

An Accessible Open Source Embedded Platform for Spatial Audio Rendering

Clemens Wegener
CITI Lab, Emeraude Group
INRIA/INSA
Villeurbanne, France
clemens.wegener@inria.fr

Romain Michon
CITI Lab, Emeraude Group
INRIA/INSA
Villeurbanne, France
romain.michon@inria.fr

Pierre Lecomte
LMFA, UMR5509
École Centrale de Lyon, CNRS,
Université Claude Bernard Lyon 1,
INSA Lyon,
69130 Écully, France
pierre.lecomte@ec-lyon.fr

Tanguy Risset
CITI Lab, Emeraude Group
INSA
Villeurbanne, France
tanguy.risset@insa-lyon.fr

Abstract—We present open source hardware and embedded software for spatial audio rendering, built around a Teensy-based compute node. The node decodes ambisonics and Wave Field Synthesis (WFS) signals, driving up to 32 output channels and is extensible to further methods such as beamforming. Audio can be received over USB or Ethernet, or read directly from an SD card, allowing the platform to operate as a standalone renderer or as a building block for larger installations. The system is programmed through an accessible, open source toolchain: FAUST for spatial audio rendering and Teensyduino for embedded programming, which lowers the entry barrier for less experienced programmers. We tested our compute node in a master-satellite configuration across two PCBs, in an anechoic chamber, where we recorded several spatial impulse responses of an 11-loudspeaker linear array reproducing a selection of point sources via a WFS algorithm. The recorded impulse responses of the WFS virtual sources were compared against both the corresponding acoustic simulations and a real sound source at the same position. The resulting spatial impulse responses closely match the reference source and the corresponding acoustic simulations, with only minor artefacts typical of WFS reproduction, proving the feasibility of using two DACs in synchronisation on the Teensy platform.

Index Terms—Spatial Audio, Ambisonics, Wave Field Synthesis, Embedded Audio Hardware, Open Source Hardware, Beamforming

I. INTRODUCTION

Spatial audio rendering is finding an increasingly wide range of applications, from cinema and concert halls to virtual reality platforms and interactive installations [1]. Techniques such as higher-order ambisonics (HOA) [2], Wave Field Synthesis (WFS) [3], and beamforming [4] can synthesise convincing three-dimensional soundscapes, but typically require driving tens to hundreds of loudspeaker channels. The computational and hardware demands of such systems have historically confined them to specialised venues with costly, purpose-built infrastructure, often built around commercial audio-transport standards such as MADI (AES10) and Dante. MADI distributes its clock physically over dedicated point-to-point links, while Dante achieves comparable sub-microsecond synchronisation over standard Ethernet via the Precision Time Protocol (PTP), but as a closed, licensed protocol requires dedicated Dante hardware to participate. Neither exposes a programmable compute layer: audio transport and DSP run

on separate hardware, so synchronisation and rendering cannot be combined on a single affordable device. Even recent open-source rendering software such as *Seamless* [5] still relies on Dante or MADI for inter-machine transport and synchronisation.

A number of recent platforms have sought to lower this barrier. FPGA-based systems built around the Syfala compilation toolchain [6, 7, 8] centralise processing onto a single device and drive large channel counts at very low latency. Distributed alternatives instead synchronise multiple networked compute nodes: microcontroller architectures using the Teensy scale gracefully through Ethernet PTP [9, 10, 11], synchronised via the ANANAS protocol [12]. Ethernet AVB [13], an established standard, takes a similar approach but additionally requires every switch to support Stream Reservation Protocol (802.1Qat) and credit-based shaping (802.1Qav), beyond the PTP-capable switching ANANAS and Dante/AES67 need. GPU-based SoCs have also been distributed across compute nodes and synchronised via the coarser Network Time Protocol (NTP) [14], and GPS-disciplined gPTP extends synchronisation to geographically separated sites [15], though it requires outdoor GPS signal, ruling out indoor use. Despite this variety, few solutions combine fully open-source hardware with a programming model accessible to non-specialist users.

This paper presents an open-source embedded compute node for spatial audio rendering. The node is built around a Teensy microcontroller on a master PCB, paired with a satellite PCB carrying a second synchronised DAC, providing up to 32 output channels in total. It is an open platform currently supporting HOA decoding and WFS rendering. Concretely, the platform can operate as a 32-channel USB audio interface running any multichannel DSP algorithm on-board, or as a fully standalone spatial audio player reading content directly from an SD card, with no host computer required. When adding synchronised transport via ANANAS [12], it can also serve as a building block in a larger installation. The entire platform is programmed through open-source tools: FAUST [16, 17] for spatial audio DSP and Teensyduino for embedded programming, which substantially lowers the entry barrier compared to FPGA-based alternatives.

