FPGA-based Low-Latency Audio Coprocessor for Networked Music Performance

FPGA-based Low-Latency Audio Coprocessor for Networked Music Performance

Diego Bert, Nicola Domini, Riccardo Peloso, Leonardo Severi, Matteo Sacchetto, Andrea Bianco, Cristina Rottondi
Department of Electronics and Telecommunications, Politecnico di Torino, Turin, Italy
{diego.bert,nicola.domini}@studenti.polito.it
{riccardo.peloso,leonardo.severi,matteo.sacchetto,andrea.bianco,cristina.rottondi}@polito.it

***Abstract*—Networked Music Performance (NMP) applications are acknowledged to be a particularly challenging field due to their stringent latency requirements and their demand for high audio quality. Most solutions developed in the last decades tried to overcome these obstacles by leveraging software approaches, that can introduce excessive time delays as a consequence of the general-purpose nature of the architectures on which they are implemented. Alternatively, a dedicated audio processor can be employed to minimize the mouth-to-ear latency.**

This paper presents the ongoing development of an hardware system that exploits an Application-Specific Instruction set Processor (ASIP) implemented on a Field-Programmable Gate Array (FPGA) to accelerate audio sample management. Specifically, a Transport Triggered Architecture (TTA) is being investigated as a processor design that aligns well with the required application domains. Preliminary empirical results indicate that the proposed solution has the potential to achieve extremely low latency, compatible with NMP requirements. Further optimizations and enhancements are actively being pursued to address the yet open challenges posed by NMP applications.

***Index Terms*—Networked Music Performance, audio latency, audio processing, Field-Programmable Gate Array**

I. Introduction

A Networked Music Performance (NMP) is a real-time musical interaction that allows musicians to play together from remote locations. Its main purpose is to link different performers who are not physically in the same room, making them feel like they are playing close to each other. This is usually done by leveraging an ultra-low latency audio stream over a telecommunication network. However, in this scenario, one of the most problematic and crucial challenges is the necessity of maintaining stable real-time communication between users. As confirmed by a large number of studies, the mouth-to-ear delay recognized to be acceptable for NMP is approximately around 20-30 ms [1]. Since there are several bottlenecks in the communication, it is not straightforward to meet this kind of requirement.

To reduce latency, different software and hardware solutions have been proposed. However, since latency requirements are very severe, a purely software-based approach is not always the best option. An alternative solution, described also in [1], is the use of a system based on a Field-Programmable Gate Array (FPGA), where audio packets are completely handled by dedicated hardware. In this way, the sources of additional latency that exist in a general-purpose architecture can be bypassed or accelerated.

The solution considered in this paper is based on the preliminary work presented in [2] and [3]. It consists of a dedicated hardware system designed to ensure extremely low latency and is created by combining a software approach and a FPGA-based one. In particular, the design of an Application-Specific Instruction set Processor (ASIP) implemented by taking advantage of a Transport Triggered Architecture (TTA) is proposed. This architecture is particularly suitable for the NMP application domain, as it can be exploited to process audio samples with a reduced amount of added latency.

To understand the possible advantages of this kind of approach, some other solutions will be first examined in Sec. II. In Sec. III, an introduction to the TTA processor architecture and the toolset exploited for its design is provided. The proposed system is described in detail in Sec. IV, providing an overview of its components and the interfaces between them, with a particular focus on the FPGA-based audio coprocessor and its firmware. Then, in Sec. V, the results obtained so far from a theoretical and an empirical point of view are summarized, focusing in particular on the performance of the FPGA. In that section, also some key future research directions are pointed out. Finally, conclusions are presented in Sec. VI.

II. Related work

During the last two decades, the interest in NMP has considerably increased, as witnessed by a large number of solutions that have been proposed both in literature and by companies. State-of-the-art NMP solutions consist mainly of software frameworks, but also some hardware approaches have recently gained attention. For a thorough comparison of existing NMP solutions, the reader can refer to [1] and [4]. In the following, we provide a brief overview of the most widely adopted ones.

One of the first software solutions, developed in the early 2000s, is Jacktrip [5], an open-source software that runs on most general-purpose Operative Systems (OSs). To achieve a real-time musical interaction, it transmits uncompressed audio samples through the network leveraging the User Datagram Protocol (UDP) at transport layer.

Instead, SoundJack [6] is an NMP software application that employs a compressed audio format. This enables integration of Packet Loss Concealment (PLC) techniques directly incorporated in audio codecs, at the price of increasing the

mouth-to-ear latency as a consequence of the time overhead introduced by compression algorithms.

Another first-generation NMP framework is DIAMOUSES [7]. This is again an open-source software that streams uncompressed audio signals, which also incorporates live video streaming. However, this introduces further latency issues, since video acquisition and encoding delays are typically higher than those required by audio signals, thus leading to misalignment between the two streams. To overcome such issue, LOLA [8] resorts to uncompressed video streaming, at the price of increasing the required bit-rate to a few Gbps.

