What's New in the Faust Ecosystem in 2024?

Stéphane Letz, Romain Michon, Yann Orlarey

Proceedings of the International Faust Conference (IFC-24), Soundmit, Turin, Italy, November 21-22, 2024

WHAT'S NEW IN THE FAUST ECOSYSTEM IN 2024?

Stéphane Letz
Univ Lyon, GRAME-CNCM, INSA Lyon, Inria, CITI, EA3720, 69621 Villeurbanne, France
letz@grame.fr

Romain Michon
Univ Lyon, Inria, INSA Lyon, CITI, EA3720, 69621 Villeurbanne, France
romain.michon@inria.fr

Yann Orlarey
Univ Lyon, GRAME-CNCM, INSA Lyon, Inria, CITI, EA3720, 69621 Villeurbanne, France
yann.orlarey@inria.fr

ABSTRACT

This paper provides an overview of the developments in the FAUST programming language and ecosystem since the 2022 International FAUST Conference. Over the past two years, the FAUST community has made significant progress in compiler enhancements, backend integrations, web-based tools, a new Widget Modulation language extension, and FPGA support for audio DSP compilation.

The Emeraude team, a collaboration between Inria, GRAME-CNCM, and INSA Lyon, started its work in March 2022 and has strengthened FAUST's development and application in academic and industrial contexts.

Additionally, a reflective process and proposed consortium aim to empower the user community in guiding FAUST's future direction.

The paper also explores several industrial applications, highlighting the practical impact and versatility of the FAUST ecosystem.

1. INTRODUCTION

The FAUST programming language and its ecosystem are key technological components used by the Emeraude team, particularly in the Syfala project, as discussed in §3.4.1.

Furthermore, a reflective process, presented in §2, has been initiated to strengthen the FAUST project, with plans to establish a FAUST consortium, aiming to provide the user community with a significant role in shaping the future of the project.

FAUST has made significant strides in compiler developments, backends integration, and community projects. Highlights in §3 include:

- New Backends: integration within JAX, JSFX, Cmajor, and RNBO, enhancing FAUST's versatility across various DSP contexts.
- Widget Modulation: enabling developers to effortlessly implement voltage control type modulation to existing Faust circuits.
- Web Developments: introduction of the `faustwasm` and `faust-web-component` packages, modernization of the FAUST IDE, Editor, and Playground for easier web-based DSP integration and update WAM 2.0 plugin model.
- FPGA Support: the Emeraude team's work on providing an audio DSP compilation flow for FPGA platforms, Linux support for Syfala, and development of multichannel audio boards.

Finally, several of the main industrial applications of the FAUST ecosystem are presented in §4.

2. THE FAUST COMMUNITY

2.1. FAUST Consortium

The FAUST project is an open-source initiative hosted on GitHub [1] and freely accessible to the public. While the community has significantly enriched the ecosystem with architecture files, libraries, and more, contributions to the language design and the compiler itself have been minimal so far.

The aim of the FAUST consortium is to give the FAUST user community a stake in the future of FAUST, by giving them the opportunity to see how the language will evolve, and to take an active part in the decision-making process, in particular the development of the roadmap.

Another goal of the FAUST consortium is to gather financial resources to ensure the maintenance and development of FAUST in the years to come.

Here's the current state of our thinking and the resulting proposals, bearing in mind that this is an ongoing endeavour that still needs some work.

2.1.1. Consortium Members

The FAUST Consortium is made up of several categories of members, according to their financial contribution to the Consortium, which determines their level of Membership, and consequently their rights in the running of the Consortium. The different categories are as follows:

- Guest member
- Paying member: platinum, gold, silver

Their rights and obligations will be defined in a "FAUST Consortium Contract" document proposed by InriaSoft [2].

2.1.2. Consortium Organization

The consortium is supported by two governing bodies:

- the Annual General Meeting (AGM)
- the Scientific and Technical Committee (STC).

2.1.3. Annual General Meeting

The General Meeting is the governing body that ensures the smooth running of the FAUST consortium. At its annual meeting:

[1] `https://github.com/grame-cncm/faust`

[2] `https://www.inria.fr/fr/inriasoft-pour-la\-diffusion-des-logiciels-open-source`

- It examines the state of the ecosystem and makes recommendations on future directions and work priorities;
- It examines the consortium's financial situation and approves the annual budget;
- It sets work priorities for the various elements of the roadmap drawn up by the Scientific and Technical Committee;
- It coordinates communication and promotional activities, such as the International FAUST Conference (IFC).

