Audio DSP to FPGA Compilation

Maxime Popoff, Romain Michon, Tanguy Risset, Pierre Cochard, Stephane Letz, Yann Orlarey, Florent de Dinechin

Audio DSP to FPGA Compilation

Maxime Popoff,[a] Romain Michon,[b] Tanguy Risset,[a] Pierre Cochard,[b]
Stephane Letz,[c] Yann Orlarey,[c] Florent de Dinechin[a]
[a]*Univ Lyon, INSA Lyon, Inria, CITI, EA3720* 69621 Villeurbanne, France. *firstname.lastname@insa-lyon.fr*
[b]*Univ Lyon, Inria, INSA Lyon, CITI, EA3720* 69621 Villeurbanne, France *firstname.lastname@insa-lyon.fr*
[c]*Univ Lyon, GRAME-CNCM, INSA Lyon, Inria, CITI, EA3720* 69621 Villeurbanne, France {*sletz, orlarey*}*@grame.fr*

***Abstract*—The implementation of real-time audio Digital Signal Processing (DSP) applications on FPGA has been extensively studied in the past. Up to now, Audio IPs[1] were designed either "by hand" in VHDL or using predefined IPs in block synthesis environments. The advent of High Level Synthesis (HLS) allows for a real compilation flow from high-level audio DSP specifications down to FPGA bit-streams. This paper presents the principles and the implementation of the first "audio DSP compiler" targeting FPGAs. Our fully open-source system compiles audio DSP programs down to FPGA hardware and up to actual sound production. This compilation flow presents two important technological breakthroughs for audio programmers: achieving ultra-low latency real-time audio DSP (few micro-seconds) and the possibility of easily deploying systems with a large number of audio channels.**

***Index Terms*—HLS, Compilation on FPGA, Audio DSP, Faust**

I. Introduction

Audio Digital Signal Processing (DSP) is used on a wide range of devices. Some audio applications require high throughput, which can be obtained using hardware accelerators such as GPUs, ASICS, or FPGAs. In some specific cases (e.g., dynamic acoustic control), ultra-low latency and/or a large number of audio channels is required, implying the use of dedicated architectures such as FPGAs or ASICS.

The use of a Domain Specific Language (DSL) for audio applications can help reducing the exploration space during hardware compilation. Although many FPGA *compilation* toolchains have been presented for image processing [2], similar flows for audio DSP are rare. [8] proposes a programmable parallel machine implemented on FPGA. [7] presents an open-source IP-based system (using MathWork *HDL coder*). The GAUT HLS tool [3] was dedicated to signal processing applications but did not use a DSL as input.

The compilation flow presented in this paper should ease the prototyping of systems (i) for active acoustic control used in artificial reverberation, noise cancelation, etc., and (ii) involving a large number of audio channels (e.g., spatial audio, ambisonics microphones, etc.). A more complete description of the tool can be found in [5], [6] or on github.[2]

II. FPGA Audio Compilation Tool

Fig. 1 presents the global view of the building blocks of an "audio DSP to FPGA compiler." A single source

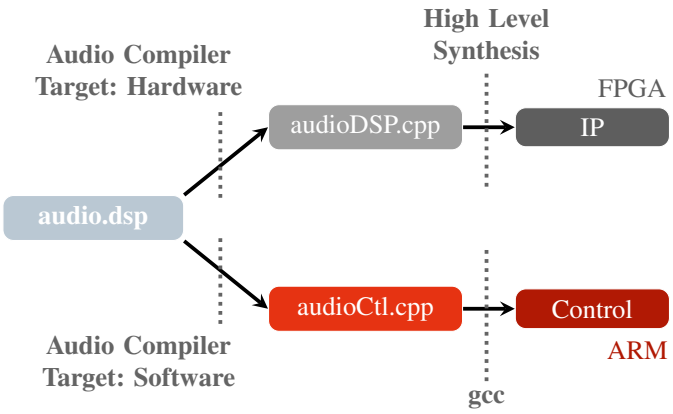

Fig. 1: Audio to FPGA compiler global view.

(`audio.dsp`) generates both hardware (i.e., audio IP) and software (i.e., control program).

For expressing our audio programs, we chose the Faust programming language [4]. Faust[3] is a functional programming language for real-time audio processing which is widely used in the field of music technology. The main strength of the Faust environment is its optimizing compiler targeting many languages such as C++, C, LLVM bitcode, WebAssembly, etc.

Our implementation of the conceptual view of Fig. 1 is presented in Fig. 2. It uses a Faust program as an input (grey boxes are generated during the compilation flow), C++ files are generated by the Faust compiler. The core DSP computation (`audioIP.cpp`) is mapped on the FPGA, the control part (`audioCtl.cpp`) is mapped on the computing system, and the user interface (`audioUI.cpp`) is mapped on the host processor or carried out in hardware. Standard interfaces allow the user to use the same compilation commands for all audio programs. Our compilation flow to VHDL uses a front-end compiler (Faust) and a back-end compiler (*Vitis HLS*/*Vivado*).

The hardware/software partitioning is performed automatically by Faust which is able to distinguish between sample-rate and control-rate computations. This partitionning generates two files: `audioIP.cpp` and `audioCtl.cpp`. As the Faust compiler is open-source, we were able to extend it with a dedicated option: `-os` (*one sample*). Activating this option restricts the generated code to compute a single sample (while many audio compiler generate code for a *buffer* of samples)

[1]Throughout this paper, *IP* stands for *Intellectual Property*, i.e., a circuit component.

