Open Source Ethernet Real-time Audio Transmission to FPGA

Pierre Cochard, Jurek Weber, Romain Michon, Tanguy Risset, Stephane Letz

RESEARCH REPORT
N° 9542
fev 2024
Project-Teams Emeraude

ISSN 0249-6399 ISRN INRIA/RR--9542--FR+ENG

Open Source Ethernet Real-time Audio Transmission to FPGA

Pierre Cochard*, Jurek Weber†, Romain Michon‡, Tanguy Risset§, Stephane Letz¶

Project-Teams Emeraude

Research Report n° 9542 — fev 2024 — 16 pages

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 while providing a reasonable latency and synchronicity. On the other hand, it introduces challenging issues such as jitter and distributed clock synchronicity. While the aforementioned standards deal with these kinds of problems, they may not be directly applicable 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.

Key-words: HLS, FPGA, Audio DSP, Ethernet

* *Inria, INSA Lyon, CITI, EA3720*
† Fakultät IV, Technische Universität Berlin, Germany
‡ *Inria, INSA Lyon, CITI, EA3720*
§ *INSA Lyon, Inria, CITI, EA3720* 69621 Villeurbanne, France
¶ *GRAME-CNCM, INSA Lyon, Inria, CITI, EA3720* 69621 Villeurbanne, France

RESEARCH CENTRE LYON

Bâtiment CEI-2, Campus La Doua
56 Bd Niels Bohr, 69100 Villeurbanne

Transmission audio Open Source en temps réel via Ethernet vers FPGA

Résumé : La transmission audio en temps réel via Ethernet est devenue une norme dans les studios d'enregistrement, les salles de concert, etc., grâce à des protocoles tels que Dante, AVB, et plus généralement AES67. L'audio sur Ethernet résout de nombreux problèmes en permettant la transmission d'un grand nombre de flux audio numériques sur des distances assez longues de manière normalisée. En revanche, elle introduit des problèmes complexes tels que la gigue et la synchronicité des horloges distribuées. Bien que les normes mentionnées précédemment traitent de ces types de problèmes, elles peuvent ne pas être directement applicables lorsque la diffusion audio est réalisée à l'aide de matériel dédié, tel qu'un FPGA.

Cet article propose un protocole dédié et simple pour la transmission de flux audio Ethernet sur un réseau local (LAN) vers un FPGA agissant comme une carte son. Ce protocole, combiné à la capacité du FPGA à gérer un grand nombre de flux audio, offre un dispositif faible coût pour déployer des systèmes audio spatialisés. Nous validons l'utilisation de ce protocole sur un système WFS de 32 haut-parleurs contrôlé par une carte de développement FPGA AMD/Xilinx Zybo Z7.

Mots-clés : HLS, FPGA, Audio DSP, Ethernet

Contents

1 Introduction **4**

2 Problem formulation and state of the art **4**

3 Technical choices and implementation **5**
3.1 Custom Transmission Protocol 6
3.2 Server-side implementation 6
3.3 Client-Side Implementation 8
3.3.1 Ethernet IP 8
3.3.2 Syfala-client 9
3.3.3 Buffer Manager and Resampling 9
3.3.4 Memory Management in Embedded Linux 10

4 Results and Performances **11**
4.1 Round-trip Latency 12
4.2 Clock Drift Correction and Scalability 12
4.3 Full 32 Channel WFS System 12

5 Conclusion and Future Works **15**

1 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 application: audio streaming, spatial audio rendering [1], live networked audio production [4], 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 (popular use).

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 [6, 17, 15] because of their unique performances for specific applications such as low latency [14]. Implementing DSP kernel algorithms on FPGAs has been addressed by the Syfala Compiler [13], 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 [7, 6], 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 audio cards.

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 has been implemented in the open-source distribution of Syfala[1] and has been used in the context of a Wave Field Synthesis (WFS) system (see Section 4.3). The solution is easy to deploy, as it relies only on JACK, Syfala, and the Rust compiler, all of which are open-source.

The protocol comprises an audio server running on a "generic" computer, interfaced with JACK, which sends UDP packets to an audio client on an FPGA connected to the network. It addresses the synchronicity problem by using a very basic resampling and is adapted to network jitter occurring on a LAN. Section 2 provides the precise problem formulation and the associated state of the art. The protocol is precisely described in Section 3. Performance metrics, including latency and clock drift correction, are presented in Section 4.

2 Problem formulation and state of the art

The development of audio networking solution has been driven by an increasing demand, particularly in professional audio and broadcasting environments. CobraNet, in the nineteens, is often credited as one of the pioneering technologies for audio-over-Ethernet [10]. 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 is the AES67 standard, published 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.

