STANDALONE INTERACTIVE AND GENERATIVE MUSIC WITH THE CSOUND-FPGA FRAMEWORK

AMAN JAGWANI[*1] · VICTOR LAZZARINI [1]

AMAN.JAGWANI.2023@MUMAIL.IE VICTOR.LAZZARINI@MU.IE

ABSTRACT

This paper presents and discusses approaches to interactive and generative music composition and deployment on Field Programmable Gate Array (FPGA)-based System on Chips (SoCs) with the Csound-FPGA framework. With the development of Csound 7, Csound is now able to target bare-metal embedded systems, opening up the possibility of running Csound on SoCs that contain both programmable hardware and processing systems. This provides a unique platform for deploying standalone generative musical systems, leveraging the flexibility and computational power of the FPGA alongside the portability of an established domain-specific language like Csound on the CPU. With careful planning and division of tasks between the Programmable Logic (PL) and the Processing System (PS), comprehensive systems and compositions can be depoloyed, in contrast with other embedded systems like microcontrollers that may be more constrained. This paper presents an overview of the framework along with an examination of how it can contribute towards ubimus interactive

1. Department of Music, Maynooth University - Maynooth, Ireland.

music practices. Following that, a complete generative and interactive musical system is presented as an example study, highlighting possible methodologies for deploying this framework. Lastly future possibilities, limitations and conclusions are discussed.

1 INTRODUCTION

With Csound 7, the ability to target bare-metal embedded platforms has been introduced [Lazzarini and Jagwani, 2025]. One such platform capable of running Csound is the field programmable gate array (FPGA)-based System-on-Chip (SoC). These chips typically consist of FPGA fabric, or programmable logic (PL), integrated with an ARM CPU or processing system (PS) that can run bare-metal software. The Csound-FPGA framework involves running bare-metal Csound on the PS, along with custom audio processing hardware modules on the PL. The framework manages all low-level detailes and handles communication between the PS and PL.

1. Department of Music, Maynooth University - Maynooth, Ireland.

This paper presents an overview of the Csound-FPGA framework and explores the opportunities it offers for generative and interactive music design within ubimus contexts. Generative music systems are typically self-contained, adaptive, and algorithmically driven, capable of evolving autonomously over time [Gradim and Pestana, 2021]. Randomness, probability, stochastic systems and flexible event scheduling are often key elements in shaping their behavior. When applied to a diverse palette of timbral and textural sound sources, these strategies can result in a complete generative music system.

While such systems can be built using sample-based or fixed media sources, digital signal processing (DSP) offers additional possibilities by enabling the dynamic shaping and synthesis of sound. In this way, DSP can serve as a powerful tool in generative music design. Given the availability of both generative and DSP features, domain-specific, high-level music programming environments such as Csound [Lazzarini et al., 2016] are particularly well suited to support these approaches.

Generative music also implies the fact that while musi-cians, artists or composers define musical parameters or instructions such as note choice, sequencing rates and timbral possibilities, they are not part of the actual musical or performance process, with the generative system independentaly realising the musical outcome at the time of performance. This independence of performance can suggest that standalone or embedded deployments may be particularly well-suited to support the self-evolving, self-

*Aman Jagwani would like to acknowledge the support of the Hume Scholarship Programme, Maynooth University.

contained nature of such systems.

[Indent] However, due to the complex logic, sequencing structures, and signal processing often required to sustain musical and timbral richness, generative systems are commonly deployed on desktop platforms, where computational resources are abundant. This can be seen in works such as [Mangwani, 2024]. Yet, this reliance on desktop environments can limit portability. Embedded platforms, while more compact and deployable, often lack the computational resources or DSP capabilities required for such tasks. For instance, ubimus toolkits based on microcontrollers such as the Interactive Audio Toolkit (IAT) [Jagwani and Lazzarini, 2025] offer flexibility and musical interfaces for sensor-based interactions and mechanical sequencing, but may be limited in audio processing capacity.

[Indent] FPGAs can address this gap by offering a reconfigurable embedded platform that supports both DSP and generative audio. As field-programmable devices, FPGAs align well with the concept of aesthetic pliability [Keller et al., 2023], allowing hardware configurations to be shaped and reshaped for specific artistic needs. Moreover, this adaptability connects with the idea of the Internet of Musical Stuff [Messina et al., 2024], where"musical stuff" is defined as morphable and context-driven. With support for both wired and wireless networking, as well as the ability to run web-servers on FPGA SoCs as demonstrated in [Murugan et al., 2016], FPGAs are well positioned to function as part of this environment.