In 2020, video streaming features were incorporated in Jacktrip and Soundjack as well, as a consequence of the massive hype on NMP applications generated by the social distancing measures enforced during the breakout of the SARS-CoV-2 pandemic. To do so, the low-latency audio streaming features enabled by the novel WebRTC paradigm [9] were also exploited to enrich those applications with web-based interfaces.

An almost completely different approach is the one adopted by the Elk Audio company [10]. The main difference with respect to the solutions described so far is that Elk offers a complete setup for real-time remote performances, consisting of both a hardware and a software environment. In particular, Elk Audio designed a hardware audio interface called Elk Bridge, optimized for low latency audio applications and conceived to be used with a proprietary software called Elk LIVE. With this kind of solution, the time delay introduced by the sound acquisition and processing chain is further decreased.

A similar approach was pursued in [11], where they propose a software running on a single-board computer (BeagleBone Black), leveraging the Xenomai kernel extensions, to implement real-time Digital Signal Processing (DSP) functionalities. Its performances were benchmarked in [12], which shows that this is the only tested solution able to meet the stringent jitter and latency requirements.

Several approaches to exploit FPGAs for real-time audio processing have already been investigated in [13]–[15]. Though, to the best of our knowledge, apart from some preliminary attempts appeared in 2013 [16], [17], this study proposes for the first time an implementation of an ultralow-latency audio streaming coprocessor based on FPGA, specifically designed for NMP applications.

III. Background

As mentioned in Sec. I, the proposed system makes use of a custom Transport Triggered Architecture processor. The datapath of this architecture comprises several Functional Units (FUs), register files and interconnecting buses.

The instruction set philosophy for the TTA processor revolves around a single type of instruction, namely the “move" operation. In this system, more complex operations can be accomplished by orchestrating a sequence of “move" instructions between different FUs, which then actually process data. As a consequence, at compile time, each user program is mapped to a discrete set of operations, moving data from one FU to another via the interconnected buses. This compilation process ensures the efficient execution of higher-level tasks using the fundamental “move" operation as a building block. The length of each instruction is usually very long because it is divided into several move slots, each one dedicated to one of the available transport buses. For this reason, a TTA processor enables a natural parallelism in the execution of instructions.

A picture of a simple TTA architecture is shown in Fig. 1. Due to its characteristics, TTA processors are particularly suitable to implement ASIPs, enabling designers to create custom units to accelerate specific tasks. Given the focus of this paper on NMP, a TTA proves to be a valuable choice to reduce the latency of audio data handling. Specifically, tasks like audio playback, audio source mixing, and audio stream processing (e.g., effects, equalization) can be efficiently sped up using the TTA architecture. To achieve this acceleration, dedicated FUs can be designed for each of these functions. The TTA’s inherent parallelism allows these specialized FUs to work simultaneously, enabling efficient and concurrent execution of multiple NMP tasks. This approach harnesses the power of parallel processing to meet the stringent latency requirements and demanding audio processing needs of NMP applications.

To facilitate the creation of a custom TTA processor, TTA-based Co-design Environment (TCE), now known as OpenAsip [18], was adopted in this project. Developed by the Customized Parallel Computing group at Tampere University, this environment offers a comprehensive set of tools to aid users in co-designing the Register Transfer Level (RTL) architecture of a TTA processor and the high-level software running within it. Thanks to TCE, users can seamlessly design and interconnect transport buses, register files, and basic FUs, forming the foundation of the TTA processor. Furthermore, custom FUs described by means of a Hardware Description Language (HDL), like those discussed in this paper, can be integrated into the system. To enable high-level programming for the generated architecture, a custom LLVM/CLANG compiler is employed. Users can write their high-level code in C/C++ programming languages, which the compiler then translates into instructions compatible with the custom processor. This cohesive ecosystem ensures efficient co-design and implementation of custom TTA processors, empowering developers to effectively tailor their solutions for specific tasks.

IV. System description

The system proposed in this paper is an initial hybrid solution combining a software and a custom digital hardware approach. In particular, as shown in Fig. 2, it consists of three main components, each of them implemented on a different hardware support: an audio board, an audio processor and a Raspberry Pi.

The three main components of the system exploit different communication protocols to interface with each other (see Fig. 2):

- the Inter-IC Sound (I2S) protocol, between the audio board and audio processor.
- the Musical Instrument Digital Interface (MIDI) protocol, between the audio board and audio processor.

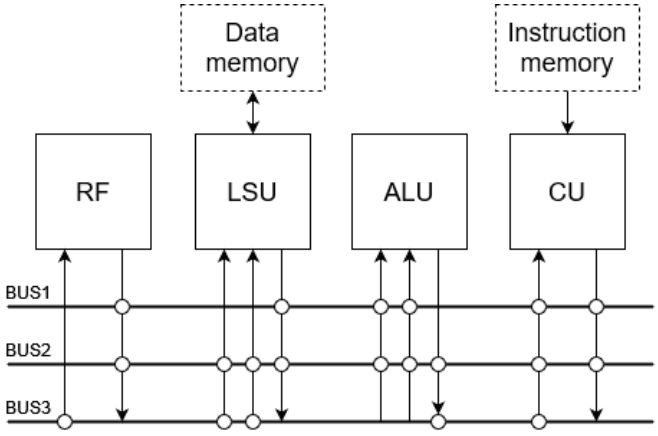

Fig. 1. Simple TTA architecture

- the Serial Peripheral Interface (SPI) protocol, between the audio processor and Raspberry Pi.