Unfortunately, in the context of our project, these standards proved to be quite complex to set up, due to the difficulty of finding complete open source implementations, even today.

Besides these developments, important work has been conducted towards simpler implementations using basic TCP/UDP protocols. The SoundWire project, initiated by the Center for

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

Computer Research in Music and Acoustics (CCRMA, Stanford University) [5], employs TCP to minimize packet loss or UDP in other audio diffusion scenarios. The JACK Audio Connection Kit has inspired many developments [11], such as JackTrip[2][8] for network music performance over the Internet, NetJack[3] [3], or the development around Zita-njbridge.[4] These tools have been developed by the open-source community and are usually not easy to modify. We encountered stability problems with NetJack and JackTrip, as well as performance issues with the Zita-njbridge package. For these reasons, we decided to use a direct interface to JACK, but our work can certainly be easily adapted to these tools.

First contributions to audio on FPGA date back to the 2000s and have continuously occurred since then. The general methodology for these works is based on manual hardware design [17, 9]. Few publications exhibit a real compilation process for audio DSP [16]. Vannoy et al. proposed an open-source IP-based design using MathWorks' *HDL Coder* [15]. FPGAs are also used in the audio industry for synthesizers or effects, but the FPGA designs are never open source.

The Syfala compiler was the first 'real' compiler for a high-level audio DSP language, namely the FAUST language,[5] to generate FPGA design bitstreams [14, 13]. It notably allowed us to reach the smallest analog-to-analog latency ever reported. With the addition of embedded Linux capabilities [6], it can drive a large number of audio codecs, potentially exceeding 1000, from a single FPGA board. However, to fully take advantage of this capability, one must stream all these audio signals directly to the FPGA. Given that the USB connection between the host and the FPGA is typically reserved for programming and debugging purposes, the Ethernet port present on all FPGA boards appeared to be the natural choice for this purpose.

The system view of an FPGA design resulting from Syfala is depicted in Fig. 1. The entire DSP application operates on the FPGA System-on-Chip (SoC). 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.[6] The embedded Linux distribution allows us to control the DSP IP through Ethernet using OSC (Open Sound Control) or an HTTP-based protocol, typically from the host computer or over the network. Ethernet also facilitates the direct streaming of audio signals on the SoC.

The problem we aim to address here is the integration of the Syfala toolchain with Ethernet transmission. 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.

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

3 Technical choices and implementation

Our system relies on the JACK API, enabling the transmission of audio streams using a protocol that encapsulates audio data in UDP packets. The intended audio transmission is bidirectional, supporting multiple clients. This section explains how the proposed architecture, as illustrated

[2] `https://jacktrip.github.io/jacktrip/`

[3] `https://netjack.sourceforge.net//`

[4] `https://kokkinizita.linuxaudio.org/linuxaudio/`

[5] `https://faust.grame.fr/`

[6] In the hardware design context, IP stands for *Intellectual Property* and designates a dedicated hardware component.

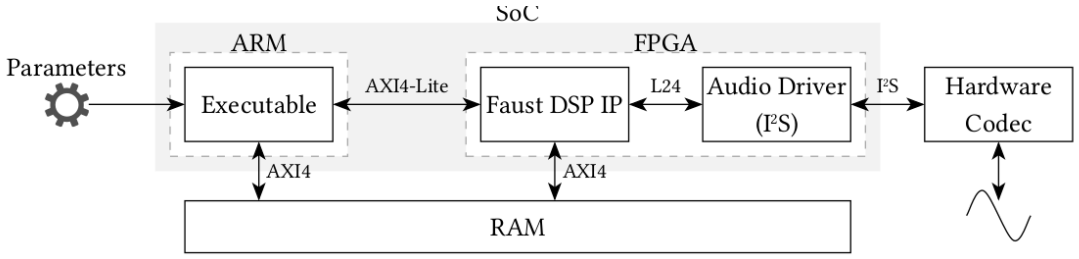


Figure 1: Original Architecture of the audio program compiled by Syfala [13]

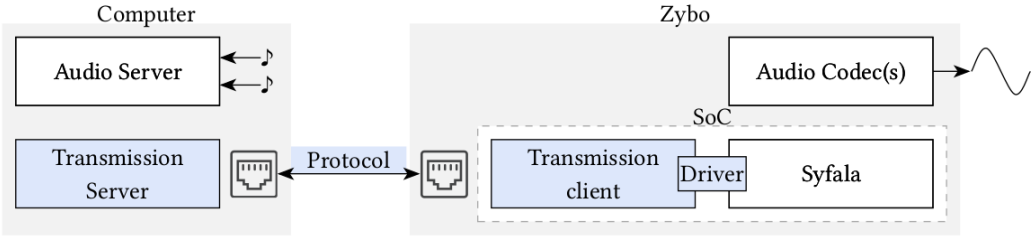


Figure 2: Targeted architecture coupling the Syfala toolchain and Ethernet transmission.

in Fig. 2, has been implemented with two transmission streams: one between the audio server on the computer and the ARM chip on a Digilent Zybo Z7 development board, and the other between the ARM chip and the Faust DSP kernel on the FPGA.