The platform also relates to the Internet of Sounds (IoS), which studies networked devices that acquire, process, exchange, or generate sound, and whose research agenda calls for embedded platforms specific to the IoS. Within the IoS, Networked Immersive Audio combines networking with immersive audio rendering [18]. Existing systems include server-based binaural rendering for remote audiences [19]. Loudspeaker-based reproduction additionally requires many output channels with sample-accurate timing at the endpoint, which is what the node presented here provides. This work validates a single node; the Teensy’s hardware PTP support and the ANANAS protocol [12] provide the path to synchronised multi-node operation, which we leave to future work.

Section II motivates the choice of the Teensy 4.1 as compute module. Section III describes the hardware architecture, including PCB design, DAC synchronisation, and audio I/O. Section IV details the FAUST-based software stack for spatial audio rendering. Sections V and VI present the experimental validation and setup, and Section VII reports the results. Section VIII concludes.

II. PLATFORM SELECTION

Prior work on cost-effective, high-channel-count spatial audio rendering has explored both FPGA architectures [7, 20, 6] and microcontroller-based systems [21, 9, 10]. Building on those findings, the design presented here opts for a microcontroller-based platform over a single centralised FPGA, with the explicit goal that the system can be extended in future work into a decentralised, networked spatial audio installation. The main drivers behind this decision were programming workflow, stability, cost, and the availability of Ethernet PTP in both hardware and software. PTP synchronises the audio clocks of multiple independent nodes over standard Ethernet, allowing nodes to be added to extend the total channel count beyond a single board’s limit (>512 channels) without distributing a fragile clock signal.

FPGA platforms, by contrast, lack low-cost or open-source PTP implementations. Without PTP, stable, sustainable, and economical channel synchronisation would rely on a self-implemented synchronisation scheme, which is beyond the scope of this work. The maximum channel count achievable from a single FPGA has been found to be around 512 [7], with the clock signal travelling through a stack of DAC boards connected to that FPGA. Synchronising multiple FPGAs through a shared clock remains unexplored and raises its own signal-integrity concerns at scale. Programming an FPGA is also more time-consuming than a Teensy: even with the Syfala compiler [6], compilation times limit rapid iteration, and exploiting FPGA compute power has proven difficult for some applications (e.g., memory-bandwidth bottlenecks in convolution). In practice, a networked microcontroller approach delivers comparable performance at lower cost while remaining easier to maintain.

Rushton et al. [11] evaluate five candidate platforms against the criteria of Ethernet connectivity, hardware PTP support, unit cost, finding the Teensy 4.1 to be the only sub-50€

board satisfying all three criteria. Additionally its 30-bit audio PLL provides sufficient resolution for PTP-derived sampling frequency correction without an external digitally controlled high resolution oscillator. They demonstrate that this correction reduces inter-node drift from several milliseconds to under $1\mu\text{s}$ over ten minutes, motivating the choice of the Teensy as the compute module for this work.

III. HARDWARE

The hardware architecture is shown in Fig. 1; a photograph of the assembled master PCB is shown in Fig. 2. The master PCB centres on a Teensy 4.1 microcontroller, which accepts audio via USB, Ethernet (RJ45 socket on the PCB; PHY integrated on the Teensy), or from the on-board SD card for standalone operation, and runs FAUST-compiled DSP, including the HOA and WFS renderers described in Section IV. The Teensy issues two independent TDM streams: TDM1 drives an ADAU1966A 16-channel DAC mounted on the master PCB; TDM2, together with I²C control and bus power, is routed to the on-board extension headers. A local power supply conditions the incoming DC and feeds both the Teensy and the DAC. The DAC outputs are brought out on two AES59 DB25 connectors carrying 16 balanced analog channels in total.

A satellite PCB uses the same layout as the master with the Teensy footprint left unpopulated. It receives TDM2, I²C, and power from the master’s extension headers and drives a second ADAU1966A, adding 16 further output channels on two further AES59 DB25 connectors. The satellite DAC’s I²C address is set via a solder jumper to avoid collision with the master. This two-board stack, yielding 32 channels in total, is the configuration validated in this work.

A. PCB Design

The master and satellite boards share an identical 111×111 mm PCB layout. The four-layer stackup places continuous ground planes on both inner layers and a ground flood on the outer layers to minimise loop impedance across the board. Bulk decoupling relies primarily on modern high-capacitance ceramic capacitors in small footprints, placed immediately adjacent to the ADAU1966A.

