# The Faust Programming Language As a Platform for Creating Hybrid Acoustical and Digital Musical Instruments

Romain Michon, Yann Orlarey, Stephane Letz, Dominique Fober

# THE FAUST PROGRAMMING LANGUAGE AS A PLATFORM FOR CREATING HYBRID ACOUSTICAL AND DIGITAL MUSICAL INSTRUMENTS

**Romain Michon**[a,b] **Yann Orlarey**[a] **Stéphane Letz**[a] **Dominique Fober**[a]
[a] GRAME-CNCM, 11 cours de Verdun-Gensoul, Lyon (France)
[b] CCRMA, Stanford University (USA)
michon@grame.fr

## ABSTRACT

FAUST is a functional programming language for real-time audio digital signal processing. The Faust compiler can generate code in lower-level programming languages such as C, C++, JAVA, LLVM bitcode, WebAssembly, etc. Code generated by the FAUST compiler can be turned into a wide range of objects (e.g., audio plug-ins, smartphone apps, web apps, etc.) for various platforms (e.g., Linux, Windows, MacOS, etc.).

By also allowing us to target embedded systems such as microcontrollers, DSPs, FPGAs, embedded Linux systems, etc. and by providing a set of tools to implement physical models of musical instruments, FAUST is particularly well suited to design hybrid acoustical and digital musical instruments mixing physical and virtual elements.

After giving an overview of the aforementioned tools, we demonstrate how they can be used for designing novel hybrid instruments.

## 1. INTRODUCTION

Hybrid acoustical and digital musical instruments combine physical and virtual elements in an attempt to take the best of both "worlds" [1]. There are many examples of such instruments both in the industry and in academia. The Korg WaveDrum [1] can be regarded as one the first member of this family. It combines a physical drum membrane to a digital resonator allowing for the implementation of a wide range of physical models using modal synthesis [2], waveguide modeling [3], etc. The acoustical excitations captured on the membrane are directly fed to the resonator, allowing for a very natural and intuitive control of the instrument. The same approach has been exploited by Dan Schlessinger [4] to implement his Kalichord, an instrument that captures sound excitations created on plastic tines to drive string physical models. More recently, Ali Momeni created Caress, which provides a form of playground for fingers where different textures can be touched to drive various kind of resonators running on a computer [5]. Of course, these are just a few examples.

Designing such instruments presents various challenges, especially if the goal is to make them standalone. Indeed, there only exists a few software and hardware platforms that can be used to reach that goal. The "ideal" platform to implement hybrid instruments should: be easily programmable, provide sensor inputs, have multiple audio inputs and outputs, be easily embeddable.

While most hardware platforms used by the music technology/computer music communities provide such features (e.g., the Raspeberry Pi [2] combined with an Arduino [3] and an external audio interface, etc.) their programming remains challenging and few software environments allow to target all these platforms from a single standpoint. `libpd` [6] allows for the embedding of PD patches into C++ programs, Mobile CSOUND [7] can be used to embed CSOUND programs in mobile apps, etc.

FAUST [8] is a functional programming language for real-time audio signal processing that can be used to target a wide range of standards and platforms. FAUST can be used to implement standalone programs, mobile apps, audio plugins, web apps, etc. In recent years, it has also been increasingly used to program embedded platforms such as embedded Linux systems like the Rapsberry Pi as well as powerful microcontrollers targeting real-time audio processing like the Teensy. [4] In parallel of that, FAUST provides a wide range of signal processing libraries implementing an extended number of resonators and musical instrument physical models which can all be used to design hybrid instruments [9]. In particular, the FAUST physical modeling toolkit [10] offers an ideal environment to prototype hybrid musical instruments. In that context, it was used at the heart of various hybrid musical instruments – some of which are presented in the following sections.

In this paper, we give an overview of how FAUST can be used to design hybrid musical instruments using various platforms and approaches. We first demonstrate how mobile platforms, embedded Linux systems, and microcontrollers can be programmed with FAUST. We then give examples of instruments based on these systems before presenting future directions for this type of work.

[1] `https://www.korg.com/us/products/drums/wavedrum_global_edition/` – All URLs were verified on March 10, 2020.

[2] `https://www.raspberrypi.org/`

[3] `https://www.arduino.cc/`

[4] `https://www.pjrc.com/teensy/`

## 2. FAUST ON MOBILE DEVICES

Mobile devices have been used as the basis of various hybrid instruments for the past few years [11]. They provide a well-suited platform for this type of applications with built-in audio inputs and outputs, potential sensor inputs through a mircocontroller connected to them through MIDI over USB, and they are completely standalone/battery powered.