3.1 Custom Transmission Protocol

An overview of our custom transmission protocol is sketched in Fig. 3. The host (i.e., server) computer opens a TCP server on port 6910 and listens for incoming connections. The ethernet-client (on the FPGA processing system) connects to the TCP server, and transmits information on the DSP program, such as its number of input and output channels. The server responds by sending to the client other configuration details, 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. If the UDP or TCP connection fails, or if it is lost, the client starts a loop of asynchronous re-connection attempts. 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 lost packets. The server and client have been programmed in Rust to take advantage of Rust networking libraries.

3.2 Server-side implementation

The server-side implementation is shown on Fig. 4. 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 a separate thread, dedicated to audio streaming over network. In the case the client also sends audio back to the server, a Ring Buffer, as well as a Buffer Manager (for

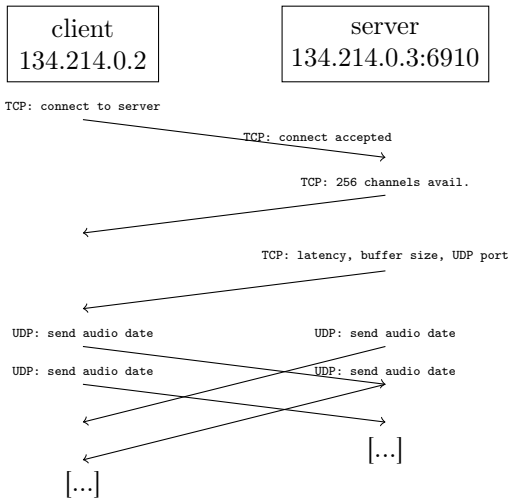


Figure 3: Custom client/serveur protocol for audio transmission

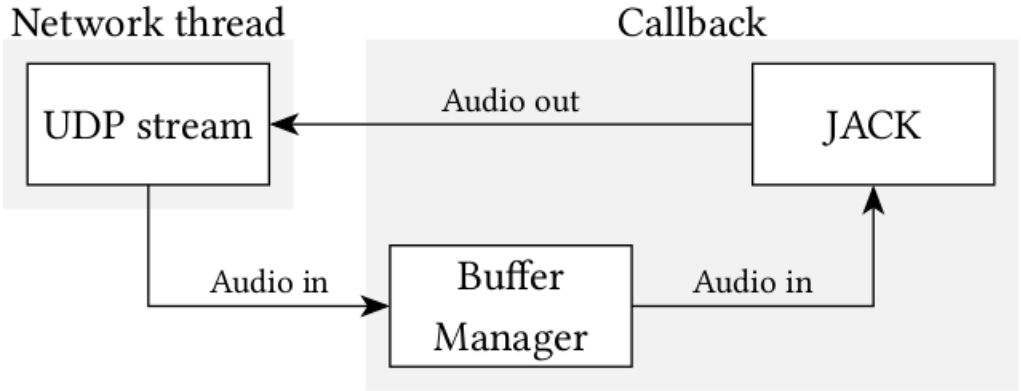


Figure 4: Server implementation using Jack.

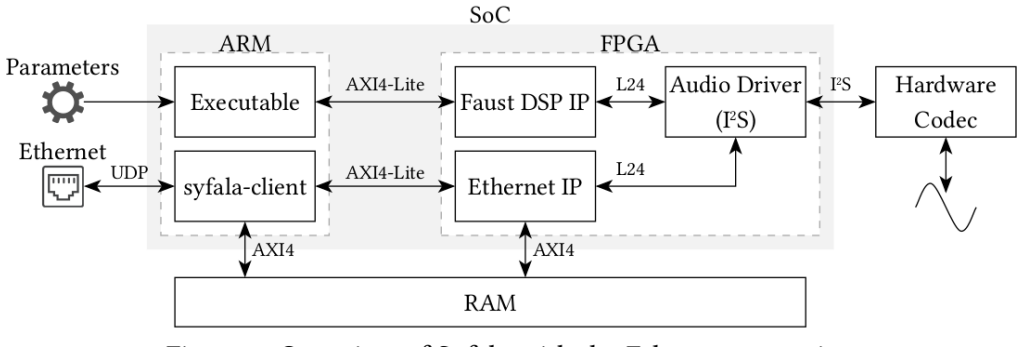


Figure 5: Client-side architecture, composed of two major components: the syfala-client and the Ethernet IP.

synchronization), are used. This process is further explained in the Section 3.3.3.