Power enters through a 2 mm barrel jack and is stepped down by a switching regulator. A separate low-noise LDO supplies the ADAU1966A analog domain. The ADAU1966A digital supplies are drawn from the Teensy’s onboard 3.3 V regulator, which also powers the MCU itself; ferrite beads are inserted to attenuate switching noise injected onto this shared rail by MCU activity. The RJ45 Ethernet connector is protected by transient-voltage suppressor diodes, routed as impedance-controlled differential pairs, and cable-shield isolation is managed by a solder-jumper-selectable Earth-to-ground connection with an optional RC filter.

The sixteen balanced audio outputs are routed as coupled differential pairs from the ADAU1966A through passive per-channel filters with a DC-blocking high-pass ($f_c \approx 0.32\text{ Hz}$) and a quantization noise post-filter ($f_c \approx 62\text{ kHz}$) to two

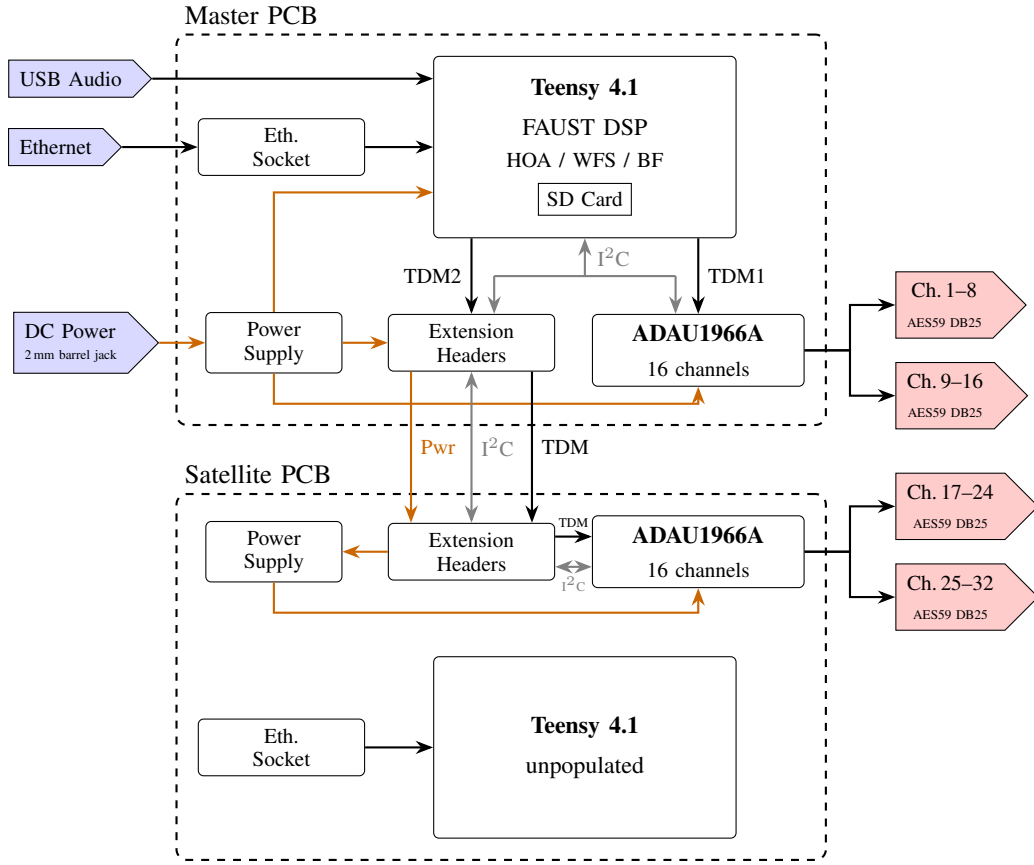


Fig. 1: Hardware architecture of the compute node, showing the master PCB and a satellite PCB.

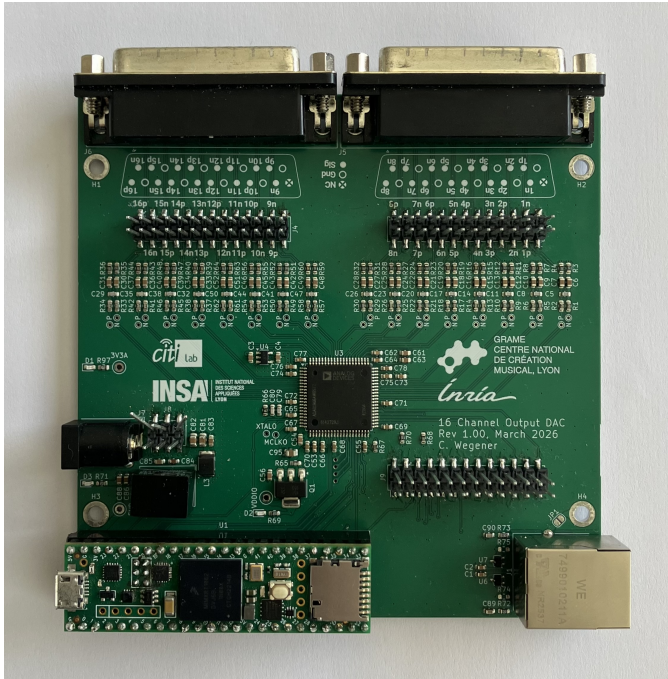


Fig. 2: Assembled master PCB with Teensy 4.1 and ADAU1966A DAC.