Internally, the audio processor and the audio board can handle up to 24-bit samples, even if at this moment the Raspberry Pi host machine works with 16-bit samples.

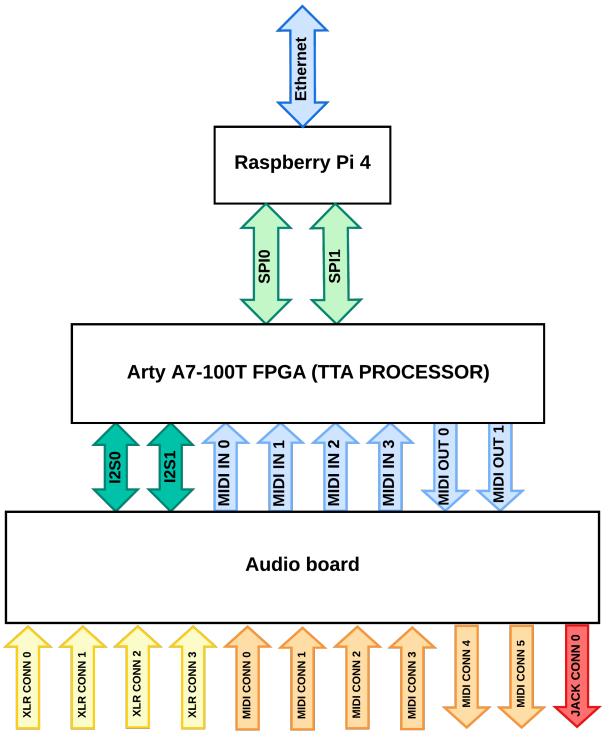

Fig. 2. System block diagram

A. Audio board

The custom audio board (Fig. 3) serves as a hardware circuit designed specifically to function as a sound card within the system. Its conception revolves around meeting stringent low-latency requirements and addressing issues commonly associated with common consumer-grade sound cards based on USB 2.0.

The Universal Serial Bus (USB) 2.0 is a widely adopted half-duplex serial interface known for its versatility and ease of use. It is commonly utilized for connecting various peripherals and devices to computers and other host systems. USB is designed to be flexible and support a wide range of functionalities, making it suitable for tasks such as data transfer, power supply and device control. However, USB is not inherently optimized for low-latency applications, such as real-time audio processing in the context of NMP. The USB protocol was originally designed to cater to various devices with different data transfer requirements, including keyboards, mice, printers, and storage devices. As a result, the USB protocol includes a complex control software stack that introduces some inherent overhead. This software stack, while necessary for managing the various devices and their functionalities, can introduce additional processing delays, which may impact real-time audio processing in NMP scenarios. The overhead introduced by the USB software stack can result in unpredictable variations in data transfer times and latency, making it challenging to achieve the precise and consistent timing required for real-time audio processing.

To ensure optimal performance, the developed audio board is engineered to efficiently acquire and play sound samples using multiple audio interfaces. It seamlessly exchanges data with the processor via the I2S protocol, employing minimal buffering and filtering. This reduction in buffering and filtering helps minimize added delays and optimize latency performance. The board is equipped with readily available commercial audio Analog-to-Digital Converters (ADCs), specifically two PCM1803 [19], and a Digital-to-Analog Converters (DAC), the PCM1771 [19], which can be directly employed as a headphone amplifier. MIDI [21] interfaces are derived by converting UART messages through commonly used optically-isolated circuits.

The custom audio board is designed to accept input DC voltages ranging from 3V to 18V. It efficiently derives the necessary power supplies for both the digital and analog sections through a combination of switching and low-noise linear regulators. Additionally, the board has the capability to generate a 48V Phantom power supply to facilitate the operation of condenser microphones directly on board.

Furthermore, the audio board assumes the critical role of directly interfacing with users for control purposes. It provides user controls, such as volume adjustment for each input audio channel, enabling smooth and intuitive interaction for performers and participants in an NMP session.

B. Audio processor

The second and most crucial component of the developed system is the audio processor, implemented as a custom TTA ASIP. Its goal is to create a communication channel between the audio board and the Raspberry Pi device, introducing the lowest possible amount of latency, as well as providing custom audio processing and manipulation, thus removing the computational load from the Raspberry Pi and accelerating these tasks. To host the processor, an FPGA-based board was chosen because of its flexibility in implementing digital

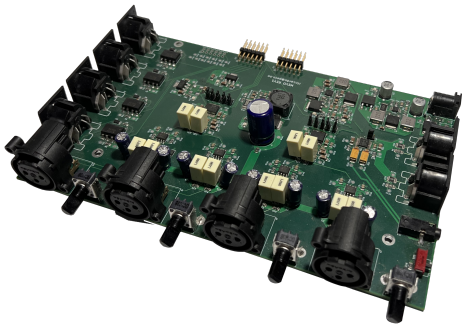

Fig. 3. Custom audio board

circuits and its possibility to be completely reconfigurable. The choice for the development board fell on the development board Arty A7-100T produced by Digilent, featuring the Artix-7 XC7A100TCSG324-1 FPGA chip from Xilinx [22]. This FPGA model boasts an abundance of digital resources that can be harnessed for future advancements in the audio processor's capabilities.