2.1.4. Scientific and Technical Committee

The Scientific and Technical Committee defines the roadmap for the evolution of the compiler and the various tools that make up its ecosystem:

- It is responsible for the official specification of the FAUST language and its evolution;
- It proposes a reference implementation of the compiler, compliant with this specification, and regularly publishes this implementation officially;
- It issues certificates of conformity for any third-party implementations of the compiler to this specification;
- It defines and maintains the standard FAUST libraries, as well as various development tools that are part of the FAUST ecosystem;
- It maintains a set of basic architecture files;
- It develops the language's official documentation and teaching resources;
- It manages the language's official websites and related Git repositories.

The STC is made up of a technical manager appointed by Inria, members of the Emeraude team working on FAUST, and possibly one or two representatives of the members. Consortium members are invited to suggest topics for the agenda of STC meetings.

2.1.5. Drawing Up the Roadmap

The roadmap defines short and medium-term developments for the language, the compiler and the various tools in the FAUST ecosystem. A first, fairly broad version is drawn up by the STC, based on proposals from the FAUST community and consortium members. In particular, the STC assesses the feasibility and labor costs of the various points, and proposes an initial ranking according to their importance and dependence.

The General Assembly selects and prioritizes the items to be included in the roadmap from the STC's proposals. It ensures that the necessary workload does not exceed 50% of the consortium's available human resources. The roadmap is then officially adopted by the General Assembly in a vote in which each member has a number of votes corresponding to its level of membership.

2.2. Communication Channels

The now automomous FAUST Discord channel [3] is an active and dynamic online community space dedicated to users, developers. This platform serves as a hub for real-time communication, collaboration, and support, fostering a sense of community among members with varying levels of expertise (see Figure 1).

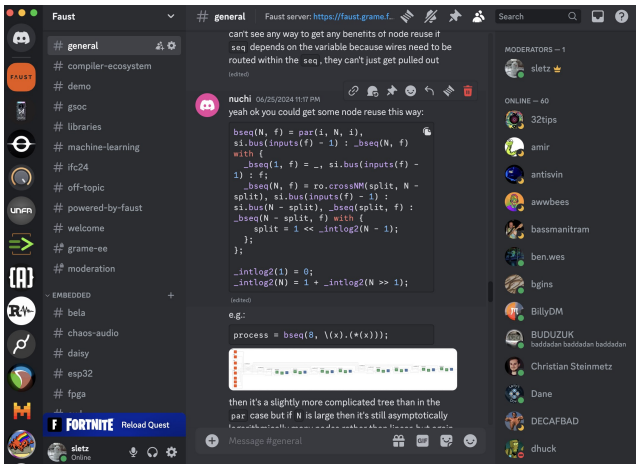

Figure 1: *The Faust Discord channel.*

2.3. The "Powered by FAUST" Page

A page listing all the significant "Powered by FAUST" projects is maintained: musical pieces or artistic projects, plugins, standalone applications, integration in audio programming environments, development tools, research projects, embedded devices, Web applications, etc. are listed.

This page is regularly enriched and as of July 2024, more than 250 projects are described (see Figure 2).

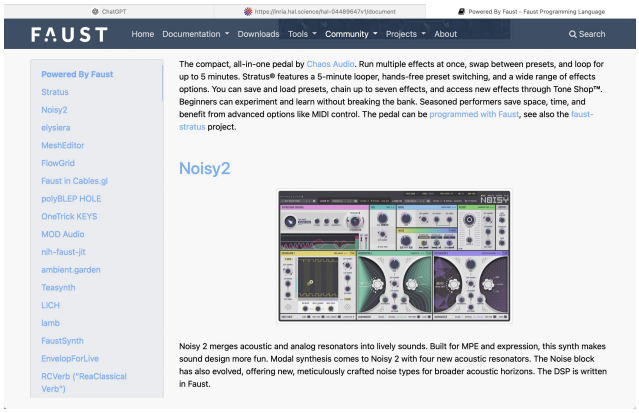

Figure 2: *Excerpt of the "Powered by Faust" list.*

3. DEVELOPMENTS

3.1. New Backends

Four new backends have been developed. They allow us to use FAUST DSP programs with a larger set of targets in new applications, or to reach new communities.