FAUST can be used to program both Android and iOS devices for real-time audio signal processing applications. `faust2android` and `faust2ios` [12] can be used to turn a FAUST program into a ready-to-use Android or iOS application (respectively). The Graphical User Interface (GUI) of the app will be declared in the corresponding FAUST code (see Figure 1) and will hence be made out of sliders, buttons, knobs, groups, etc. `faust2smartkeyb` [13] on the other hand can target both iOS and Android from a single standpoint and replaces the standard FAUST GUI with a SMARTKEYBOARD interface declared in the FAUST code as a metadata (see Figure 2).

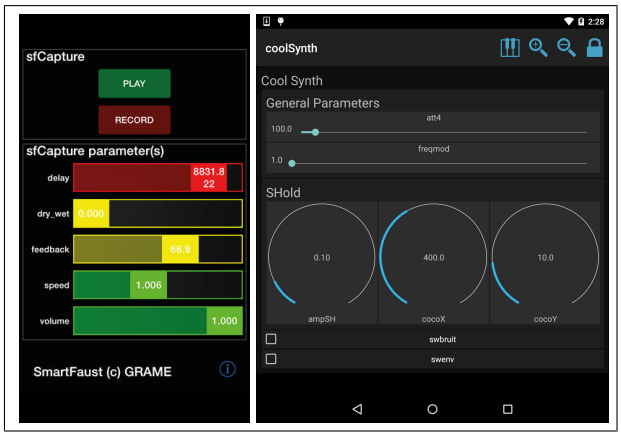


**Figure 1**. Example of a Graphical User Interfaces generated by `faust2ios` (on the left) and `faust2android` (on the right).

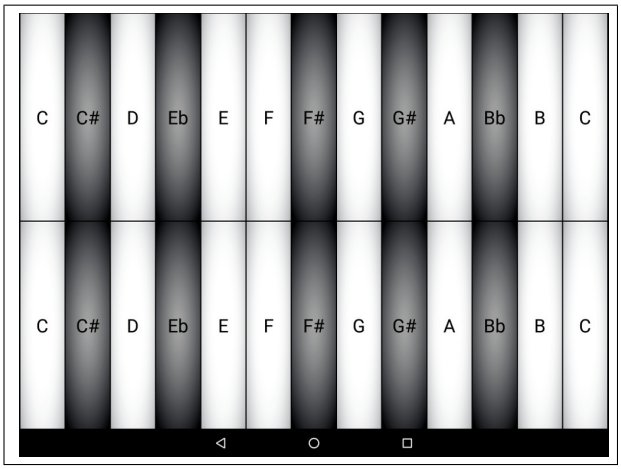


**Figure 2**. Simple SMARTKEYBOARD interface.

In all cases, audio inputs can be easily retrieved in FAUST to drive the digital part of the instrument and sensors can be mapped to sound synthesis parameters using MIDI metadata. [5] Additionally, the built-in motion sensors of the device (e.g., accelerometers, gyroscopes, etc.) can also be mapped to the parameters of a FAUST program using metadata. [6]

## 3. FAUST ON EMBEDDED LINUX SYSTEMS

FAUST is very well supported on Linux and can therefore target embedded Linux systems in many ways. For instance, any of the Linux FAUST target can be used on this type of platform (e.g., Alsa standalones, Jack standalones, PureData externals, CSOUND opcodes, web apps, etc.).

As for mobile platforms, audio inputs can be retrieved in a FAUST program from the built-in or external audio interface of the system, and sensors can be mapped to the sound synthesis parameters using MIDI metadata.

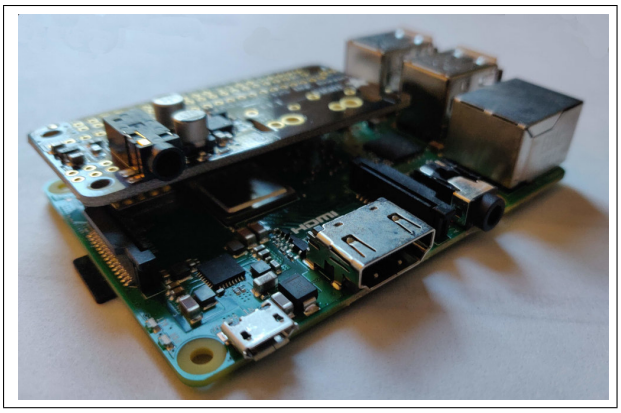

**Figure 3**. The Raspberry Pi 3B equipped with an I2S audio hat providing a stereo PCM audio output.

Specific FAUST architectures targeting the Raspberry Pi (see Figure 3) are also available: `faust2rpialsaconsole` and `faust2rpinetjackconsole`. They can be used to generate command-line applications (e.g., to be used through SSH) using an Alsa or a Jack audio driver (respectively). They differ from "standard architecture" by carrying out C++ compilation optimizations directly linked to the processor architecture of the RPI.