DB25 connectors wired according to the AES59-2012 standard. Extension headers on both master and satellite carry the TDM audio streams, I²C bus, and supply voltage to the satellite; the satellite DAC I²C address is selected via a solder jumper to avoid bus contention. The boards were professionally manufactured and assembled, with through-hole parts soldered manually. The hardware design is open source.¹

B. DAC Synchronisation

The system uses two ADAU1966A DACs to produce 32 channels in total. For spatial audio rendering, both DACs must output the same audio frame simultaneously. At 48 kHz, one sample corresponds to roughly 7.1 mm of acoustic path length, which is already sufficient to disturb localisation when loudspeaker positions are known to within a millimetre.

The two SAI peripherals of the iMX RT1062 microcontroller (SAI1 and SAI2) operate as independent TDM transmitters. SAI1 acts as the master: it generates MCLK, BCLK, and the frame-sync (LRCLK), which are distributed to the satellite board and also wired externally to the SAI2 input pins. SAI2 is configured as a clock slave (bit clock and frame sync as inputs), locking it to SAI1. The shared clocks are carried through the extension headers between the master and satellite boards, so both DACs share a single clock tree.

¹<https://gitlab.inria.fr/cwegener/teensy-adau1966a>

Synchronising the TDM data streams requires modifications to the Teensyduino audio library. The standard library starts each SAI transmitter independently inside its `begin()` method, which can leave up to one sample of offset between the two outputs. The modification adds a `syncStart()` mechanism that decouples start-up from initialisation. SAI2 is armed first by setting its transmit-enable bit and issuing a data-synchronisation barrier (`__DSB()`) to flush the Cortex-M7 store buffer, ensuring the write reaches SAI2 before SAI1 is touched. SAI1 then enables BCLK first, waits beyond the two-clock-cycle propagation time of SAI2's clock-domain crossing synchroniser, and finally enables the frame sync and its own transmitter. Both SAIs latch onto SAI1's first LRCLK pulse, producing sample-accurate output.

A second modification controls DMA channel priority. By declaring `AudioOutputTDM` before `AudioOutputTDM2` in the sketch, SAI1 receives the lower (higher-priority) DMA channel. Because SAI1 and SAI2 share a clock, their DMA half-buffer interrupts fire simultaneously every 128-sample block. SAI1's interrupt service routine wins arbitration and holds `update_responsibility`, so it calls `update_all()` first, evaluating the audio graph and filling the output buffers for both SAIs before SAI2's ISR reads them. Both modifications are part of the same library fork as the USB audio stack.

C. Audio I/O

The platform supports three audio input paths. USB audio is the primary mode used in this work and is described in detail in Section IV. The Ethernet socket has been verified at the hardware level, and audio streaming over the ANANAS protocol [12] is planned but requires modifications to the ANANAS client firmware that are currently in progress. SD card playback has been demonstrated with 48 kHz stereo files over the SPI interface available on the Teensy. Full standalone operation with 32-channel playback would require SDIO, which offers sufficient throughput; writing 32 audio channels to the SD card via SDIO has been validated in a separate lab-internal project, while reading at that channel count has not yet been tested.

IV. SOFTWARE

A. USB Audio Stack

The USB audio stack is based on an existing community extension of the Teensyduino USB audio stack [22] that already added multi-channel UAC2 support. This work extends it further to up to 32 channels, with the channel count selectable at compile time. Several bugs in the standard implementation were identified and corrected: a hardcoded `bInterval` value of 4 ms on the USB feedback endpoint caused audio dropouts at 12 or more channels (the correct value for high-speed USB is 1 ms); a `wMaxPacketSize` mismatch in the endpoint queue head caused descriptor errors at high channel counts; and a throughput-vs-size comparison error in the receive buffer selection logic was corrected. A proportional-integral feedback controller was added to regulate

the host-side buffer depth, replacing the original open-loop approach. Correct UAC2 spatial channel position descriptors are provided for all channel counts up to 32. The modified stack, including the TDM synchronisation changes, is publicly available.²