[3] https://faust.grame.fr/community/help/#faust-on-discord

3.1.1. JAX

The introduction of the JAX backend opens up a new domain of exploration for FAUST, significantly expanding its capabilities and potential applications, especially in the field of machine learning.

JAX is a library for high-performance numerical computing, particularly popular in the machine learning research community for its flexibility in model development and its capability to handle complex mathematical operations efficiently. It extends the capabilities of NumPy by enabling automatic differentiation, allowing us to compute gradients of functions with respect to their inputs. JAX also supports just-in-time (JIT) compilation to highly optimized machine code, which significantly improves performance for numerical routines. It supports the creation and training of neural networks through libraries built on top of it, like Flax and Haiku.

With the assistance of GRAME, David Braun has contributed a JAX backend allowing for the direct creation of differentiable FAUST programs for potential uses in machine learning applications. It is designed to be used within DawDreamer [4], an audio-processing Python framework that supports core DAW features and more, also developed by David Braun.

Several Flax examples with a learnable low-pass filter, a differentiable subtractive synthesizer, a differentiable polyphonic wavetable synthesizer whose wavetables are learnable, as well as a parametric equalizer written in FAUST to a QDax environment, have been explored [5].

3.1.2. JSFX

Developed by Cockos, the creators of Reaper, JSFX [6] is a scripting language which allows users to extend the capabilities of the DAW with custom audio and MIDI processing scripts adapted to their specific needs.

With the assistance of GRAME, Johann Philippe has contributed a backend enabling the creation of synthesizers and effects with MIDI control, as well as polyphonic MIDI-controllable audio plugins. Following the standard JSFX file structure, several `@init`, `@block`, `@slider` and `@sample` blocks are filled with the appropriate part of the generated FAUST code. The result is self-contained, with the architecture part directly inserted by the backend (even the voices allocation and MIDI control in polyphonic mode), allowing it to be loaded and executed in Reaper seamlessly. A comprehensive tutorial has been written. [7]

3.1.3. Cmajor

Cmajor is a C like procedural high-performance language specifically designed for audio processing providing a runtime with dynamic LLVM JIT based compilation. Cmajor is intended to complete other technologies like C++, JUCE, and CLAP, and also support Web export. DSP is deployed with Cmajor as a patch, which includes a description of the plugin, the source code for the DSP, and an associated GUI implemented in JavaScript. The language supports a signal flow through a graph structure with nodes containing implementations of specific DSP building blocks.

A Cmajor backend has been written to generate a processor from a FAUST DSP program. Parameters such as sliders, buttons, and bar graphs correspond to Cmajor's concept of input and output events. The actual sample computation code is generated within the essential `run()` function.

A `faust2cmajor` script enables the creation of ready-to-use Cmajor patches, which can be directly executed with the `cmaj` runtime, or possibly exported as C++, or JUCE, CLAP, Web plugins. The script supports both regular DSP programs and polyphonic MIDI-controllable programs. A comprehensive tutorial has been written. [8]

3.1.4. RNBO

RNBO is a library and toolchain that can take Max-like patches, export them as portable code, and directly compile that code to targets like a VST, a Max External, or a Raspberry Pi.

FAUST programs can be compiled to the internal `codebox~` sample level scripting language. In this model, several sections are generated for parameter definitions, DSP state construction, initialization, the `control` function (called once per block), and finally, the `compute` function (called at audio rate).

The `faust2rnbo` tool transforms a FAUST DSP program into a RNBO patch containing a `rnbo~` object and including the codebox code (generated using the codebox backend) as a subpatch. Needed audio inputs/outputs and parameters (with the proper name, default, min and max values) are automatically added in the patch. Additional options allow us to generate a specific version of the RNBO patch used in the testing infrastructure. The code is written in Python and uses the very powerful `py2max` [9] library to generate the maxpat JSON format.

The script supports both regular DSP programs and polyphonic MIDI-controllable programs. A comprehensive tutorial has been written. [10]

3.2. Widget Modulation

An extension to the FAUST programming language, *Widget Modulation*, has been recently developed. Inspired by *Modular Synthesizer*, this extension enables developers to effortlessly implement *voltage control type* modulation to existing Faust circuits.