Finally, FAUST can also target the BELA [14] and the Elk [7] which are specific kinds of embedded Linux systems where audio signal processing tasks are ran outside of the operating system. They provide a very appealing platform to create hybrid musical instruments.

## 4. FAUST ON MICROCONTROLLERS

As microcontrollers became more powerful in recent years and provided a Floating Point Unit (FPU), they became a viable option for real-time signal processing applications.

[5] `https://faust.grame.fr/doc/manual/index.html#midi-and-polyphony-support`

[6] `https://faust.grame.fr/doc/manual/index.html#sensors-control-metadatas`

[7] `https://elk.audio/`

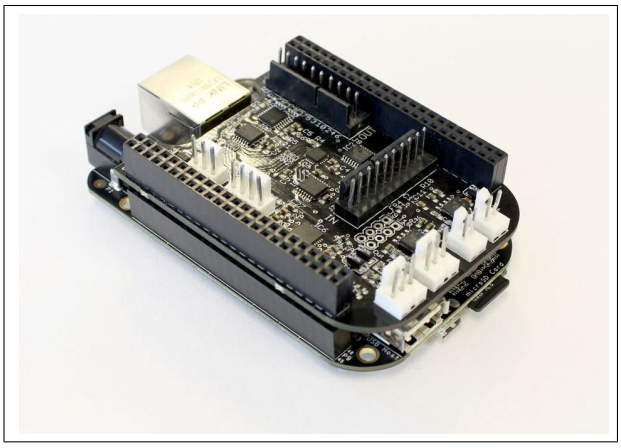

**Figure 4**. The BELA.

They offer a wide range of advantages over embedded Linux systems such as low audio latency (no Operating System), lightness, low power, built-in sensors inputs, and low cost (full systems including audio inputs and outputs can be found for less than $15). On the other hand, their limited memory (which is often less than 1MB) and CPU power are their main flaw. It can quickly become a problem when using algorithms with large memory footprints (e.g., echo, reverb, etc.) or too CPU demanding.

FAUST can be used to program the Teensy microcontroller (see Figure 5) through `faust2teensy` which generates DSP objects for the Teensy Audio Library [15]. This system doesn't provide MIDI support yet but it is not necessarily a problem since sensors can be mapped to DSP parameters directly from the microcontroller itself.

Similarly, FAUST can now be used to produce DSP engines for the ESP32 microcontroller (see Figure 6) thanks to `faust2esp32` [16]. The ESP32 is used at the heart of a wide range of cheap audio development boards such as the Lyra [8] and the LilyGO TTGO T-Audio, [9] which have been increasingly used by the music technology community. `faust2esp32` can also be used to fully program some ESP32-based boards such as the TTGO T-Audio without writing a single line of C++ or Arduino code.

## 5. EXAMPLE OF HYBRID INSTRUMENTS MADE WITH FAUST

FAUST has been used at the heart of a wide range of hybrid musical instruments. Some of them are presented in this section.

### 5.1 The BladeAxe

The BLADEAXE [17] is an instrument based on an iPad where plastic tines equipped with piezo contact microphones are used to capture the sound of the plucks of the performer to drive a series of physical models implemented in FAUST. `faust2smartkeyb` was used to implement