B. FAUST Spatial Audio DSP

Spatial audio processing is implemented in FAUST [16] and compiled to Teensyduino-compatible `AudioStream` subclasses using the `faust2teensy` tool, which generates C++ that integrates directly into the `AudioConnection` graph. Two rendering algorithms have been implemented. The WFS driving function targets a linear loudspeaker array reproducing a 2.5D point source. Following [23], the driving signal for loudspeaker $\mathbf{x}_0$ is

$$D_{2.5D}(\mathbf{x}_0, \omega) = \sqrt{\frac{2\pi}{c}} j\omega |\mathbf{x}_{\text{ref}} - \mathbf{x}_0| \cdot \frac{y_s}{r_s^2} \cdot e^{-j\omega r_s/c}, \quad (1)$$

where $r_s = |\mathbf{x}_s - \mathbf{x}_0|$ is the source-to-loudspeaker distance, y_s is the distance from the source to the array along the listening-plane normal, $\mathbf{x}_{\text{ref}}$ is a reference point in the listening plane at which the 2.5D synthesis is exact (set to 1 m in front of the array centre in this implementation), and c is the speed of sound. The amplitude weight y_s/r_s^2 decreases with source distance since $r_s \geq y_s$. In the time domain the $\sqrt{j\omega |\mathbf{x}_{\text{ref}} - \mathbf{x}_0| 2\pi/c}$ factor is realised as a second-order IIR half-differentiator with a per-loudspeaker gain. The ambisonic decoder implements third-order, three-dimensional ambisonic decoding with near-field compensation (NFC) filters, targeting a 16-loudspeaker dome arrangement. Both are authored by Pierre Lecomte using Faust and the Ambitools library [24]. The source code is publicly available.³

C. Audio Routing Architecture

Audio routing follows the Teensyduino `AudioConnection` graph model: each processing stage is an audio object, and objects are wired together at compile time. In the 32-channel USB streaming configuration, 32 `AudioConnection` instances route the USB input object to the two TDM output objects (channels 0–15 to SAI1, channels 16–31 to SAI2). The system operates at 48 kHz with a block size of 128 samples, giving a processing cycle of approximately 2.67 ms. FAUST DSP objects are inserted into the graph between the input and output stages.

V. VALIDATION

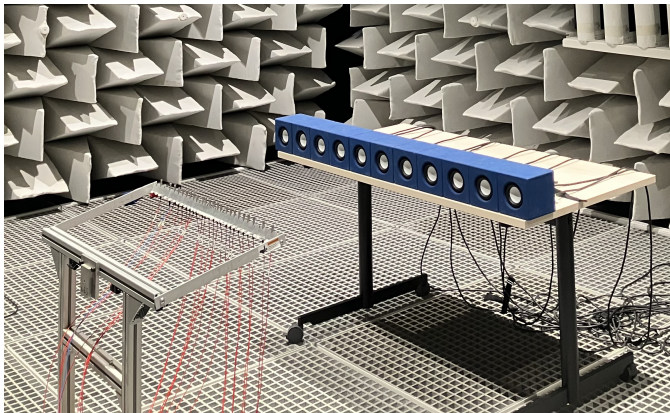
To verify the inter-channel synchronisation implemented in III-B, a single-sample impulse was sent simultaneously to all 32 outputs and the output pairs were probed with an oscilloscope. All 32 channels were found to be synchronised to sub-sample accuracy; the remaining jitter was below 100 ns, which was at the limit of the available test equipment.

²<https://gitlab.inria.fr/cwegener/teensy-4-usbaudio>

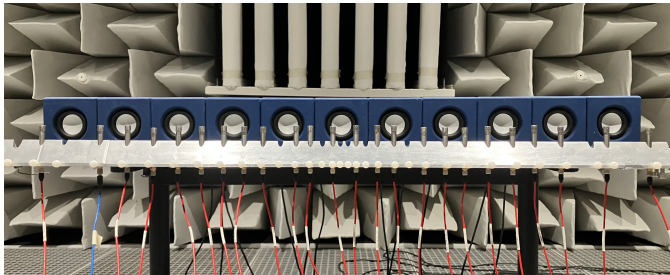
³<https://gitlab.inria.fr/cwegener/IS2-2026-source-code>

While this confirms that the timing precision of the hardware is sufficient for WFS reproduction, it does not validate the full signal chain. The remaining open questions are whether the FAUST-compiled WFS driving function integrates correctly with the Teensyduino audio graph at 32 channels without buffering artefacts, and whether hardware and software together qualify as a reliable research platform for spatial audio. The acoustic experiment addresses both: the excitation signal is a sine sweep read from the SD card, exercising the standalone playback path, and the WFS driving function runs on the platform during the measurement. By comparing the recorded spatial impulse responses against a real point source at the same position and against an acoustic simulation, we can assess whether the complete system reproduces the expected sound field. WFS was prioritised for rigorous acoustic measurement since its wavefront reconstruction depends directly on precise inter-channel timing, unlike the more level-based ambisonic decoding; the ambisonic decoder was validated only informally through listening tests.