[Indent] Being embedded also allows FPGAs to create standalone systems that integrate directly with a wide range of sensors and peripherals, enabling interactivity to

complement generative music. This opens new possibilities for audience engagement. For example, an FPGA could autonomously generate an evolving soundscape in an installation, while real-time data from LiDAR or ultra-sonic sensors modulate aspects of the audio, such as spectral brightness, spatialisation, or dynamics, based on audience movement. This kind of interaction allows for meaningful participation by non-expert audiences, aligning with the ubimus concept of inclusivity [Keller et al., 2019b].

[Undefined] Such systems offer a promising platform for inclusive interaction design, accommodating both passive and active engagement. They can support the kind of embodied participation seen in the Chaal installation discussed in [Jagwani and Lazzarini, 2025], and can extend the ideas explored in the Handy Metaphor [Keller et al., 2019a], where touchless, sensor-based musical interactions were facilitated using technologies like ultrasonic and vision-based systems. FPGAs, with their ability to handle sensor data and perform tasks such as video-based computer vision [B.K. et al., 2017], can extend these approaches.

[Undefined] One common barrier associated with FPGAs is their relative inaccessibility, particularly due to the complexity of hardware design and low-level programming [Jagwani, 2023]. However, the Csound-FPGA framework abstracts these low-level details and introduces Csound as a high-level interface for working with the FPGA platform. This opens up the technology to a broader community of musicians and composers who may not have experience with hardware design but are comfortable with musical programming environments, once again highlighting the idea of inclusivity but in the realm of the composer or artist in this case instead

of the audience.

With this improved accessibility, the framework can ali-gn with the second wave of ubimus, particularly with the ideas of do-it-yourself (DIY) practices [Keller et al., 2024]. Timoney et al. [Timoney et al., 2020] outlined the evolution of DIY ubimus practices, examining microcontroller-based DIY ubimus platforms like Arduino. The Csound-FPGA framework can build on this foundation by offering a more powerful, yet still accessible, platform for DIY exploration, particularly due to its integration with a familiar music programming language.

Additionally, [Brown and Ferguson, 2024] discussed di-gital fabrication as a continuation of DIY ubimus practices, following the transition from non-programmable to programmable electronics. Similarly, the move from general-purpose CPUs to custom, programmable hardware via FPGAs represents a further step in this progression. This transition is only possible, however, if the tools involved provide a high enough level of access, something the Csound-FPGA framework facilitates. Moreover, while digital fabrication enables the creation of custom circuit boards, this framework mirrors this idea with the designing of custom hardware circuits directly in the FPGA's programmable logic (PL), effectively bringing a new layer of fabrication to DIY musical practices.

As highlighted in [Keller et al., 2025], "possibly one of the most complex challenges faced by ubimus frameworks is the demand for supporting both legacy practices and prospective exploratory initiatives." The Csound-FPGA framework is particularly well suited to address this challenge. Csound, with its history as a widely used FLOSS

environment, brings strong connections with legacy musical practices while maintaining an active and evolving user base. At the same time, FPGAs, with their reconfigurable, morphable hardware, embody the spirit of exploratory design, offering new possibilities for signal processing, interaction, and generative systems. Thus, this framework can align well with the ubimus concepts of replicability and aesthic pliability while being conducive to innovation and exploration in a cutting-edge technological platform.

The subsequent sections provide background on the use of FPGAs in audio and music applications. This is followed by an overview of the Csound-FPGA framework, including its mechanisms, methodologies, and how it supports common generative music practices within Csound. The paper concludes with a complete example application demonstrating generative and interactive music design using the framework, along with a discussion of potential future developments.

2 FPGAS IN AUDIO AND MUSIC

FPGAs are integrated circuits with programmable logical units that can be configured into custom hardware through low-level hardware design techniques. Along with their inherent flexibility and reconfigurability, in an audio processing context, FPGAs provide the benefits of ultra-low latency, high-throughput, high sampling rates and comprehensive connectivity through a large number of general-purpose inputs and outputs (GPIOs) [Jagwani, 2023]. These features make them well-suited to musical tasks.