Fig. 4 depicts a block diagram of the top module of the processor. Several components are shown:

- *TTA core*, the actual processor, hosting the different FUs.
- *Instruction memory*, containing the instructions obtained by compiling the user code.
- *Data memory*, used to store data and global variables.
- *I2S clock generator*, needed since the developed architecture acts as the master device for the I2S protocol. It generates the required clocks, namely the bit clock (bclk) and the left-right clock (lrclk).
- *Phase-locked loop (PLL)*, generating the master clock running at 11 MHz.

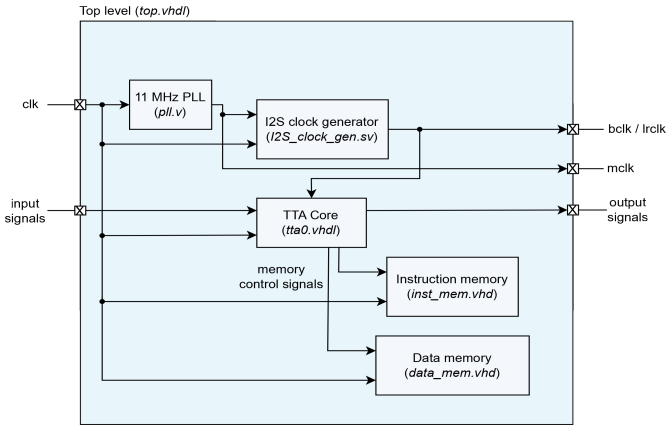

Fig. 4. Audio processor top-level block diagram

It is possible to observe in Fig. 5 that the developed TTA processor is composed of several FUs connected together by means of four 32-bit transport buses. Some of these units are common to most architectures, such as Arithmetic Logic Unit (ALU), Global Control Unit (GCU), Load-Store Unit (LSU), two register files ("bool" and "REG_FILE_0" in the figure) and a timer ("TIMER_0" in the figure). Then, some custom units were developed in order to deal with the Light Emitting Diodes (LEDs) and switches of the FPGA board but, most importantly, to create peripherals that interface the system components and provide some processing on the audio data:

- *I2S_LJ_master_TX_x*: I2S transmitter with Left-Justified format and 24-bit audio samples on 32-bit frames.
- *I2S_LJ_master_RX_x*: I2S receiver. Its behavior is symmetrical to that of the I2S transmitter.
- *UART_TX_x*: UART transmitter, working at 31250 Hz rate. It sends 8-bit MIDI messages.
- *UART_RX_x*: UART receiver, working at 31250 Hz rate. It is used to receive 8-bit MIDI messages.
- *SPI_slave_x*: an SPI transmitter/receiver, working with 8-bits frames.
- *MIXER_0*: performs the mixing operation on multiple audio sources. It features gain and left-right panning controls for each source.
- *REVERB_0*: applies a reverb effect on an audio source. It features a dry/wet control.

The last two units of the list, as well as other possible future ones, can be exploited to accelerate time-consuming audio processing tasks typically required by musicians. Moreover, the introduction of a reverb unit can have a positive impact on the way users interact, leading to more synchronized performances [23].

All the functional units are linked together with a fully connected structure, i.e., each one of them has access to all the available buses. In this way, the compiler is free to choose how to route the data and how to organize the move operations, depending on the required tasks.

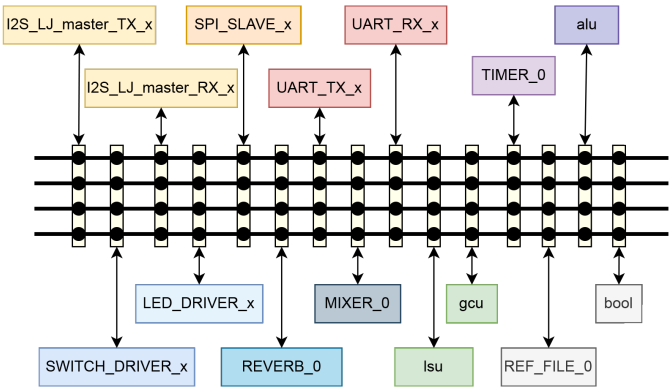

Fig. 5. TTA core: FUs and interconnections

For what concerns the uploaded firmware routine, the FPGA processor's task is to collect data coming from the audio board peripherals (I2S and MIDI) and route them to the SPI peripheral connected to Raspberry Pi, and vice versa.