VI. EXPERIMENTAL SETUP



(a) Overview of the measurement setup.



(b) Close-up of the loudspeaker and microphone arrays.

Fig. 3: Experimental setup in the anechoic chamber. The 11-loudspeaker WFS array (blue enclosure, rear) faces the 24-microphone linear array (front), separated by 0.72 m.

A. Loudspeaker Array and Room

Measurements were conducted at LMFA's anechoic chamber⁴ at an air temperature of 22.8 °C and relative humidity of 60%, giving a speed of sound of approximately 345 m/s. The WFS source array consists of 11 AuraSound NS3-193-8A1 3-inch full-range drivers⁵ mounted in a linear arrangement with 13 cm inter-speaker spacing, driven by the compute node described in this paper. A 24-element linear microphone array with an aperture of 708 mm was placed parallel to the loudspeaker array at a distance of 0.72 m, as shown in Fig. 3.

B. WFS Configuration

The driving function described in Section IV (Eq. 1) is compiled directly onto the compute node, with the virtual source position fixed at compile time. No spatial tapering window is applied to the array, so truncation artefacts from the finite aperture are present in the reproduced field.

The acoustic simulation used as a reference is computed using the same driving function: each loudspeaker contributes a fractional-delay impulse scaled by the amplitude weight y_s/r_s^2 , filtered by the IIR approximation of $\sqrt{j\omega}$ with $c = 340$ m/s, and summed at each microphone position. Propagation from loudspeaker to microphone uses the measured speed of sound $c = 345$ m/s.

C. Measurement Procedure

The excitation signal is a synchronised exponential sine sweep [25] from 100 Hz to 20 kHz, 5 s in duration at 44.1 kHz, generated offline and stored as a stereo WAV file on the SD card. The right channel carries a trigger pulse at the sweep onset, routed from a DAC output to the hardware trigger input of the acquisition system to start recording sample-accurately with the beginning of the sweep. A clean copy of the sweep was additionally routed from a DAC output directly to the acquisition system as a loopback reference channel, bypassing the WFS processing and the loudspeakers. The 24 microphone channels and the loopback reference were recorded simultaneously by a National Instruments DAQ system at 51.2 kHz.

Impulse responses were extracted by ESS deconvolution. The loopback reference was used to de-embed the bulk delay and gain of the DAC and ADC transfer functions from each microphone channel using a pure-delay model, which introduced fewer artefacts than regularised spectral division for this measurement. Each impulse response was truncated to 10 ms. To produce the 2D plots, per-channel impulse responses were ordered by microphone position x_a and spatially interpolated to 200 positions using a PCHIP spline. Each figure shows amplitude as a function of x_a and time t . The Python scripts used for impulse response extraction and spatial interpolation are available in a separate repository.⁶

⁴Laboratoire de Mécanique des Fluides et d'Acoustique, UMR 5509, École Centrale de Lyon,

⁵<https://www.erinsaudiocorner.com/driveunits/aurasound-ns3-193-8a1/>

⁶<https://gitlab.pmcs2i.ec-lyon.fr/cwegener/sync-ess-toolkit/>

D. Reference Measurements

To provide a ground truth for comparison, the same loudspeaker used in the WFS array was placed at the virtual source position and measured using the same sweep excitation, trigger, and acquisition chain. The recorded spatial impulse response of the real source is then compared side by side with the WFS reproduction at the same position and the free-field WFS synthesis simulation described in Section VI-B. The comparison is visual: the wavefront curvature, arrival time, and amplitude distribution across the microphone array serve as the basis for assessing the quality of the WFS reproduction.

VII. RESULTS

Fig. 4–7 show the measured 2D spatial impulse responses. Each plot shows amplitude as a function of array position x_a and time t ; wavefront curvature and arrival time encode the apparent source position and distance. This (x, t) -domain visualisation of linear array impulse responses follows the convention introduced by Berkhout et al. [26]. Source coordinates are given as (x_s, y_s) in metres, relative to the centre of the loudspeaker array. The y coordinate of the source position is the axis towards the microphone array, so a negative coordinate denotes a position behind the speaker array, away from the microphone array.

Fig. 4 compares the real source, the WFS simulation, and the WFS measurement at the same position $(x_s, y_s) = (0, -0.1)$ m. All three show a slightly curved wavefront arriving at approximately 2.4 ms, consistent with the expected travel time of $(0.72 + 0.1)$ m / 345 m/s ≈ 2.4 ms. The WFS reproduction closely matches both the real source and the simulation in wavefront shape and arrival time, with some additional low-level artefacts in the tail attributable to the finite array aperture and the absence of a tapering window.