Commonly, FPGAs are packaged as a

system-on-chip with an associated CPU (PS). A C or C++ application generally runs on the PS while custom hardware runs on the FPGA fabric or PL. This system architecture provides the opportunity for efficient division of tasks across both parts of the SoC through hardware-software co-processing. The hardware used for experimentation and discussion in this paper was the Xilinx Zynq 7000 SoC, running on a Digilent Zybo Z7020 development board [Digilent, 2023].

[Indent] Generally, FPGAs are complex to program. Low-level hardware design languages such as Verilog and VHDL are most commonly used to program them. These may be out of reach for the general music and audio community, making FPAGA usage and adoption for DIY ubimus practices and instrument design limited.

[Indent] However, recent explorations of higher-level methodo-logies and frameworks for FPGA programming have provided wider access to the platform. An early exploration of the use of high-level synthesis (HLS) for FPGA-based audio synthesis was seen in [Fitzgerald, 2019], highlighting how HLS can enable hardware designs to be generated from C or C++ programs. [Popoff et al., 2023] presented the Syfala toolchain which enables complete FPGA applications to be designed with the FAUST [GRAME, Centre National de Cr′eation Musicale, 2025] language. [Keller et al., 2025] presented a ubimus plugging framework in the form of ModFPGA, highlighting a system for modular audio synthesis and processing on FPGAs.

[Indent] All of these studies present comparatively user-friendly forms of FPGA based audio programming but their

[FormattingFormatting]

focus was generally on the technical aspects of running audio processing programs. Musical and compositional approaches are not extensively discussed. Additionally, these studies show FPGA audio programs being generated from higher-level sources but not actually running a complete higher-level domain-specific language like Csound itself on an FPGA SoC platforms.FPGA SoC platforms.

[Indent]These gaps can be addressed with the Csound-FPGA framework, particularly in the context of standalone generative and interactive music design.

3 THE CSOUND-FPGA FRAMEWORK

The FPGA SoC architecture necessitates a careful division of tasks between the PS and PL, with control-rate operations typically assigned to the PS and audio-rate processing handled by the PL, as demonstrated in [Keller et al., 2025, Popoff et al., 2023].

[Indent]With the development of Csound 7 and the emergence of its co-processing capabilities, enabled by bare-metal csound [Lazzarini and Jagwani, 2025], it is now possible to run Csound on the PS while delegating audio processing operations to custom intellectual property (IP) cores or modules in the PL. [Keller et al., 2025] presents the methodologies and strategies for deisgning these IP cores with HLS in detail. These can include custom hardware modules or hardware ports of Csound opcodes such as reverbsc [Lazzarini and Jagwani, 2025]. The overall system architecture is summarised below:

[FormattingFormatting]

- **PS-based processing:** The PS runs bare-metal Csound, handling scheduling, sequencing, modulation, and control tasks, as well as potentially performing some synthesis and audio processing.

- **PL-based processing:** High-throughput, low-latency audio-rate processing is performed on the PL. These operations take place on a sample-by-sample basis and can be run at very high sampling rates, enabling the exploration of techniques such as higher-order FM synthesis [Lazzarini and Timoney, 2024] or virtual analog synthesis.

- **PS-PL communication:** The PS and PL communicate in different ways depending on the type of data they are sharing. For control data, the AXILITE protocol [AMD, 2024] is used and for the streaming audio data, a Direct Memory Access (DMA) IP/hardware module is used. More details for the communication methodologies are presented in [Lazzarini and Jagwani, 2025].

- **MIDI input:** MIDI can be received via USB or through a UART peripheral using one of the Zybo board's assignable IO pins.

- **Sensor input:** Analog sensors are connected to the system via the XADC Wizard IP core [AMD, 2012] in the PL. The data from this IP core can be accessed from the PS with the XADC driver and is transferred to Csound through its sofware bus [The Csound Developers, 2025] or control channels.

- **Additional connectivity:** The Zybo board also facilitates additional connectivity; a networked system can be integrated with the help of the ethernet port or a visual system can be integrated with the help of the HDMI port on the board, for example.

- **System Extension:** The Zynq chip contains a large amount of flexible, assiginable general-purpose inputand output (GPIO) pins, which can be assigned a range of protocols such as UART, I2S or SPI, enabling flexible communication with other devices, breakout boards or even other microcontrollers. For example an ESP32 board running the Interactive Audio Toolkit (IAT) described in [Jagwani and Lazzarini, 2025] can be connected to a Zybo board running the Csound-FPGA framework. This can allow the ESP32 to handle sensor-based interactions, WI-FI connectivity and mechanical output sequencing while the Zybo handles generative audio and DSP tasks. Combined with the DIY digital fabrication process presented in [Brown and Ferguson, 2024], a custom PCB for a standalone extended hybrid interactive sound installation system can be created.