Although signal modulation can traditionally be achieved by writing the necessary code during circuit development, *Widget Modulation* expressions enable it *a posteriori*, after the circuit has been developed and without modifying its code. This feature allows for direct *reuse and customization* without prior planning by the original circuit designer. It offers a new level of expressivity and flexibility in FAUST circuit design. A separate paper on the subject has been proposed to IFC 2024 [7].

3.3. Web-Related Developments

The development of the FAUST to Web glue code started in 2014, initially as a collection of JavaScript files. To streamline maintenance and facilitate future development, the compiler's C++ export layer has been modernized using the Embind model, [11] which is

[4] `https://github.com/DBraun/DawDreamer`

[5] `https://github.com/DBraun/DawDreamer/tree/main/examples`

[6] `https://www.reaper.fm/sdk/js/js.php`

[7] `https://faustdoc.grame.fr/tutorials/jsfx/`

[8] `https://faustdoc.grame.fr/tutorials/cmajor/`

[9] `https://github.com/shakfu/py2max`

[10] `https://faustdoc.grame.fr/tutorials/rnbo/`

[11] `https://emscripten.org/docs/porting/connecting_cpp_and_javascript/embind.html`

part of the Emscripten compiler.[12] Additionally, the FAUST to WebAudio glue code has been completely restructured and rewritten in TypeScript, developed and distributed as a separated npm package. Several projects have been developed using this new framework, demonstrating its robust performance and ease of integration.

3.3.1. The faustwasm Package

The FaustWasm library [13] provides a user-friendly high-level API that wraps around the FAUST compiler. While the interface is primarily tailored for TypeScript, it also includes API descriptions and documentation for pure JavaScript users. This WebAssembly version of the FAUST Compiler, suitable for both Node.js and web browsers, has been compiled with Emscripten 3.1.31.

The library enables the compilation of FAUST DSP code into WebAssembly, allowing it to be used as WebAudio nodes within a standard WebAudio node graph. It also supports offline rendering scenarios. Additionally, tools are available for generating SVGs from FAUST DSP programs.

Users can create "mono" synthesizer and effect nodes, as well as polyphonic nodes. MIDI support is automatically activated when MIDI metadata is included in the DSP code for mono nodes, and is always enabled in polyphonic mode.

Sensors (accelerometer and gyroscope) are supported, as well as the Progressive Web Application model, so playable instruments to be used on smartphones and tablets can be easily deployed.

3.3.2. Modernized Faust IDE, Faust Editor and FaustPlayground

As the main outcomes of Ian Clester's Google Summer of Code 2023 projects, [14] the three FAUST IDE, FAUST Editor and FaustPlayground projects have been modernized with updated build tools, and the use of the `faustwasm` package.

3.3.3. faust-web-component

Another outcome of Ian Clester's Google Summer of Code project is the `faust-web-component` [15] package which provides two web components for embedding interactive FAUST snippets in web pages:

- `<faust-editor>` displays an editor (using CodeMirror 6) with executable, editable FAUST code, along with some bells & whistles (controls, block diagram, plots) in a side pane. This component is ideal for demonstrating some code in FAUST and allowing the reader to try it out without having to leave the page.
- `<faust-widget>` just shows the controls and does not allow editing, so it serves simply as a way to embed interactive DSP, and can been tested here.

These components are built on top of `faustwasm` and `faust-ui` [16] packages and are released as an npm package.

3.3.4. Web Audio Module (WAM) 2.0

In 2015, Jari Kleimola and Olivier Larkin proposed Web Audio Modules (WAM), a standard for Web Audio plugins and DAWs. The 2.0 version [1], released in 2021, was a collaborative effort involving many contributors, resulting in multiple open source and free software plugins and hosts. WAM 2.0 includes an SDK, an abstract API, numerous open source repositories with various plugins, tutorials, and several hosts demonstrating WAM capabilities. The design of WAM 2.0 aimed to support diverse development workflows, from web developers using plain JavaScript, React developers, to C/C++ developers cross-compiling code to WebAssembly.

WAM 2.0 [2] plugins can be developed using FAUST and easily generated using the FAUST IDE [17], with the adapted targets [18].