Fig. 5 shows the same virtual source position with a deliberate one-block (128-sample, ≈ 2.67 ms) synchronisation delay artificially introduced between the two DAC outputs in software. A second spurious wavefront is clearly visible, displaced by one block in time, demonstrating the effect of unsynchronised outputs and confirming by contrast that the synchronised case in Fig. 4 produces a single coherent wavefront.

Fig. 6 shows the WFS virtual source at $x_s = 0$ m for increasing distances $y_s \in \{-0.1, -0.5, -1.0\}$ m. The wavefront arrival shifts to later times and the curvature progressively decreases with distance, as expected: at 1.0 m the wavefront is nearly flat, approaching a plane wave.

Fig. 7 shows the virtual source displaced to $x_s = 1.0$ m. The wavefront is clearly slanted, arriving earlier at the right end of the microphone array, which correctly reflects the off-axis source position. Across all positions, the reproduced wavefronts are consistent with the expected free-field behaviour of a point source at the specified coordinates.

VIII. CONCLUSION

We presented an open-source embedded platform for spatial audio rendering comprising a custom PCB, a Teensy mi-

crocontroller, and a pair of synchronised 16-channel DACs providing 32 balanced outputs in total. The platform is programmed through an accessible open-source toolchain: FAUST for spatial audio DSP and Teensyduino for embedded programming. We described modifications to the Teensyduino USB audio stack to support up to 32 channels, and a sample-accurate DAC synchronisation scheme requiring modifications to the TDM audio library.

The platform was validated acoustically by measuring the spatial impulse responses of an 11-loudspeaker WFS array in an anechoic chamber. The reproduced wavefronts closely match the expected free-field behaviour of a point source across a range of source positions, confirming that the DAC synchronisation is sufficient for WFS reproduction and that the FAUST-compiled driving function integrates correctly with the hardware. Compared to solutions built around general-purpose Linux computers [5], the platform requires no dedicated rendering servers or professional audio networking hardware such as Dante or MADI: a single PCB with an embedded microcontroller is sufficient for standalone operation. Compared to FPGA-based approaches [6, 7], the programming model is significantly more accessible, avoiding hardware description languages and FPGA-specific build flows. The hardware and software are fully open source. The Teensy’s hardware PTP support provides a direct path to multi-node networked installations; scaling beyond a single node requires only a PTP-capable network switch in addition to the compute nodes themselves.

IX. FUTURE WORK

Several directions remain open for the single-node platform. Standalone SD card operation with a physical user interface requires switching from SPI to SDIO for 32-channel playback, which has not yet been fully validated at that channel count. The per-node channel count could also be increased: placing a second ADAU1966A on each TDM bus to address the upper 16 time slots would yield 64 outputs per node, though the doubled master clock rate and its signal-integrity implications through the extension headers have not yet been evaluated. Further characterisation remains to be done: single-channel output signal quality (THD+N, frequency response, noise floor, transient response) and end-to-end latency across the USB and Ethernet input paths have not yet been measured.

Ethernet audio input is a planned extension: ANANAS [12] synchronises multiple compute nodes over a PTP-capable network, though integrating it requires client firmware modifications. Since the Teensy’s hardware PTP support is already exercised by ANANAS, the protocol could also extend to AES67 reception, letting the node receive audio from professional AES67-compatible devices.

The principal future direction is scaling to multi-node networked installations. Multiple compute nodes will be operated in parallel and synchronised over Ethernet using PTP, targeting high-channel-count spatial audio installations beyond what a single node can drive. The system is modular: each panel