[Undent] Thus, the Csound-FPGA Framework, takes care of task delegation between the PS and PL, and handles communication protocols and interactivity. With these lower level details abstracted, the user can simply interact with the framwork like any other Csound project. For instance, the Csound main engine input and output buffers, spin and spout are automatically connected to a DMA IP that manages audio data communication with the on-board

[FormattingFormatting]

audio CODEC. Users can simply call the ins, outs opcodes for audio IO. For using a hardware processing module on the PL, users can simply use the plug-in opcodes, fpgaOut and fpgaIn for sending and receiving audio data as explained in [Lazzarini and Jagwani, 2025]. Internal connections and DMA transfers are once again, taken care of by the framework. The system architechture is illustrated in figure 1.

4 GENERATIVE MUSIC WITH CSOUND ON FPGAS

Csound presents vast opportunities for generative music design through flexible sequencing capabilities with opcodes such as sequ, random number and probabilistic generators such as rspline, jspline, random, randi, randh as well as flexibile scheduling with opcodes like schedkwhen. The code excerpt below shows how some of these tools can be utilised to create a simple generative melodic system. Instrument 1 is the main clocking system that triggers Instrument 2, which is the melodic instrument based on FM synthesis, at continuously evolving rates. When triggered, Instrument 2 chooses a random note from gkseq, which is an array of midi note numbers.

```
gkseq[ ] fillarray 71, 67, 79, 74, 71, 74, 79, 76, 83, 76

instr 1 // generative clocking system
krandcps rspline 0.1, 4, 0.1, 0.5
gkrandtrig metro krandcps
kranddiv trandom gkrandtrig, 1, 6
schedkwhen gkrandtrig, 0, 1, 2, 0, 0.1 + rnd(0.3)
endin
```

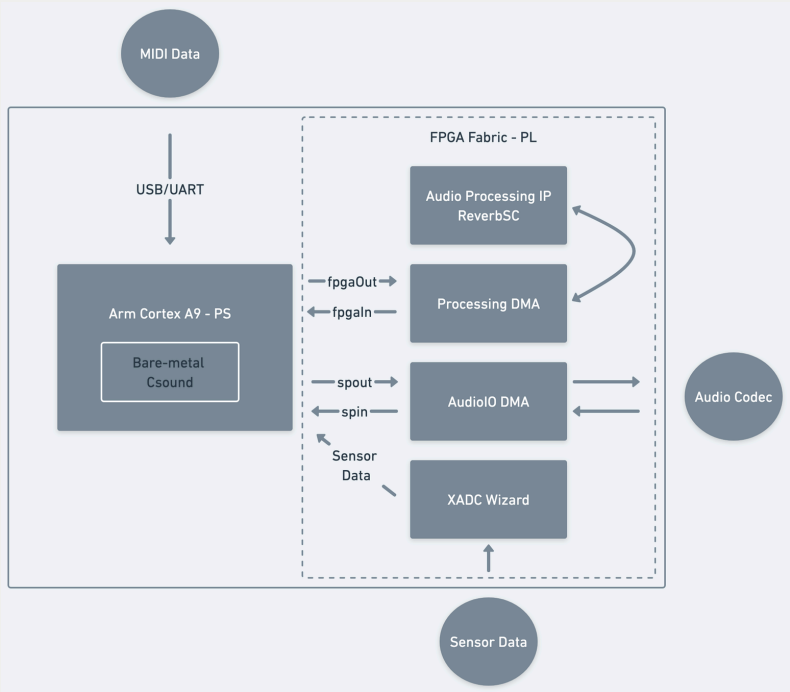

FIGURE 1: CSOUND-FPGA FRAMEWORK ARCHITECTURE

```
instr 2 // FM mallet-style instrument
index random 0, 10
index = int(index)
imodindx = rnd(1)
knote = gkseq[index]
iamp = (0.7 + rnd(0.3))
idur = p3
knote = cpsmidinn(knote-24)
aindx expseg 30, idur/2, 0.00001
aenv expsegr 0.001, 0.01, 0.1, idur, 0.01, idur/4, 0.001
amod1 oscili (aindx*imodindx)*(knote*4), knote*3
acar1 oscili 1, knote+amod1
aout = (acar1)*aenv*iamp
outs aout, aout
endin

schedule(1, 0, 1000)
```