3.4. Emeraude Team Projects

The Emeraude team is continuing its work on the FAST ANR project [19], initiated in 2021. This project aims to facilitate high-level programming of FPGA-based platforms for multichannel ultra-low-latency audio processing using FAUST.

3.4.1. Syfala: Compilation of Audio DSP on FPGA

The team has been actively extending the Syfala toolchain, first released in 2022 [3]. It is meant to be a powerful audio to FPGA compilation toolchain. All the possible use of the compilation toolchain have been combined in a single software suite. This section describes the extensions that have been added to Syfala in 2023.

When compiling FAUST programs to FPGA, Syfala relies on the High Level Synthesis (HLS) tool provided by Xilinx, which takes a C++ program as an input. Hence, FAUST generates C++ code from a FAUST program and Syfala feeds it to HLS. The topology of the C++ code provided to HLS has a huge impact on the performances of the generated Intellectual Property (IP). In 2023, a study has been conducted aiming at understanding the kind of optimizations that can be carried out on C++ code in the context of the high-level synthesis of real-time audio DSP programs.

Thanks to this work, the applications generated by Syfala have been significantly optimized, allowing for much more complex audio DSP algorithms to be run on the FPGA. While these findings haven't been integrated to the FAUST Syfala backend, they can be used with the new Syfala C++ support. Indeed, a new mode in Syfala allowing for C++ code to be used as a substitute for FAUST has been added. This, combined with an exhaustive public documentation of the aforementioned optimizations will help increasing the attractivity of Syfala.

3.4.2. Linux Support for Syfala

Most modern FPGA boards host a CPU SoC tightly coupled to the FPGA. Real-time audio DSP applications running on such boards can leverage the CPU of the board to carry out control computations or to provide high-level functionalities (i.e., user interface,

[12] `https://emscripten.org/index.html`
[13] `https://github.com/grame-cncm/faustwasm`
[14] `https://ijc8.me/2023/08/27/gsoc-faust/`
[15] `https://github.com/grame-cncm/faust-web-component`
[16] `https://github.com/Fr0stbyteR/faust-ui`
[17] `http://www.webaudiomodules.com/docs/usage/generate-with-faustide`
[18] `https://faustdoc.grame.fr/manual/deploying/#exporting-wam-20-plugins`
[19] `https://fast.grame.fr`

external controllers, etc.) [3]. Up to now, the CPU portion of applications generated by Syfala was implemented as a bare-metal kernel. In 2023, the possibility to run Alpine Linux on the CPU of the Zybo board while carrying out audio DSP operations on the FPGA has been added, taking a hardware accelerator approach. This enables the compilation of complete audio applications involving various control protocols and approaches such as OSC (Open Sound Control) through Ethernet or Wi-Fi, MIDI, web interfaces running on an HTTPD server, etc. It also opens the door to the integration of hardware accelerators in high-level computer music programming environments such as Pure Data, SuperCollider, etc.

This work led to a publication at the 2023 Sound and Music Computing conference (SMC-23)[4].

3.4.3. Syfala PipeWire Support

During the work on applications for Syfala requiring the handling of a large number of audio channels in parallel for spatial audio, a way to send and receive audio streams in parallel between a laptop computer and our FPGA board has been developed. For this, an open standard named PipeWire, which allows for the transmission of digital audio streams in real-time over an ethernet connection has been chosen. PipeWire was implemented in the Linux layer of Syfala and is now perfectly integrated to the toolchain. It will allow us to significantly expand the scope of the various spatial audio systems that has been working on in the context of the PLASMA project.

3.4.4. Multichannel Audio Boards for FPGA

Two audio FPGA sister boards aiming various kinds of spatial audio applications have been developed:

- One targets the Digilent Zybo Z7 (10 or 20) board and is designed to be cost-efficient, accessible, and easily reproducible. It provides 32 amplified (3W) audio outputs to which small speakers can be directly connected. Its goal is to provide an affordable way to deal with a large number of audio outputs in the context of spatial audio.
- The other board that has been developed is meant to be connected to a Digilent Genesys board and targets high-end spatial audio applications with a strong focus on active control. It provides 32 ultra-low latency (10us) balanced inputs and outputs. It is currently used as part of the FAST ANR project for implementing FxLMS algorithms for active control.

This work has been published at the 2024 at the Sound and Music Computing conference (SMC-24) [5].