3.3 Client-Side Implementation

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 the Ethernet IP running on the Programmable Logic (PL).

3.3.1 Ethernet IP

The Syfala toolchain already supports multiple I/O audio codecs [13]. 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 (I²S) 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 3.3.3.
- An AXI4-Lite bus, a lighter subset protocol 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`. This function takes a few arguments, representing its input/output ports, as well as the AXI4/AXI4-Lite interfaces, and performs a sample-by-sample computation each time it is called by I²S. Once synthesized to VHDL code, This IP can be integrated to the final Syfala block-design and connected to other modules, such as the I²S or the AXI interconnect system, see Ehternet IP on Fig. 5.

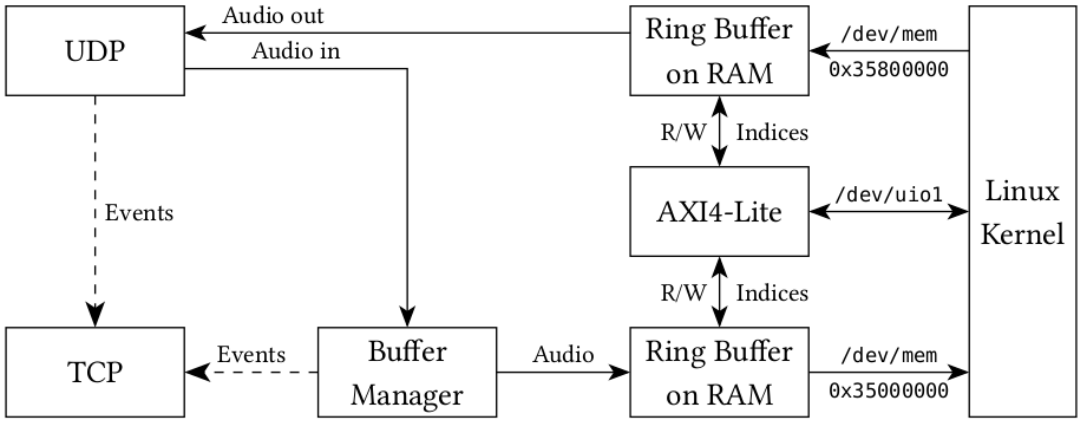


Figure 6: Syfala-client application running on the ARM processor and its interface with the DDR memory.

3.3.2 Syfala-client

The Syfala-client application is shown on Fig. 6. It serves as the interface with the remote server on the network, and also manages synchronization with 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-Lite protocol, which is interfaced as another Linux file device (`/dev/uio1`). By comparing the read index and the write index, the client can determine the fill level of the ring buffer and perform buffer management.

3.3.3 Buffer Manager and Resampling

We must consider that audio buffers are read on one side and written on the other and take into account network non-determinism, jitter, and clock drift. In this context, there is a possibility that one side of the system may read audio data that has not arrived yet, or write a sample that has not been read yet. In our system, clock drift correction is done by inserting or removing samples, a technique that can be seen a as a vary basic *resampling*. While a more robust solution would involve a clock synchronization mechanism based on the Precision Time Protocol (PTP), implementing such a system would significantly increase complexity.

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 is valuable in scenarios where the transmission is used for recording or debugging.

At the beginning of the transmission, 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.

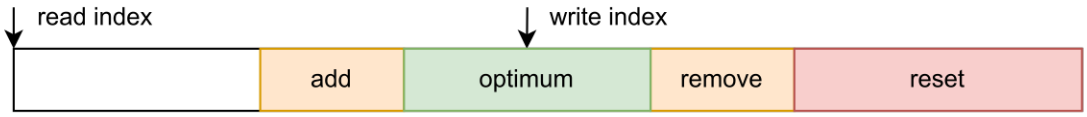


Figure 7: Read and Write indexes in the Ring buffer used to transmit audio data, and buffer zone used to perform resampling to correct clock drift.