[Indent] An installation that utilises Csound for its generative so-und design can be seen in [Mangwani, 2024]. In this installation, all sounds are generated in real-time with Csound running on a desktop computer. Additionally, heart rate and breath sensors are used to create modulations in sound and lights in coherence with a participant's physiological responses. This installation illustrates the power of combining generative techniques with interactive systems. The generative piece is able to evolve autonomously while the breath and heart-rate provide a level of grounding and connection with the audience, allowing them to participate in a seemingly passive but meaningful way. This also enables the extension of audience participation into passive, physiological domains, increasing inclusivity and allowing non-expert audiences to contribute to the musical process without requiring active technical engagement.

[Indent] However, a key limitation of this installation is its relian-ce on a desktop platform. In general, target platforms for generative music tend to be confined to Desktop and possibly web platforms. This may be due to the lack of resources in commonly available embedded platforms to run complex generative and audio processing tasks efficiently. Furthermore, embedded programming often requires low-level C or C++ development, which may be limiting for composers. While platforms like Daisy [Electro-Smith, 2023] and Arduino [Ard,] do simplify some aspects, they are limited in providing high-level, expressive generative capabilities. Another limitation of this installation is the reliance on proprietary, commercial devices such as the Fit-bit smart watches [Google LLC, 2025] for heart-rate and breath sensing. These bring additional costs and limitations

of vendorsupplied APIs for accessing data.

[Indent] The Csound-FPGA framework can provide a unique and suitable solution for this. The availability of a large amount of logical resources in the PL along with the ability to divide tasks between the PS and the PL, means that complex generative programs can easily be run in an embedded, portable context. The embedded FPGA SoC format also allows on-board interfacing with sensors, removing the need for any proprietary devices, reducing costs and providing flexibility in real-time data access. For example, the Analog Devices MAX30102 [Analog Devices, 2016] can replace the Fit-bit in the above installation. At the same time, the familiarity and all of the generative abilities of Csound are preserved, allowing composition and sound design at a higher, more comfortable level for artists. Additionally, the portability of Csound programs, ideas and sketches across platforms also means that the FPGA-based SoC can become a platform to explore and deploy previous Csound developments such as the Organic Generative Structures presented in [Heintz, 2024].

5 EXAMPLE

This section presents an example of a generative and interactive music piece with the Csound-FPGA Framework.

[Indent] In this example, sequencing, synthesis, and audio genera-tion are all handled by Csound running on the PS, while audio processing, specifically with the hardware port of the reverbsc opcode, is handled by the PL. This approach reduces the computational load on the PS by delegating a complex feedback delay network to the FPGA, which

[FormattingFormatting]

processes it on a sample-by-sample basis with ultra-low latency. As a result of this offloading, the system can operate with a lower ksmps value in Csound, minimising latency. Retaining synthesis on the PS also enables the use of Csound's unique opcodes, such as gbuzz, which is featured in this example. Two sensors are used for adding modulations, a Li-DAR sensor along with a light dependent resistor (LDR). Both of these add motion-based interaction to the system with the LiDAR sensor reacting to long-range motion and the LDR reacting to close-range motion.

The code below contains the csound program used in this example, highlighting the use of the software bus for sensor data and FPGA plugin opcodes for PL processing:

```
gkseq1[] fillarray 71, 67, 74, 79, 74, 71, 74, 79
gkseq2[] fillarray 76, 71, 79, 83, 79, 76, 79, 83
gisine ftgen 0,0,4096,10, 1

instr 1 // sequencer
ktempo chnget "lidar" // LiDAR sensor modulating tempo
gkharms chnget "ldr" // ldr sensor modulating harmonic content
gkbuzz metro ktempo
gkpad metro 0.125*ktempo*0.5
gkdur = 1/(ktempo)
kadditions[] fillarray -12, 0, 12, 0, -12, 0, 12, -12
kindx trandom gkbuzz, 0, 8
kindx = int(kindx)
knote = gkseq1[kindx]
knote2 = gkseq2[kindx]
```