3.4.5. FAUST *to VHDL Backend*

Syfala uses HLS for compiling C++ code down to VHDL, the C++ code being itself generated from FAUST . However, FAUST , as a functional language, exhibits all the parallelism of the audio application. The code is sequentialized in the C++ code and then re-parallelized by the viti_hls tool for the FPGA.

An interesting alternative is to translate directly FAUST down to VHDL. FAUST programs can be represented as audio circuits connected together and hence provides a natural equivalence with VHDL structural representation of such circuits. The VHDL program is just a translation of the data-flow graph of the audio application.

However, for an efficient implementation on FPGA, this data-flow graph must be retimed. Retiming is an old classical transformation that adds registers in a digital circuit without changing its functional behaviour but allowing for a much faster clock rate.

A first `Faust2VHDL` translator prototype was issued in 2022 generating a fully combinatorial data path on the FPGA. In 2023 the first real `Faust2FPGA` compiler which includes retiming and fixed point computations has been released.

Preliminary results shows that the IP generated by our `Faust2FPGA` compiler are twice smaller than the IP generated by viti_hls. However, the use of HLS is still preferred because many features are not included in the `Faust2FPGA` compiler (i.e., control from the ARM processor or use of the external RAM).

3.4.6. Fixed-Point Extension for the FAUST *Programming Language*

This recent paper [6] addresses the challenge of efficiently utilizing fixed-point arithmetic in FAUST. Instead of the standard floats format, fixed-point arithmetic can be more resource-efficient and faster than floating-point arithmetic, particularly on FPGAs where the required circuitry can be precisely configured. However, it implies the careful determination of number formats at each step of the computation tree.

The need to reconsider the representation of real numbers in this context is highlighted, where fixed-point numbers, represented as scaled integers, can offer significant efficiency improvements. The introduced key concept is "pseudo-injectivity" which ensures that output values of each function in the language retain the necessary precision. The method extends the previously existing interval range analysis to determine the range of values variables can take and error analysis to manage rounding and ensure precision.

Enhancements to the FAUST compiler to facilitate automatic fixed-point format determination have been done. The precision constraints are propagated through the signal graph to maintain pseudo-injectivity. Additionally, when generating C++ code, a `sfx_t` macro is added in the generated code at each step of the computation, to represent fixed-point formats with the most significant bit (MSB) and the least significant bit (LSB) values.

Results from testing on FAUST programs, such as sine wave generation and the Karplus-Strong string synthesis algorithm, indicate that the method can maintain high audio quality, though inferred formats tend to be wider than necessary. Future improvements will include backward propagation of precision constraints and targeted optimizations to further refine the fixed-point format determination.

3.5. PLASMA: Pushing the Limits of Audio Spatialization with eMerging Architectures

Plasma (Pushing the Limits of Audio Spatialization with eMerging Architectures) is an associate research team gathering the strength of Emeraude and of the Center for Computer Research in Music and Acoustics (CCRMA) at Stanford University. [20]
The two main objectives of Plasma are:

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

- Exploring various approaches based on embedded systems towards the implementation of modular audio signal processing systems involving a large number of output channels (and hence speakers) in the context of spatial audio.
- Making these systems easily programmable to create an open and accessible system for spatial audio where the number of output channels is not an issue anymore.

Two approaches are being considered in parallel:

- Distributed using cheap simple embedded audio systems (i.e., Teensy, etc.),
- Centralized using an FPGA-based (Field-Programmable Gate Array) solution based on the multichannel interfaces presented in §3.4.4.

The focus is on enhancing the hardware and computational capabilities of current spatial audio systems, rather than on the DSP algorithms for spatial audio themselves. FAUST plays a central role in this project by allowing us to deploy spatial audio and virtual acoustics programs from the same source in a generic way.

4. INDUSTRIAL APPLICATIONS

Here is a non-exhaustive list of some recent industrial applications of FAUST.

4.1. Expressive E

Expressive E [21] is a French company that creates innovative musical instruments and software designed to enhance expressive performance. Their products include the Osmose, a standalone expressive synthesizer, and Touché, a device that adds tactile control to existing synthesizers.

They also offer a range of software instruments and sound libraries, such as Noisy and Imagine, which are especially designed to be used with the Osmose and Touché devices, but are versatile enough be used with other MIDI controllers and digital audio workstations (DAWs).