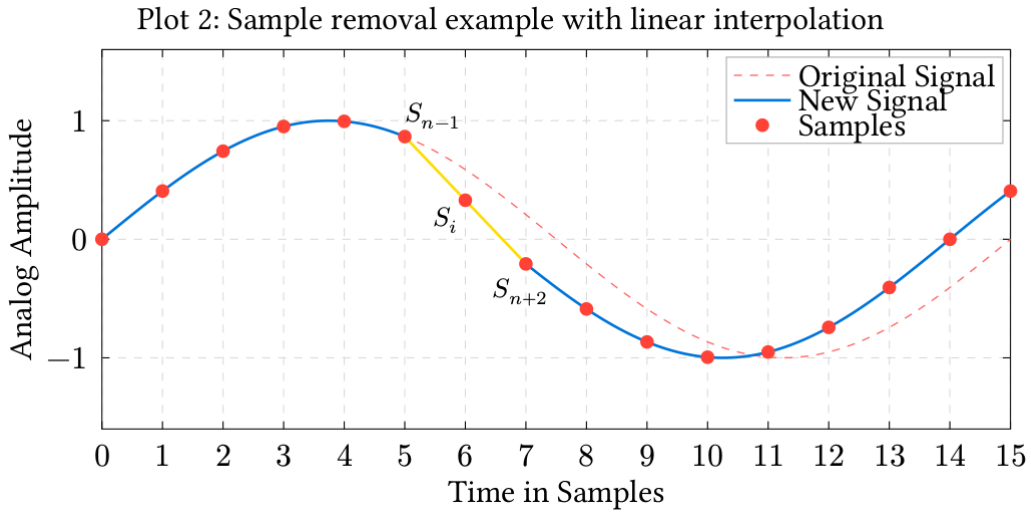


Figure 8: Linear interpolation used to correct clock drift.

The buffer manager evaluates the number of samples in the ring buffer as shown on Fig. 7. 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. 8. 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 real *resampling* stragegy.

3.3.4 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 [6]. 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 on initialization and also to exchange read-and-write pointers between FPGA and ARM at each sample.

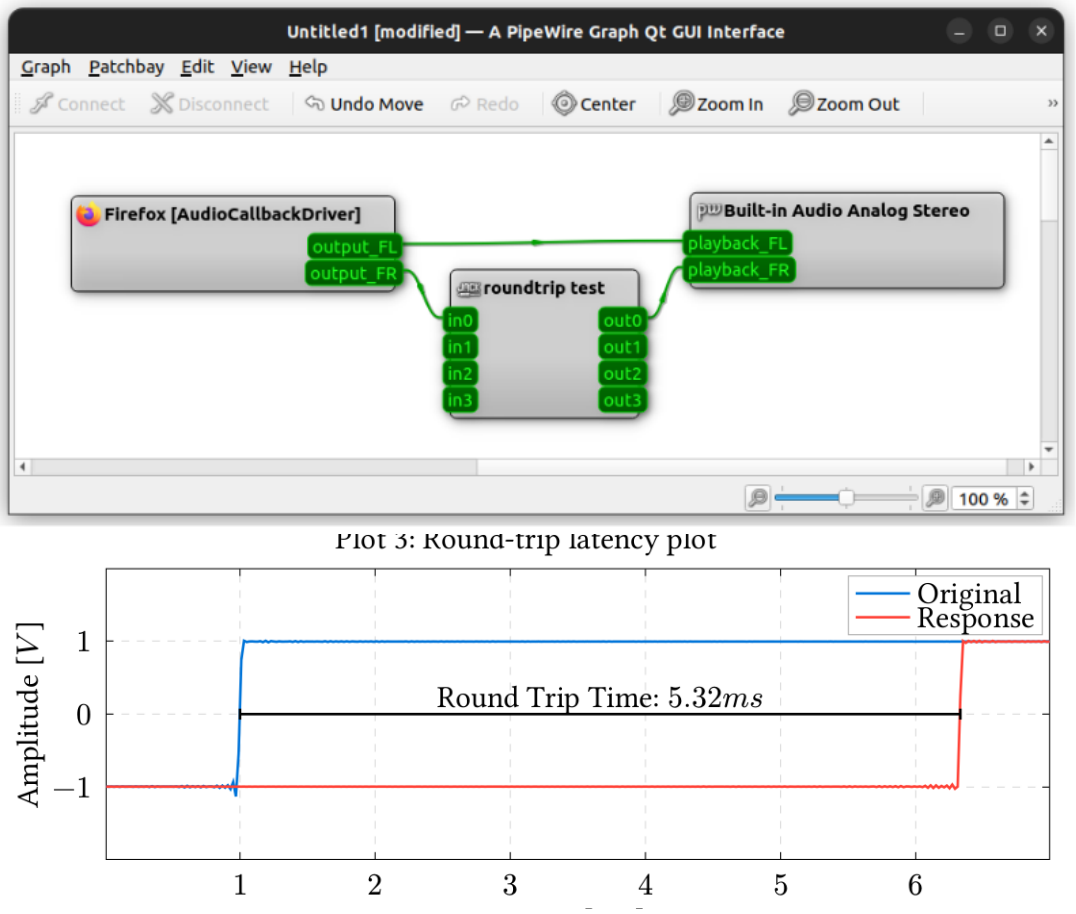


Figure 9: Round trip latency measure for 64 samples buffer.