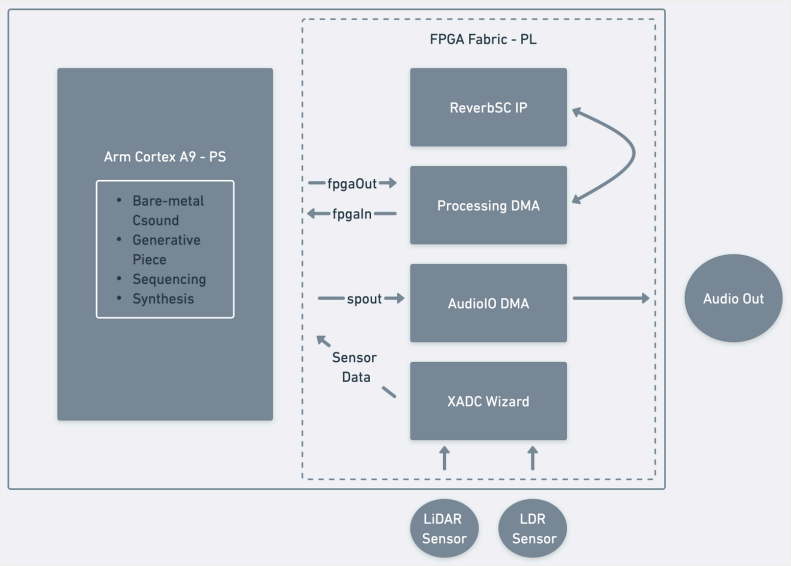

FIGURE 2: EXAMPLE GENERATIVE AND INTERACTIVE SYSTEM DESIGN WITH THE CSOUND-FPGA FRAMEWORK

```
kadd = kadditions[kindx]
schedkwhen gkbuzz, 0, 8, 11, 0, gkdur*0.5, knote, 0.3, 0
schedkwhen gkpad, 0, 8, 11, 0, 0.25*16, knote2+kadd-12, 0.3, 1
endin

instr 2
aenv linsegr 0, p3*0.5, 0.4, p3*0.25, 0.6, p3*0.25, 0.4, p3*00.25, 0
kdetune1 rspline 0.01, 3, 0.1, 3
kdetune2 rspline 0.01, 3, 0.1, 3
kmul rspline 0.1, 0.99, 0.01, 5
kamp rspline 0.1, 0.4, 0.01, 0.1
a1 gbuzz aenv*kamp*(0.4+(0.4)), cpsmidinn(p4-12)+kdetune1, 50, 1,
gkharms *
kmul, gisine
a2 gbuzz aenv*kamp*(0.4+(0.4)), cpsmidinn(p4-12)+kdetune2, 50, 1,
gkharms *
kmul, gisine
kpan rspline 0, 1, 0.1, 2
al, ar pan2 a1 + a2, kpan
```

```
// FPGA opcodes used to send audio to PL for processing
fpgaOut al, ar
afpgaL, afpgaR fpgain
// outs connected to audio output from Zybo board
outs (afpgaL)*p5, (afpgaR)*p5
endin

schedule(1, 0, 10000)
```

This example can be heard at the link below:
https://youtu.be/sEWzhKcb3wc

[Indent] The system design and delegation of tasks is highlighted in figure 2.

[Indent] While this example is effective sonically and musically, the hardware-software configuration can also be modified to have Csound function primarily as a control engine, handling sequencing and parametric control on the PS with all audio synthesis and signal processing tasks implemented in the PL, using modular IP blocks from the ModFPGA ubimus plugging framework [Keller et al., 2025]. This demonstrates how the Csound-FPGA framework can integrate with another ubimus platform seamlessly for system design. With all of the audio processing in the PL, this alternate configuration could further leverage the FPGA's capabilities of low-latency as well as its computational resources.