SPI was selected as the interconnection protocol between the FPGA and the Raspberry Pi due to its widespread availability as a standard interface for prototyping purposes. This synchronous and dependable interface is well-suited for transmitting versatile data payloads, encompassing both audio data and command signals. For this task, I2S has not been employed as it is primarily designed for the transmission of stereo audio streams and incorporates an embedded clock signal. One advantage of the custom-designed TTA I2S interface

is its internal buffering, which eliminates the need for a master clock. Sample re-synchronization occurs only when audio data is sent to the Digital-to-Analog Converter (DAC), ensuring precise timing and synchronization. DSP tasks are executed asynchronously at the FPGA's system clock frequency. This design approach is geared towards minimizing latency, thereby ensuring that audio processing tasks are carried out with the lowest possible latency.

Support for two different operating modes has been considered, depending on whether the mixing is done at the Raspberry Pi or FPGA level. The latter option accelerates the mixing task and so speeds up the system, but will likely experience limitations for what concerns the maximum number of sources it can handle, due to the bandwidth of the SPI link (more details in Sec. V-D).

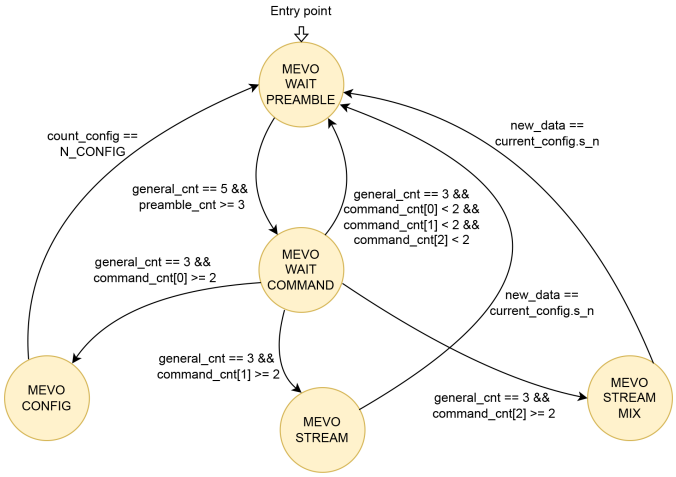

Fig. 6. Flow diagram of the firmware routine

During operation, the FPGA processor waits for a command coming from the Raspberry Pi, that is acting as an SPI master device. Three possible commands can be issued:

a) Command 1: is used to change the environment configuration variables, such as the number of audio samples bitwidth, the number of enabled input channels on the audio board, the number of sources coming from the Raspberry and others.

b) Command 2: starts the streaming of a packet of samples and MIDI messages from the audio board to the Raspberry Pi and vice versa. The mixing operation is performed by the Raspberry Pi.

c) Command 3: starts the streaming of a packet of audio samples and MIDI messages from the audio board to the Raspberry Pi and vice versa. In this case, the mixing operation is done at the FPGA level.

Furthermore, an analysis has been carried out regarding the effects of errors that can potentially occur in the SPI channel. While the impact of sporadic errors in random audio samples, resulting in occasional audio glitches, can be deemed tolerable, keeping the error rate as low as possible is crucial when issuing commands, in particular command 1. In fact, if one or more of the configuration variables are not correctly received by the FPGA side, the whole communication could fail. To address this problem, a custom communication protocol has been developed. First of all, each command is preceded by a preamble sequence informing that the Raspberry Pi is asking to start the communication. Then, the transmission continues with an ACK/NACK mechanism. Moreover, redundancy in the preamble and commands is intentionally introduced, guaranteeing that the correct information is exchanged.

In the final implementation, SPI communication packets will incorporate Error-Correcting Codes (ECCs) such as Cyclic redundancy checks (CRCs) to enhance the reliability of the communication channel. The ECC scheme will be supported by firmware or hardware during the packet encoding and decoding processes, thereby providing an additional layer of assurance for data transmission integrity.

Fig. 6 represents a flow diagram of the firmware routine, devised as a Finite State Machine (FSM) consisting of the following states:

- **MEVO_WAIT_PREAMBLE**: during this state, the processor waits for the preamble sequence coming from the SPI side.
- **MEVO_WAIT_COMMAND**: when the preamble sequence has been identified, the processor enters this state expecting a command to be issued.
- **MEVO_CONFIG**: in this state, command 1 is executed.
- **MEVO_STREAM**: in this state, command 2 is executed.
- **MEVO_STREAM_MIX**: in this state, command 3 is executed.

Currently, the processor does not perform many operations and is clearly underused. Anyway, its real potential is still to be exploited in enhanced versions of our architecure, that are forseen as future work. Transferring tasks from the Raspberry Pi to the FPGA offers several advantages, particularly in terms of speed. Consider, for instance, a reverb effect. In software, processed samples must continually shuttle back and forth between the processor and memory. Conversely, hardware implementation allows for efficient utilization of parallelism and pipelines, resulting in a notable reduction in computational overhead. It is true that developing hardware solutions entails higher upfront costs, but in a latency-critical context like NMP, the investment is worth the effort.