[8] `https://www.espressif.com/en/products/hardware`

[9] `https://github.com/LilyGO/TTGO-TAudio`

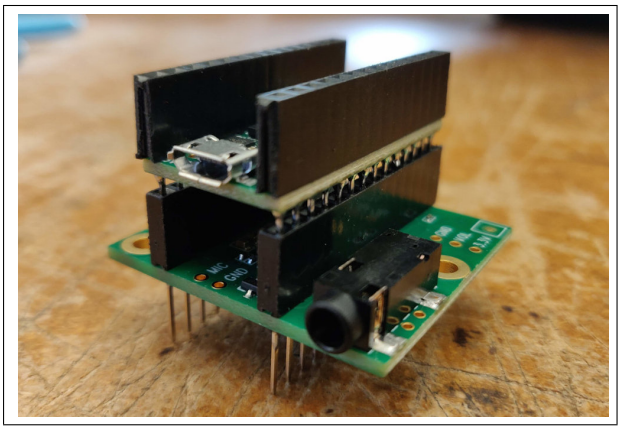

**Figure 5**. The Teensy 4.0 and its audio shield.

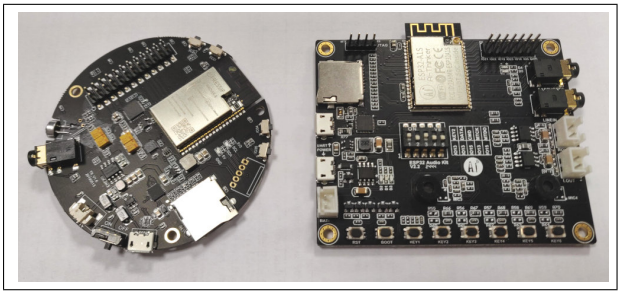

**Figure 6**. ESP32-based audio Processing boards (the TTGO T-Audio on the left and the ESP32 Audio Dev Kit on the right).

the app running the models on the iPad. Models were implemented using the FAUST physical modeling toolkit [10].

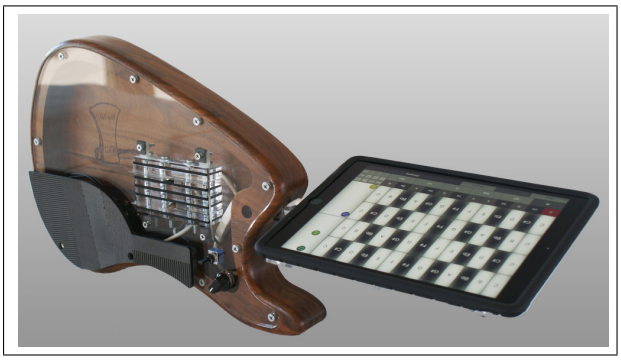

**Figure 7**. The BLADEAXE.

The BLADEAXE has been featured in many performances and continues to be used on stage by the first author of this paper.

### 5.2 Instruments for Music 250a

Music 250a *Physical Interaction Design for Music* is a class offered at the Center for Computer Research in Music and Acoustics every year in Winter quarter. Students learn how to make musical interfaces, hybrid instruments, fundamentals of electronics, microcontroller programming, etc. FAUST has been used to teach this class for the past three years and many student final projects feature hybrid

instruments.

Luigi Sambuy's Sweeps and Collisions is a good example of that (see Figure 8). This instrument allows to roll balls on a plastic surface equipped with piezo contact microphones. The sound of the balls rolling is captured and used to drive resonators running on a Teensy microcontroller that was programmed using `faust2teensy`.

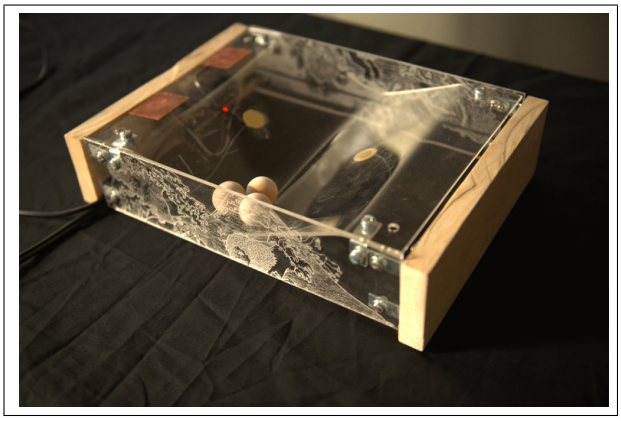

**Figure 8**. Luigi Sambuy's Sweeps and Collisions.

More examples of such instruments can be found on the 2019 and 2020 Music 250a courses websites. [10]

## 6. FUTURE WORKS

We plan to continue facilitating the design of hybrid instruments with FAUST by supporting new platforms and by sustaining the expansion of the FAUST DSP libraries.