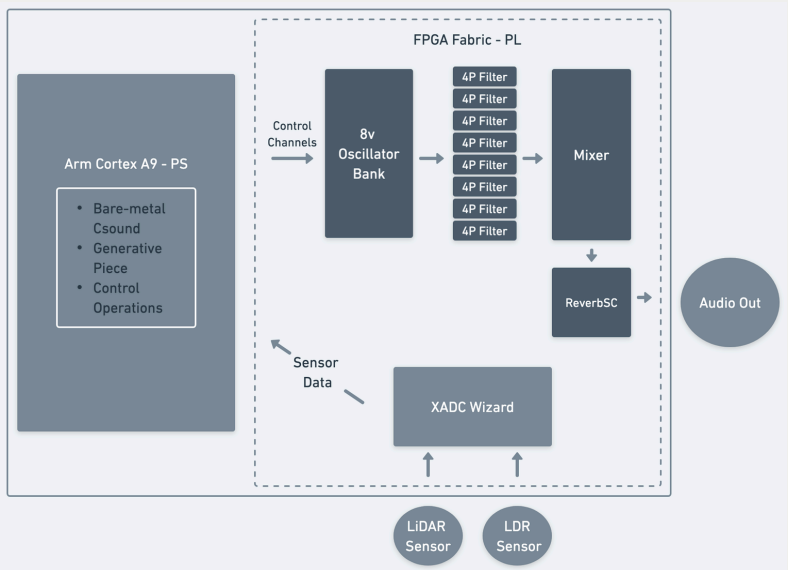

FIGURE 3: ALTERNATIVE GENERATIVE AND INTERACTIVE SYSTEM DESIGN WITH THE CSOUND-FPGA FRAMEWORK INTEGRATING WITH MODFPGA IP CORES

For example, a synthesizer with eight polyphonic voices with two oscillators and a fourpole filter each, along with the reverbsc module can comfortably run on the PL on a sample-by-sample basis and Csound can generatively control this synthesizer from the PS.

This alternate system design can be seen in figure 3.

6 FUTURE DEVELOPMENTS AND CONCLUSIONS

The Csound-FPGA framework can present a compelling platform for interactive ubimus practices, combining the strengths of both Csound and FPGAs. While FPGAs are inherently flexible, the integration of a complete and mature library like Csound makes artistic deployment even more pliable through custom Csound code. This is further

supported by the ability to leverage different hardware–software co-processing configurations within the same system, as demonstrated in the example discussed.

Importantly, Csound itself is not a new tool, but a well-established and familiar environment now extended to a new platform. This continuity can help lower the learning curve and reduce barriers to entry, enabling the portability of existing Csound projects to this relatively new embedded ubimus platform with greater ease.

Furthermore, the combination of ease of use, computa-tional power, and the inherently interactive ability of embedded systems offers a unique space for generative DSPbased composition that also supports real-time interaction. This opens up new possibilities for engaging non-expert audiences in the music-making process, allowing them to participate meaningfully through intuitive, self-adapting and responsive systems that do not require technical engagement.

Future work would include the development of additio-nal Csound opcode ports as hardware IP modules. In particular, FPGA parallelism can be leveraged for computationally intensive techniques such as spectral processing or time-varying convolution. Another key development would be the automation of the build process, enabling users to generate complete FPGA applications directly from .csd files. While the current framework abstracts many low-level details, it still involves a considerable setup process, particularly with respect to the Xilinx tool chain. With these improvements, the framework can evolve into a more comprehensive system for DSP, interactivity, and generative audio design, offering a more a

more accessible entry point for musicians and artists.

The Csound-FPGA framework, along with the example discussed in this paper, will be available in the Csound github repository:

https://github.com/csound/csound

REFERENCES

Arduino. https://www.arduino.cc/. Accessed: 2024-02-16.

AMD (2012). LogiCORE IP XADC Wizard v2.2 User Guide (UG772). AMD Adaptive Computing. Version 2.2.

AMD (2024). MIPI D-PHY LogiCORE IP Product Guide (PG202): AXI4-Lite Interface. AMD Adaptive Computing. Version 4.3.

Analog Devices (2016). MAX30102: High-Sensitivity Pulse Oximeter and Heart-Rate Sensor. Analog Devices. Accessed: 2025-05-10.

B.K., A., Venkatraman, V., Kumar, A. R., and S., S. D. (2017). Accelerating real-time computer vision applications using hw/sw co-design. In 2017 International Conference on Computer, Communications and Electronics (Comptelix), pages 458–463.

Brown, A. R. and Ferguson, J. (2024). Diy musical instruments: From handmade electronic circuits to microcontrollers and digital fabrication. Journal of Ubiquitous Music, 1(1):8–22.

Digilent (2023). Zybo z7 reference manual. https://digilent.com/reference/programmable-logic/zybo-z7/start. Accessed:2023-08-24.

Electro-Smith (2023). Daisy. https://www.electro-smith.com/daisy. Accessed: 2023-06-08.