Noisy 1 and 2 products were largely created using FAUST, and benefited from a close collaboration between Expressive E's development team and GRAME, in particular in developing performance measurement and optimisation tools.

Figure 3: *The Noisy2 plugin interface.*

[21] https://www.expressivee.com

4.2. Punk Labs

Punk Labs LLC is a tiny studio of just two people creating apps, games, music, and even social networks. They develop for desktops and game consoles, mobile and embedded devices. Four plugins have been developed using the Rust NIH-plug framework and FAUST for DSP[22], which allows us to develop and export VST3 and CLAP format, as well as a standalone module:

- OneTrick KEYS: a physically modeled piano synth with a lo-fi sound.
- OneTrick URCHIN: an hybrid drum synth that models the gritty lo-fi sound of beats from vintage records without sampling.
- OneTrick CRYPTID: whispers of a drum machine with the cold clanging heart of a DX7 in the fearsome frame of a TR-808 echo in dusty backrooms of backstreet recording studios.
- OneTrick SIMIAN: crash into the 80s with an open source drum synth inspired by hexagonal classics like the Simmons SDS-V.

4.3. Joué Play

The Joué Play [23] is a system that combines an expressive multi-instrument, an intuitive app and interactive content, with a range of musical instruments that use touch-sensitive technology to create a unique playing experience. These instruments are designed to be highly expressive, allowing musicians to play with greater nuance and emotion. Part of the audio effects are coded in FAUST.

5. ACKNOWLEDGMENT AND CONCLUSION

This paper reflects the richness and diversity of the contributions carried out during the past two years. The significant advancements detailed herein are the result of the collaborative efforts of numerous individuals and teams.

Thanks to all contributors for all the different components and projects that have been described, and especially this time: Johann Philippe, David Braun, Shihong Ren, Michel Buffa, Ian Clester, and the Emeraude team: Tanguy Risset, Romain Michon, Pierre Cochard, Florent de Dinechin, Anastasia Volkova, Thomas Rushton, Maxime Popov and Agathe Herrou.

6. REFERENCES

[1] Michel Buffa, Shihong Ren, Owell Campbell, Tom Burns, Steven Yi, Jari Kleimola, and Oliver Larkin, "Web Audio Modules 2.0: An Open Web Audio Plugin Standard," in *Companion Proceedings of the Web Conference 2022 (WWW 22)*, Lyon, France, April 2022.

[2] Shihong Ren, Stéphane Letz, Yann Orlarey, Dominique Fober, Romain Michon, Michel Buffa, and Laurent Pottier, "Modernized Toolchains to Create JSPatcher Objects and WebAudioModules from Faust Code," in *Proceedings of the Web Audio Conference (WAC-22)*, Cannes, France, July 2022.

[22] https://github.com/robbert-vdh/nih-plug
[23] https://jouemusic.com/en

[3] Maxime Popoff, Romain Michon, Tanguy Risset, Yann Orlarey, and Stéphane Letz, "Towards an FPGA-Based Compilation Flow for Ultra-Low Latency Audio Signal Processing," in *Proceedings of the Sound and Music Computing Conference (SMC-22)*, Saint Etienne, France, June 2024.

[4] Pierre Cochard, Maxime Popoff, Antoine Fraboulet, Tanguy Risset, Stéphane Letz, and Romain Michon, "A Programmable Linux-Based FPGA Platform for Audio DSP," in *Proceedings of the Sound and Music Computing Conference*, Stockholm, Sweden, 2023.

[5] Maxime Popoff, Romain Michon, and Tanguy Risset, "Enabling Affordable and Scalable Audio Spatialization With Multichannel Audio Expansion Boards for FPGA," in *Proceedings of the 2024 Sound and Music Computing Conference*, Porto, Portugal, July 2024.

[6] Agathe Herrou, Florent de Dinechin, Stéphane Letz, Yann Orlarey, and Anastasia Volkova, "Towards Fixed-Point Formats Determination for Faust Programs," in *Proceedings of the Journées d'Informatique Musicale (JIM-24)*, Marseille, France, May 2024.

[7] Yann Orlarey, Stéphane Letz, Romain Michon, and the Emeraude team, "Widget Modulation in Faust," in *Proceedings of the International Faust Conference (IFC-24)*, Turin, Italy, November 2024.