On the hardware side, we're currently investigating the possibility of programming FPGA-based boards with FAUST in the context of the SyFaLa project. [11] While we already have a working prototype, much work remains to be done to optimize generated objects. FPGAs could offer extended computational power for specific kind of algorithms such as modal resonators used in many hybrid instruments. They also provide unparalleled audio latency performances (as low as $100\mu s$) which would be extremely useful to design hybrid instruments involving active control.

On the software side, we want to improve the support of FAUST for linear algebra to implement complex physical modeling algorithms (e.g., Finite Difference Schemes, etc.).

## 7. CONCLUSION

By supporting an extended range of platforms and by providing dozens of DSP libraries, FAUST is well suited to design hybrid acoustical and digital musical instruments. We plan to keep expanding the scope of FAUST in the near future by supporting new platforms and by adding new algorithms to its libraries.

[10] `https://ccrma.stanford.edu/courses/250a-winter-2019/projects/`

[11] `https://faust.grame.fr/syfala/`

## 8. REFERENCES


[1] R. Michon, "The hybrid mobile instrument: Recoupling the haptic, the physical, and the virtual," 2018.

[2] J.-M. Adrien, "The missing link: Modal synthesis," in *Representations of Musical Signals*, ch. The Missing Link: Modal Synthesis, pp. 269–298, Cambridge, USA: MIT Press, 1991.

[3] J. O. Smith, *Physical Audio Signal Processing for Virtual Musical Instruments and Digital Audio Effects*. W3K Publishing, 2010.

[4] D. Schlessinger and J. O. Smith, "The Kalichord: A physically modeled electro-acoustic plucked string instrument," in *Proceedings of the 9th International Conference on New Interfaces for Musical Expression (NIME-09)*, (Carnegie Mellon Univeristy, USA), June 2009.

[5] A. Momeni, "Caress: An enactive electro-acoustic percussive instrument for caressing sound," in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME-15)*, (Baton Rouge, USA), 2015.

[6] P. Brinkmann, P. Kirn, R. Lawler, C. McCormick, M. Roth, and H.-C. Steiner, "Embedding PureData with libpd," in *Proceedings of the Pure Data Convention*, (Weinmar, Germany), 2011.

[7] V. Lazzarini, S. Yi, J. Timoney, D. Keller, and M. Pimenta, "The mobile Csound platform," in *Proceedings of the International Conference on Computer Music (ICMC-12)*, (Ljubljana, Slovenia), 2012.

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

[9] R. Michon, J. Smith, and Y. Orlarey, "New signal processing libraries for Faust," in *Proceedings of the Linux Audio Conference (LAC-17)*, (Saint-Etienne, France), 2017.

[10] R. Michon, J. O. Smith, C. Chafe, G. Wang, and M. Wright, "The faust physical modeling library: a modular playground for the digital luthier," in *Proceedings of the 1st International Faust Conference (IFC-18)*, (Mainz (Germany)), 2018.

[11] R. Michon, J. O. Smith, M. Wright, C. Chafe, J. Granzow, and G. Wang, "Mobile music, sensors, physical modeling, and digital fabrication: Articulating the augmented mobile instrument," *Applied Sciences*, vol. 7, no. 12, p. 1311, 2017.

[12] R. Michon, "faust2android: a Faust architecture for Android," in *Proceedings of the 16th International Conference on Digital Audio Effects (DAFx-13)*, (Maynooth, Ireland), 2013.

[13] R. Michon, J. Smith, C. Chafe, G. Wang, and M. Wright, "faust2smartkeyb: a tool to make mobile instruments focusing on skills transfer in the Faust programming language," in *Proceedins of the International Faust Conference (IFC-18)*, (Mainz, Germany), July 2018.

[14] A. McPherson, "Bela: An embedded platform for low-latency feedback control of sound," *The Journal of the Acoustical Society of America*, vol. 141, no. 5, pp. 3618–3618, 2017.

[15] R. Michon, Y. Orlarey, S. Letz, and D. Fober, "Real time audio digital signal processing with Faust and the Teensy," in *Proceedings of the Sound and Music Computing Conference (SMC-19)*, (Malaga, Spain), 2019.

[16] R. Michon, D. Overholt, S. Letz, Y. Orlarey, D. Fober, and C. Dumitrascu, "A Faust architecture for the esp32 microcontroller," in *Submitted to the Sound and Music Computing Conference (SMC-20)*, (Turin, Italy), 2020.

[17] R. Michon, J. O. Smith, M. Wright, and C. Chafe, "Augmenting the iPad: the BladeAxe," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, (Brisbane, Australia), 2016.