C. Raspberry Pi

In the proposed system, the Raspberry Pi 4 model B [24] plays a crucial role in managing the transmission and reception of audio packets over the network through its full gigabit Ethernet port. Given the stringent latency requirements of NMP, the choice of an appropriate device was crucial, and the Raspberry Pi 4 specifications proved to be a fitting option. It boasts a 1.5 GHz Quad-core 64-bit ARM Cortex-A72 processor and 8 GB of RAM, providing ample processing power and memory capacity for real-time audio processing tasks.

Moreover, the Raspberry Pi 4 offers 28 General Purpose Input/Output (GPIO) pins, providing various interface options and including support for multiple concurrent SPI communication lines. These capabilities make it a versatile platform for handling audio data efficiently. To ensure a robust and

flexible system, the Linux operating system was deployed on the Raspberry Pi, benefiting from its stability and adaptability within the Raspberry Pi framework.

From a networking perspective, two crucial design choices were made. Firstly, UDP was chosen as transport protocol for communication. UDP's lack of handshake techniques for connection setup enables faster data transmission, making it well-suited for time-critical applications like NMP. However, it is important to acknowledge that UDP communications are not reliable and may require PLC methods to address missing data packets. Secondly, a peer-to-peer architecture was implemented among the users of the NMP environment. This choice further reduces perceived latency compared to a client-server solution, facilitating direct and low-latency communication between participants.

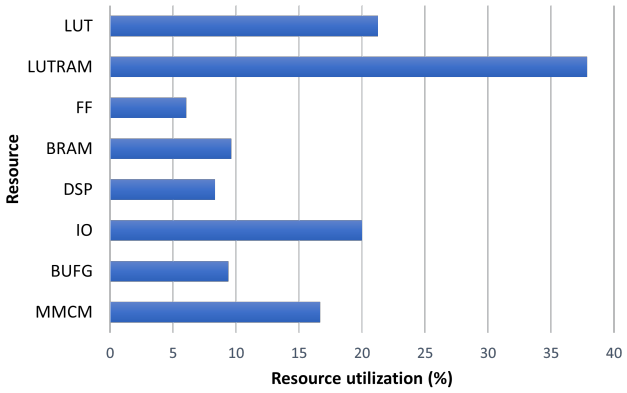

Fig. 7. FPGA resource utilization

V. Results

The following sub-sections present the results obtained so far for the proposed system. It is worth mentioning that, since the architecture design is still ongoing, most of the experimental evaluations cannot yet be assessed, especially the ones related to the Raspberry Pi component. For the same reason, a thorough evaluation of the impact of network delay and of the overall system performance will be possible only once the design phase is completed.

A. FPGA utilization

The proposed TTA processor was implemented on FPGA leveraging the Vivado IDE by Xilinx to evaluate the efficiency of the design and the resource consumption. The reported utilization results, shown in Fig. 7, indicate how the ASIP makes use of the FPGA resources. Specifically, approximately 21% (13479 out of 63400) of the available logic elements, i.e. Lookup Tables (LUTs), were employed, while only 8% (20 out of 240) of the DSP cells were used. These outcomes emphasize that many signal processing operations on sound samples can still be handled by means of specific units, performing tasks which are recognized to be particularly time consuming in a general-purpose environment.

Regarding memory blocks, it can be noticed that 10% (13 out of 135) of the available Block RAMs (BRAMs) and 38% (7193 out of 19000) of the Lookup Table RAMs (LUTRAMs) were used, mainly to store instruction and data memories of the TTA processor and to address the need for long delay lines in the reverb unit. This results indicates that the design made efficient use of the FPGA, leaving room for potential inclusion of additional functionalities which require to store data. Thanks to this available programmable logic, several features can be taken into account for future developments as custom FUs, such as new audio effects and hardware PLC techniques to cope with missing data packets.

B. FPGA power consumption

Power consumption is a critical factor, especially in portable or energy-constrained systems. To assess the power efficiency of the proposed FPGA-based processor, the Vivado static power consumption estimator has been used. The power consumption of the developed TTA architecture was found to be significantly lower compared to traditional software-based approaches, with a total estimated power of approximately 350 mW. In particular, in Fig. 8, the different power contributions are reported. The main ones are those related to the PLL circuit (indicated as "MMCM" in the figure) and to the static power consumption of the ASIP (indicated as "PL static" in the figure).

The power consumption results obtained so far are acceptable estimations provided by Vivado. However, to understand the real power dissipation of the processor, future empirical evaluations will be considered by exploiting FPGA power monitoring capabilities.