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

[3]https://faust.grame.fr/

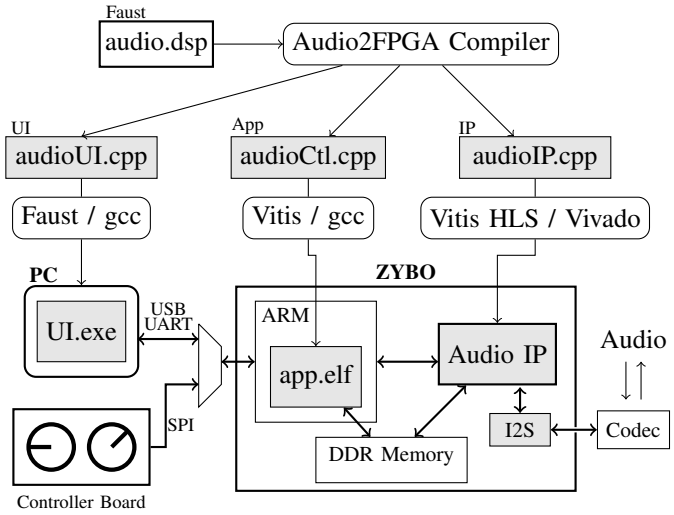

Fig. 2: Audio to FPGA compiler complete view from a FAUST program to Xilinx Zynq board programming.

FAUST DSP	Cycles/sample	DSP	FF	LUT
`fir.dsp` parallel	292	19	7739	15864
`fir.dsp` sequential	2839	8	4417	13741
`bell.dsp` parallel	147	256	33045	46682
`bell.dsp` sequential	310	28	14386	15877

TABLE I: Performance results for two DSP programs Sequential (i.e. no compilation options in Vitis HLS) or parallel version (i.e. using `-unsafe_math_optimizations` compilation option) yield very different complexity results.

allowing for a one sample latency for the generated IP.

Many audio programs use delay lines requiring access to external DDR when FPGA block RAMs are not sufficient. This is a potential important performance bottleneck. Our current strategy for organizing memory is the following: a greedy algorithm first maps small arrays on the FPGA, and switches to DDR mapping when the size of FPGA block RAM is exceeded.

Both a hardware and a software control interface were developed for the system presented here. The hardware interface takes the form of a PCB board with physical controllers (i.e., buttons and potentiometers) interfaced to the ARM processor via an SPI ADC chip. A more flexible software interface can be used on the host computer thanks to the USB/serial connection with the FPGA board. It uses the GTK library on the host and it is automatically generated from the audio DSP program (`audioUI.cpp`).

III. Performance Results

The main contribution of this work is an open-source flow to compile audio DSP programs down to an FPGA with many tunable parameters (i.e., number of input/output audio channels, different audio codec used for each channel, sample rate up to 768 kHz, target FPGA board, etc.) and achieving ultra-low latency.

Table I shows an example of ressource used by two DSP applications (`bell.dsp` with 32 harmonics and `fir.dsp` with 350 coefficients), compiled with or without parallelization optimization from `Vitis HLS`. These optimizations have now to be evaluated more precisely in order to improve the compilation tool. Currently the FAUST compiler flattens the loops in the audio program to exhibit maximum possible parallelism. More performance results related to latency, throughput, and design complexity are presented in [5].

Number of channels	4	8	16	32	256
LUT usage for standard I2S	222	274	386	606	3601
GPIO pin usage for standard I2S	4	6	10	18	130
LUT usage for TDM I2S	159	208	271	358	2878
GPIO pin usage for TDM I2S	4	4	4	6	34

TABLE II: Size and pins usage for standard I2S and our optimized TDM I2S implementation for various numbers of I/O channels.

The main objective of this first compiler version was to obtain very low latency (around 10 μs) from analog input to analog output. The latency performance is presented in [6] for various codecs and various sampling rates. Further dedicated optimization will allow more efficient implementations but will require dedicated studies.

FPGA hardware platforms also enable for the use of many audio channels. Each channel uses standard or TDM[4] I2S protocols for interfacing audio codecs. The size complexity and number of GPIO used are presented in Table II for a standard I2S IPs and for our optimized TDM I2S implementation (40 GPIOs are available on Zybo-Z20). This illustrates the fact that a large number of channels can be handled by our system.

We hope that many audio applications will benefit from this tool, we are currently using it in the context of active acoustic control [1] as well as for spatial audio (WFS and ambisonics).

References

[1] L. Alexandre, P. Lecomte, M.-A. Galland, and M. Popoff. Feeedback Acoustic Noise Control with Faust on FPGA: Application to Noise Reduction in Headphones. IFC 2022, June 2022.

[2] P. Amiri, A. Pérard-Gayot, R. Membarth, P. Slusallek, R. Leißa, and S. Hack. Flower: A comprehensive dataflow compiler for high-level synthesis. In *2021 International Conference on Field-Programmable Technology (ICFPT)*, pages 1–9. IEEE, 2021.

[3] P. Coussy. GAUT: an open-source HLS tool. In *Free Silicon Conference (FSiC)*, Paris, France, Mar. 2019.

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

[5] 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.

[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 *SMC-22 - Sound and Music Computing*, Saint-Étienne, France, June 2022.

[7] T. Vannoy, T. Davis, C. Dack, D. Sobrero, and R. Snider. An open audio processing platform using soc FPGAs and model-based development. In *Audio Engineering Society Convention 147*. Audio Engineering Society, 2019.

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

[4]Time Division Multiple access I2S (TDM I2S) provides up to 16 IS slaves per master.