali, and D. K. Sharma, "Implementation of Audio Effect Generator in FPGA," Nepal Journal of Science and Technology, vol. 15, no. 1, pp. 89–98, Feb. 2015.

[18] C. Dragoi, C. Anghel, C. Stanciu, and C. Paleologu, "Efficient FPGA Implementation of Classic Audio Effects," in 2021 13th International Conference on Electronics, Computers and Artificial Intelligence (ECAI). Pitesti, Romania: IEEE, Jul. 2021.

[19] D. Cannon, T. Fang, and J. Saniie, "Modular Delay Audio Effect System on FPGA," in 2022 IEEE International Conference on Electro Information Technology (eIT), May 2022, pp. 248–251.

[20] A. S. Deulkar and N. R. Kolhare, "FPGA implementation of audio and video processing based on Zedboard," in 2020 International Conference on Smart Innovations in Design, Environment, Management, Planning and Computing (ICSIDEMPC). Aurangabad, India: IEEE, Oct. 2020, pp. 305–310.

[21] K. Vaca, M. M. Jefferies, and X. Yang, "An Open Audio Processing Platform with Zync FPGA," in 2019 IEEE International Symposium on Measurement and Control in Robotics (ISMCR), Sep. 2019, pp. D1–2–1–D1–2–6.

[22] M. Verstraelen, J. Kuper, and G. J. Smit, "Declaratively Programmable Ultra Low-Latency Audio Effects Processing on FPGA." in Proceeding of the International Conference on Digital Audio Effects (DAFx-14), 2014.

[23] D. Bert, N. Domini, R. Peloso, L. Severi, M. Sacchetto, A. Bianco, and C. Rottondi, "FPGA-based low-latency audio coprocessor for networked music performance," in 2023 4th International Symposium on the Internet of Sounds, 2023, pp. 1–8.

[24] L. Turchet and C. Fischione, "Elk audio os: An open source operating system for the internet of musical things," ACM Trans. Internet Things, vol. 2, no. 2, mar 2021.

[25] L. Vignati, S. Zambon, and L. Turchet, "A comparison of real-time linux-based architectures for embedded musical applications," Journal of the Audio Engineering Society, vol. 70, pp. 83–93, 01 2021.

[26] D. Fober, "Audio Cards Clock Skew Compensation over a Local Network," GRAME, Technical Report, 2002.

[27] R. Michon, J. Bizien, M. Popoff, and T. Risset, "Making Frugal Spatial Audio Systems Using Field-Programmable Gate Arrays," in Proceedings of the 2023 New Interfaces for Musical Expression Conference, Mexico City, Mexico, May 2023.

[28] L. Alexandre, P. Lecomte, and M.-A. Galland, "Experimental Active Noise Control Using Faust On FPGA: Comparison Between A Multi-Point and Spherical Harmonics Method," in Forum Acusticum 2023 - 10th Convention of the European Acoustics Association. Torino, Italy: European Acoustics Association, Sep. 2023.