4 Results and Performances

The complete transmission is now implemented in the Syfala toolchain and is available on GitHub[7]. It is currently used by the Emeraude Team in Lyon and by the Center for Computer Research in Music and Acoustics (CCRMA) at Stanford University (USA) as part of the PLASMA projects[8] 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 Manjaro laptop 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.

4.1 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. The top of Fig. 9 shows the JACK graph obtained with a Qt frontend. The bottom of the same figure illustrates the round-trip latency on a simple signal switching from -1 to 1. The round-trip latency is approximately 5.32 ms.

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 $4 \cdot T_B \simeq 5.32, \text{ms}$, this highlights again the very low latency added by the Syfala DSP IP (below 10 μs).

4.2 Clock Drift Correction and Scalability

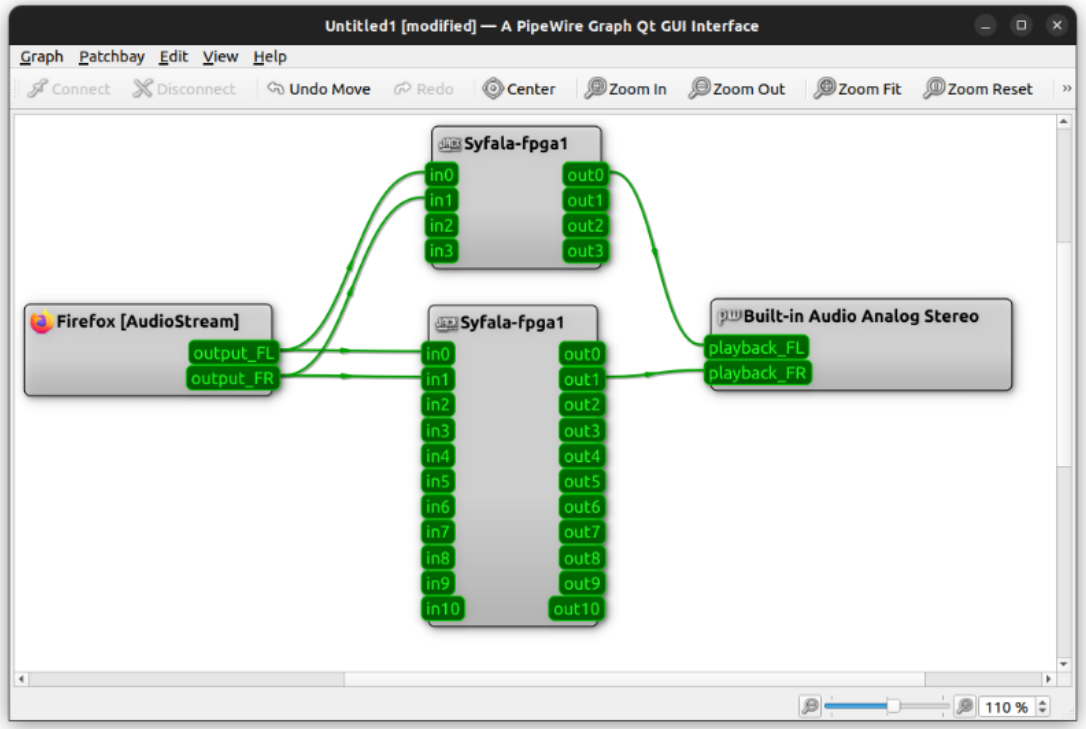
It is important to assess the scalability of our system. If a very large number of speakers is necessary, we may need several FPGA boards. We tested the system with two Zybo boards connected to the same server, totaling 24 channels in both directions (see top figure 10). Two Zybo boards running the same 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 crashes. During this test, we closely monitored the number of samples that were removed or added. The results in Fig. 10 show the expected clock drift of the Zybo. The clock drift was compensated, and 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 audio. When playing back a pure sine wave of a higher frequency (+400Hz), a soft clicking is sometimes audible, though it is not noticeable for a music-playing stream. The incorporation of the Rubato[9] resampling library resulted in slight improvements, but the marginal benefits were insufficient to justify the associated increase in CPU usage. This limitation would restrict the number of channels that can be transferred.