Fitzgerald, L. (2019). Sound synthesis using programmable system-on

chip devices. Master's thesis, National University of Ireland, Maynooth, Co. Kildare, Ireland.

Google LLC (2025). Google store watches. Accessed: 2025-05-10.

Gradim, R. and Pestana, P. D. (2021). Overview of generative processes in the work of brian eno. In Proceedings of the 11th Workshop on Ubiquitous Music (UbiMus 2021), pages 45–56, Matosinhos, Portugal. HAL ID: ffhal-03398725f.

GRAME, Centre National de Cr´eation Musicale (2025). Faust: Functional audio stream. Accessed: 2025-05-14.

Heintz, J. (2024). Creating organic generative structures in csound. In Stojak, S. and Hofmann, A., editors, Proceedings of the 7th International Csound Conference, Hannover, Germany. ICSC. Accessed: 2025-05-14.

Jagwani, A. and Lazzarini, V. (2025). Interactive audio toolkit: Creating sonic experiences and installations with low-cost, low-power microcontrollers. In Estadieu, G. V., Messina, M., Gomez Meja, C. M., Keller, D., Kramann, G., and Koszolko, M., editors, Proceedings of the Ubiquitous Music Symposium 2024 (UbiMus 2024), pages 1–132, Macau. Ubiquitous Music Symposium.

Jagwani, A. P. (2023). Developing a modular sound synthesis platform for fpgas with high level synthesis techniques. Master's thesis, Maynooth University, Department of Music.

Keller, D., Gomes, C., and Aliel, L. (2019a). The handy metaphor: Bimanual, touchless interaction for the internet of musical things. Journal of New Music Research, 48:1–12.

Keller, D., Jagwani, A., and Lazzarini, V. (2025). The ubimus plugging framework: Deploying fpga-based prototypes for ubiquitous music hardware design. Computers, 14(4):155.

Keller, D., Lazzarini, V., Turchet, L., and Brooks, A. L. (2024). Ubimus [Formatting Formatting]

contributions to digital creative practices (editorial). Digital Creativity, 35(1):1–12.

Keller, D., Messina, M., Bridges, B., and Yaseen, A. (2023). Editorial: Ubiquitous music as a movement. In Proceedings of the Ubiquitous Music Symposium 2023 (Ubimus 2023), Derry, Ireland. Ubiquitous Music Group. HAL Id: hal-04372736.

Keller, D., Schiavoni, F., and Lazzarini, V. (2019b). Ubiquitous music: Perspectives and challenges. Journal of New Music Research, 48:1–7.

Lazzarini, V., Ffitch, J., Yi, S., Heintz, J., Brandtsegg, O., and McCurdy, I. (2016). Csound: a sound and music computing system. Springer.

Lazzarini, V. and Jagwani, A. (2025). CSOUND 7: Bare Metal and Co-Processing Hardware. In Proceedings of the 19th Linux Audio Conference (LAC-25), Villeurbanne, France.

Lazzarini, V. and Timoney, J. (2024). Theory and practice of higher-order frequency modulation synthesis. Journal of New Music Research, 0(0):1–16.

Mangwani, H. (2024). Effection: A biofeedback-based interactive art installation. https://www.hansikart.com/biofeedback-art. Accessed: May 10, 2025.

Messina, M., de Souza Stolfi, A., Aliel, L., Simurra, I., and Keller, D. (2024). The internet of musical stuff: Towards an aesthetically pliable musical internet. International Journal of Software Innovation (IJSI), 12(1):1–19.

Murugan, V. D., Saravanan, C., and Imtiaz, R. (2016). Fpga based standalone embedded web server for remote control of display devices. International Journal of Microelectronics and Embedded Systems, 4(2):224–228.

Popoff, M., Michon, R., Risset, T., Cochard, P., Letz, S., Orlarey, Y., and de Dinechin, F. (2023). Audio DSP to FPGA Compilation: The Syfala Toolchain Approach. Technical Report RR-9507, Univ Lyon, INSA Lyon, Inria, CITI, Grame, Emeraude.

The Csound Developers (2025). Csound Manual: Software Bus. Csound Project. Accessed: 2025-05-10.

Timoney, J., Lazzarini, V., and Keller, D. (2020). Diy electronics for ubiquitous music ecosystems. In Lazzarini, V., Keller, D., Otero, N., and Turchet, L., editors, Ubiquitous Music Ecologies, chapter 3. Routledge.