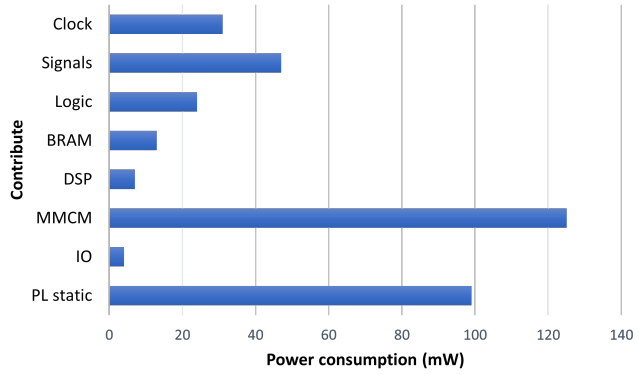

Fig. 8. FPGA power consumption contributions

C. Local system latency

The primary performance metric for NMP applications is the total system latency, which directly impacts the real-time musical interaction between remote performers. To evaluate the effectiveness of the proposed solution, latency measurements were carried out with a local audio loopback test on the system composed of the audio board and the FPGA-based processor (the Raspberry Pi was not considered in this initial analysis). Tests were conducted without involving the audio mixing unit, since this process was assumed to be

performed by the Raspberry Pi. Moreover, the reverb effect was excluded from the analysis, because it could alter the time delay evaluation. The experimental setup consisted of a software tool capable of generating a chirp sound and collecting it after the audio loopback to measure the latency introduced by the system. Since the sampling time of the exploited measurement tool is equal to 192 kHz, empirical results are affected by an uncertainty of 0.005 ms.

From a theoretical point of view, the overall latency considered can be summarized as:

$$T_{lat} = T_{ADC} + T_{I2S} + T_{proc} + T_{I2S} + T_{DAC}$$

where T_{ADC} and T_{DAC} are the time delays associated with audio conversions on the audio board, T_{I2S} is the data transfer delay (considered two times since data are transferred back and forth) and T_{proc} refers to the latency introduced by the TTA processor. By performing this computation, it was possible to estimate a theoretical value for the loopback test of 0.89 ms, which was then confirmed by empirical results, showing a 0.859 ms delay.

D. SPI bandwidth and protocol overhead

The required bandwidth to link the FPGA to the Raspberry Pi through the SPI interface is primarily determined by the number of audio channels being utilized. With a fixed audio sample rate of $f_s = 44.1\,\mathrm{kHz}$ and $w_s = 16$ bits per sample, each channel demands the following bandwidth

$$BW_{chann} = f_s \cdot w_s = 705.6\,\mathrm{kbps}$$

However, it is essential to consider the overhead introduced by the communication protocol introduced in Sec. IV-B, as it affects the effective data rate. For instance, when dealing with command 2 (Raspberry Pi mixing mode), the protocol introduces redundancy that needs to be accounted for in the bandwidth calculation. To quantify the overhead, a formula can be used, taking into account the samples bit width (w_s), the number of channels sent by the FPGA through the communication link (n_{CHF}), and the number of samples inside an audio data packet (s_n). For example, with $w_s = 16$ bits, $n_{CHF} = 4$, and $s_n = 112$, the total overhead introduced in a packet of samples can be computed as

$$OH = \frac{8}{(\frac{w_s}{8} \cdot n_{CHF} \cdot s_n) + 8} \approx 0.88\%$$

where the number 8 at the numerator represents the number of bytes required by the preamble+command sequence, while the denominator term represents the total number of bytes transferred during one occurrence of command 2.

Another interesting figure is the number of audio sources that can be streamed through the SPI link, especially when the mixing operation is performed at the FPGA level. To be sure to work in "safe" conditions, BW_{chann} can be increased to a higher value, for instance $BW'_{chann} = 750\,\mathrm{kbps}$. According to the datasheet of the chip BCM2711, which implements the SPI communication on the Raspberry Pi device [25], the maximum speed the link can reach is $BW_{RPI} = 125\,\mathrm{MHz}$. Thus, supposing that the electrical connection between the Raspberry Pi and the FPGA board permits to reach this speed, the theoretical maximum amount of sources that can be streamed through the SPI link is

$$N_{SRC_{MAX}} = \frac{BW_{RPI}}{BW'_{chann}} = \frac{125\,\mathrm{MHz}}{750\,\mathrm{kbps}} \approx 166$$

Moreover, Raspberry Pi features two SPI peripherals, that can potentially be used in parallel to increase the throughput and/or to spread the bandwidth requirement on both of them.

VI. Conclusions

This paper presents the ongoing design and implementation of a hardware audio streaming system designed for Networked Music Performance applications. This system employs an Application-Specific Instruction set Processor (ASIP) implemented on a Field-Programmable Gate Array (FPGA). The focus of this research is the exploration and design of a Transport Triggered Architecture (TTA) coprocessor, as its characteristics align well with the specific requirements of NMP applications.

The results obtained so far demonstrated that the proposed ASIP-based solution achieved remarkably low latencies in local loopback tests, meeting optimal requirements for NMP communication. Additionally, lossless audio processing leads to high quality input and output sound streams, ensuring a seamless musical interaction among remote performers. Even if the results achieved are mostly focusing on the FPGA performances up to now, the overall system can be deemed as a novel and promising approach to NMP.

The combination of software and FPGA-based hardware solutions offers an alternative approach to tackle the challenges of NMP applications, providing a significant advancement in the field of NMP. Future work could explore further optimizations and refinements in the hardware design to enhance its efficiency and versatility, opening up possibilities for wider adoption in various real-time audio applications, even beyond NMP.

Acknowledgments

This work has been supported by the Italian Ministry for University and Research under the PRIN program (grant n. 2022CZWWKP).