4.3 Full 32 Channel WFS System

The system was demonstrated at a public meeting, Tech Fest in Grenoble[10]. The application used the frugal 32 loudspeakers WFS system presented in [12] for rendering a music band performance involving 4 instruments. Each instrument could be moved via a web interface GUI. The system proved to be highly reliable, running for two complete days without interruption. Fig 11 shows, using Catia, the connection between the Ardour digital audio Workstation and the FPGA using 4 channels corresponding to the 4 audio sources of each instrument. 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. The control of the position of the instrument was send to the DSP IP by OSC, using a GUI not shown here. Fig. 12 shows the loudspeaker bar used [12]) and its connection with the FPGA Zybo board (Zybo Z7-20). The system kept on running during the two days of the meeting without problem.

[7] `https://github.com/inria-emeraude/syfala`
[8] `https://team.inria.fr/emeraude/plasma/`
[9] `https://github.com/HEnquist/rubato`
[10] `https://www.tech-fest.fr/`



Plot 4: Clock drift relative to the server clock

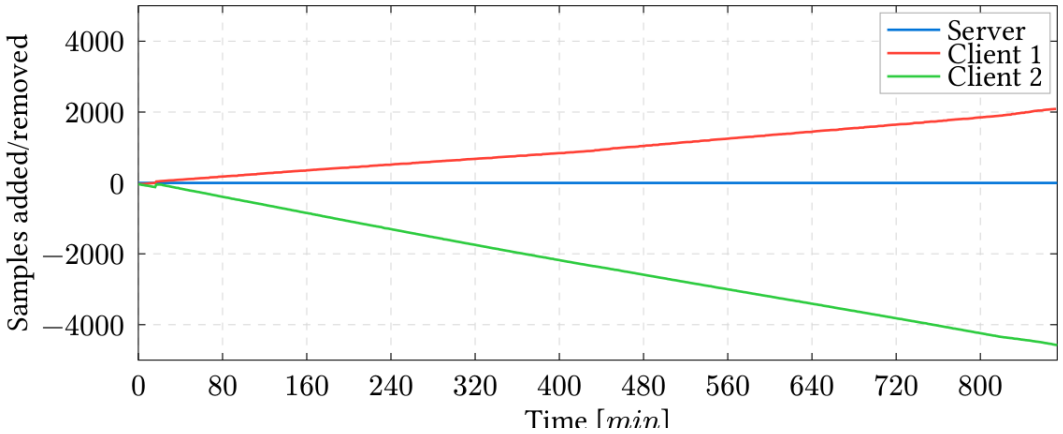


Figure 10: The JACK graph using 24 channel used for measuring clock drift between two FPGA boards.

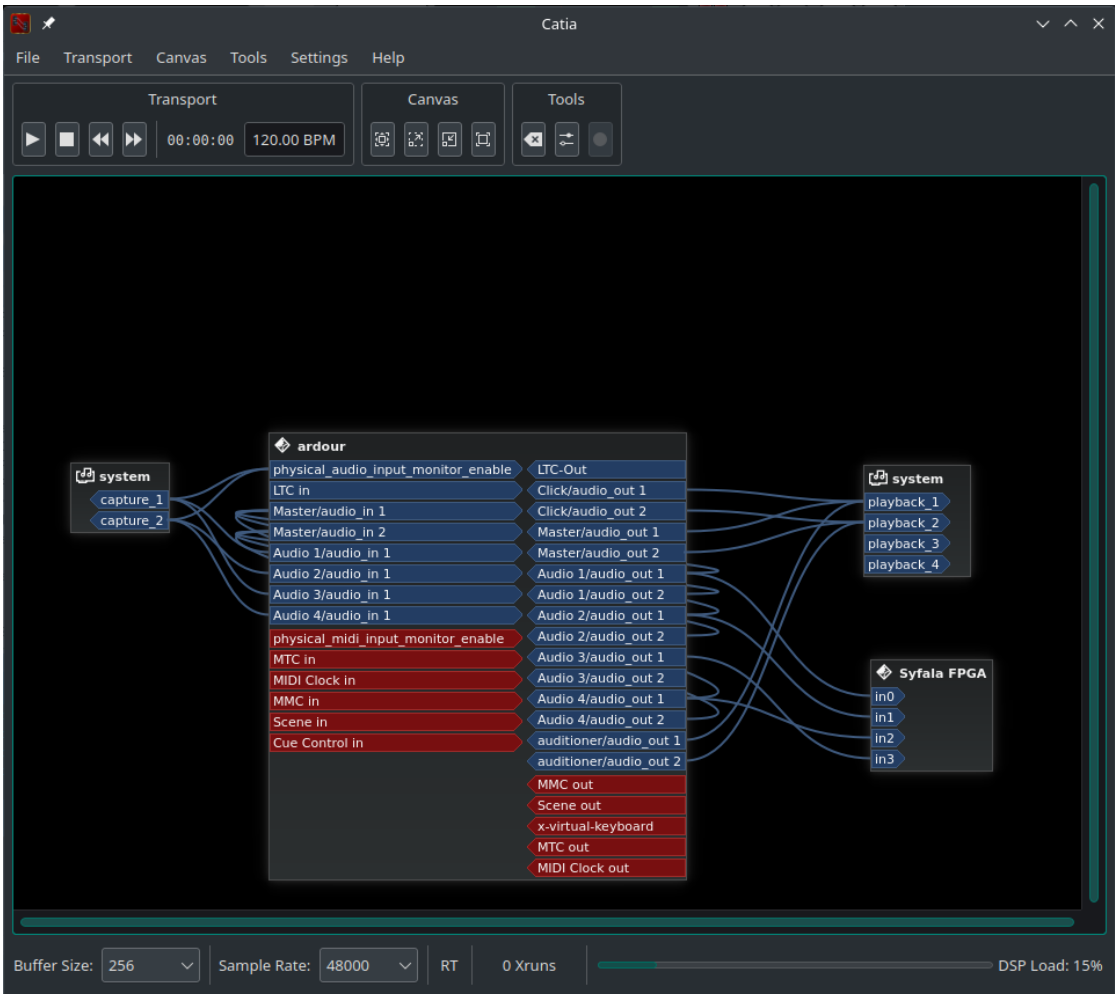


Figure 11: A screenshot of the connection between the Ardour digital audio workstation and the FPGA using 4 channels, one for each instrument's audio source.

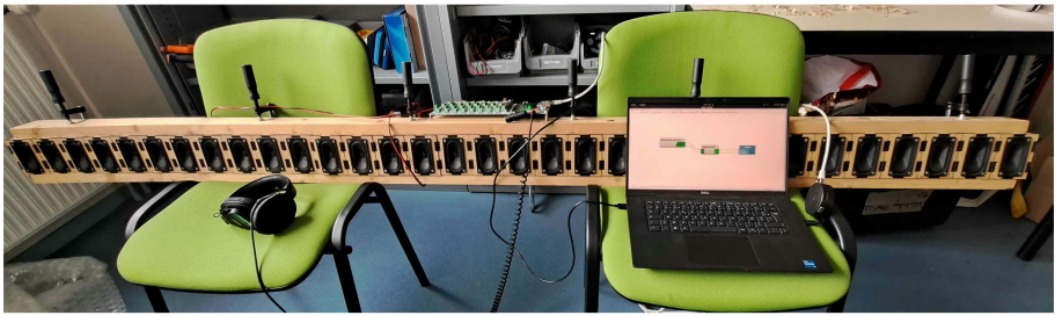


Figure 12: The WFS diffusion system ([12]) connected to the FPGA Zybo board

5 Conclusion and Future Works

We presented the first prototype of an Ethernet real-time audio transmission protocol to 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.

This prototype paves the way to many application in the domain of spatial audio, in particular. Would it by using ambisonic or WFS, the "frugal" spatial audio diffusion could be easily deployed in many places, 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 is also targeted for real-time noise canceling using the FXLMS algorithm such as the one presented in [2]. The proposed prototype is open source and fully accessible on GitHub[11].

References

[1] J. Ahrens, R. Rabenstein, and S. a. Spors. The theory of wave field synthesis revisited. In *Audio Engineering Society Convention 124*, May 2008.

[2] 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, Sept. 2023. European Acoustics Association.

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

[4] C. Chafe. I am streaming in a room. *Frontiers in Digital Humanities*, 5, 2018.

[5] 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.

[6] 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)*, pages 110–116, 2023.

[7] 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.* Strathclyde Academic Media, Glasgow, GBR, 2014.

[8] 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, Sept. 2010.

[9] 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, July 2021. IEEE.

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

[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] F. Lopez-Lezcano. From jack to udp packets to sound, and back. In *Linux Audio Conference 2012*, CCRMA, Stanford University, 04/2012 2012. CCRMA, Stanford University, CCRMA, Stanford University.

[12] 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.

[13] 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. Technical Report RR-9507, Univ Lyon, INSA Lyon, Inria, CITI, Grame, Emeraude, May 2023.

[14] 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 *Proc. Int. Conf. in Sound and Music Computing, SMC-22*, Saint-Étienne, France, June 2022.

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

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

[17] 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.

RESEARCH CENTRE
LYON

Bâtiment CEI-2, Campus La Doua
56 Bd Niels Bohr, 69100 Villeurbanne

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399