Leonardo Severi's PhD Programme is funded by the European Union in the framework of the Resiliency and Recovery Plan (RRP), within the NextGenerationEU initiative.

References

[1] Cristina Rottondi, Chris Chafe, Claudio Allocchio, and Augusto Sarti. "An Overview on Networked Music Performance Technologies". In: IEEE Access (2016).

[2] Nicola Domini. "Ultralow Latency Audio Processing Via FPGA For Networked Music Performance Applications", Master thesis, 2022.

[3] Diego Bert. "FPGA-based implementation of audio effects for ultralow-latency Networked Music Performance applications", Master thesis, 2023.

[4] Comparison of Remote Music Performance Software. URL: https://en.wikipedia.org/wiki/Comparison_of_Remote_Music_Performance_Software (accessed: 2023/07/29).

[5] Juan-Pablo Cáceres and Chris Chafe. "JackTrip: Under the Hood of an Engine for Network Audio". In: Journal of New Music Research (2010).

[6] Alexander Carôt, Alain B. Renaud, and Bruno Verbrugghe. “Network Music Performance (NMP) with Soundjack". In: Proceedings of the 6th NIME Conference (2006).

[7] Chrisoula Alexandraki, Panayotis Koutlemanis, Petros Gasteratos, Nikolas Valsamakis, Demosthenes Akoumianakis, and Giannis Milolidakis. “Towards the implementation of a generic platform for networked music performance: The DIAMOUSES approach". In: Proceedings of the 2008 International Computer Music Conference (2008).

[8] Carlo Drioli, Claudio Allocchio, and Nicola Buso, “Networked performances and natural interaction via LOLA: Low latency high quality A/V streaming system". In: International conference on information technologies for performing arts, media access, and entertainment (2013).

[9] Matteo Sacchetto, Paolo Gastaldi, Chris Chafe, Cristina Rottondi, and Antonio Servetti, “Web-based networked music performances via WebRTC: a low-latency PCM audio solution". In: Journal of the Audio Engineering Society, 70(11), 926-937, (2022).

[10] Elk Audio. URL: https://www.elk.audio/ (accessed: 2023/07/29).

[11] Andrew Mcpherson and Victor Zappi, “An Environment for Submillisecond-Latency Audio and Sensor Processing on BeagleBone Black". In: Journal of The Audio Engineering Society, (2015).

[12] McPherson, Andrew P and Jack, Robert H and Moro, Giulio, “Action-sound latency: Are our tools fast enough?". In: Proceedings of the International Conference on New Interfaces for Musical Expression, (2016).

[13] Wegener, Clemens and Stang, Sebastian and Neupert, Max, “Fpga-accelerated real-time audio in pure data". In: Proceedings of the 19th Sound and Music Computing Conference, (2022).

[14] Popoff, Maxime and Michon, Romain and Risset, Tanguy and Orlarey, Yann and Letz, Stéphane, “Towards an fpga-based compilation flow for ultra-low latency audio signal processing". In: Proceedings of the 19th Sound and Music Computing Conference, (2022).

[15] Tanguy Risset, Romain Michon, Yann Orlarey, Stéphane Letz, Gero Müller and Adeyemi Gbadamosi “Faust2FPGA for Ultra-Low Audio Latency: Preliminary work in the Syfala project". In: IFC2020 - Internationnal Faust Conference (2020).

[16] George Xylomenos, Christos Tsilopoulos, Yannis Thomas, and George C. Polyzos, “Reduced switching delay for networked music performance” in Proceedings of Packet Video Workshop (Poster Session), (2013).

[17] George Baltas and George Xylomenos, “Ultra low delay switching for networked music performance” in Proceedings of the 5th International Conference on Information, Intelligence, Systems and Applications (IISA), (2014).

[18] Pekka Jääskeläinen, Timo Viitanen, Jarmo Takala, and Heikki Berg. “HW/SW Co-design Toolset for Customization of Exposed Datapath Processors". In: Computing Platforms for Software-Defined Radio (2017).

[19] PCM1803 reference page. URL: https://www.ti.com/product/PCM1803 (accessed: 2023/07/29).

[20] PCM1771 reference page. URL: https://www.ti.com/product/PCM1771 (accessed: 2023/07/29).

[21] MIDI specification page. URL: https://www.midi.org/specifications (accessed: 2023/07/29).

[22] Arty A7 Reference Manual. URL: https://digilent.com/reference/programmable-logic/arty-a7/reference-manual (accessed: 2023/07/29).

[23] Snorre Farner, Audun Solvang, Asbjørn Sæbø, and U. Peter Svensson. “Ensemble Hand-Clapping Experiments under the Influence of Delay and Various Acoustic Environments". In: Journal of the Audio Engineering Society (2009)

[24] Raspberry Pi 4 Model B datasheet. URL: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf (accessed: 2023/07/29).

[25] BCM2711 datasheet. URL: https://datasheets.raspberrypi.com/bcm2711/bcm2711-peripherals.pdf (accessed: 2023/07/29).