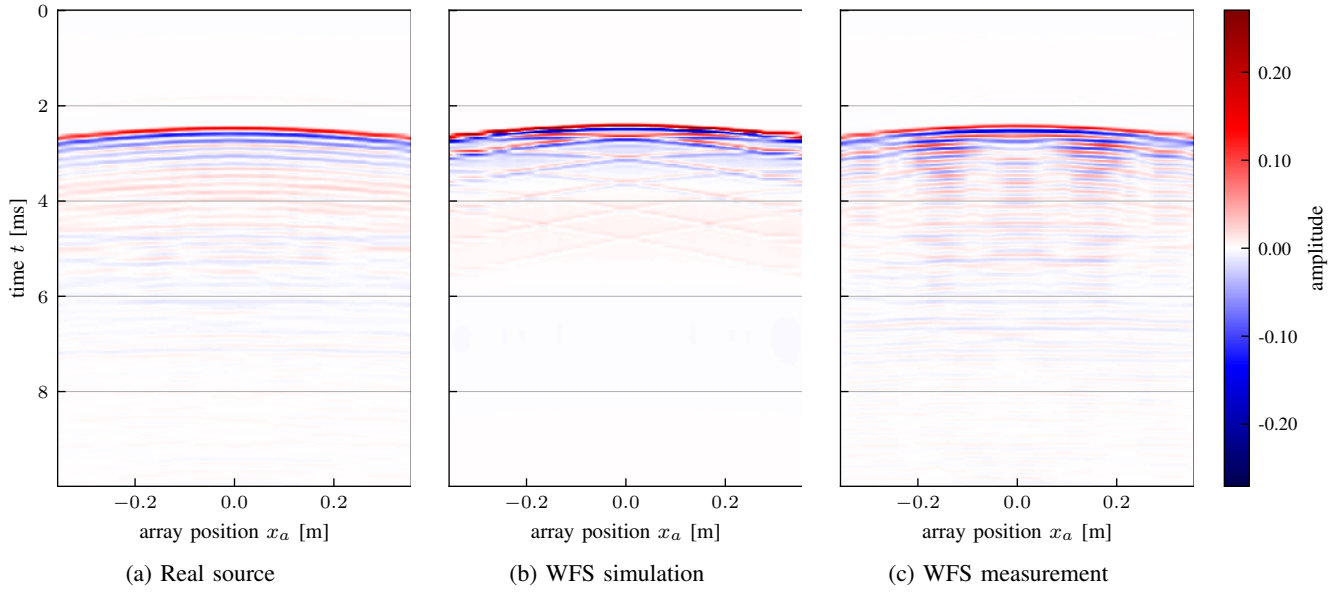


Fig. 4: Spatial impulse response of a real source, a 2.5D WFS simulation, and a WFS measurement at the source position $(x_s, y_s) = (0, -0.1)$ m. Seen from the microphone array, this is 0.82 m away, perpendicular to its centre.

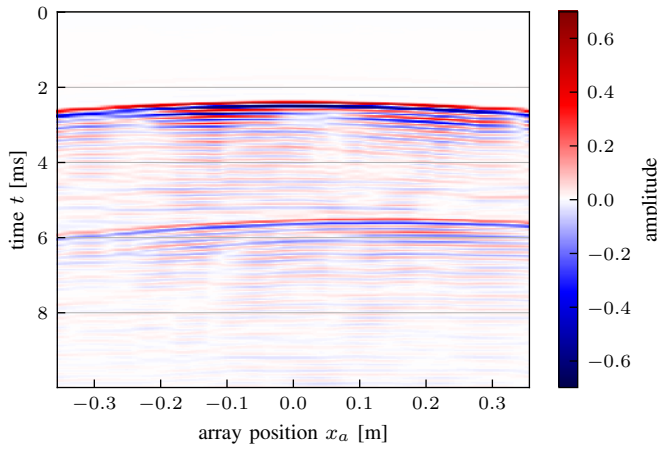


Fig. 5: Spatial impulse response of a WFS virtual source at $(x_s, y_s) = (0, -0.1)$ m, with a deliberate one-block synchronisation delay introduced between the two DACs.

combines a compute node, DACs, amplifiers, and a loudspeaker configuration for a specific rendering method (WFS, ambisonics, or beamforming), and panels can be mixed within one installation. A loudspeaker dome currently under construction will serve as the first testbed for evaluating distributed rendering accuracy across mixed spatial audio configurations.

REFERENCES

- [1] Charles Laubier. “Le son immersif, nouveau défi de l’industrie musicale pour séduire spectateurs et auditeurs”. In: *Le monde* (May 5, 2024).
- [2] Franz Zotter and Matthias Frank. *Ambisonics: A Practical 3D Audio Theory for Recording, Studio Production, Sound Reinforcement, and Virtual Reality*. Vol. 19. Springer Topics in Signal Processing. Cham: Springer International Publishing, 2019. DOI: 10.1007/978-3-030-17207-7. (Visited on 03/04/2025).
- [3] A. J. Berkhout, D. de Vries, and P. Vogel. “Acoustic Control by Wave Field Synthesis”. In: *The Journal of the Acoustical Society of America* 93.5 (May 1993), pp. 2764–2778. (Visited on 12/15/2022).
- [4] Fabio Olivieri et al. “Theoretical and experimental comparative analysis of beamforming methods for loudspeaker arrays under given performance constraints”. In: *Journal of Sound and Vibration* 373 (2016), pp. 302–324. DOI: 10.1016/j.jsv.2016.03.015.
- [5] Fares Schulz, Maximilian Weidauer, Stefan Weinzierl, and Henrik von Coler. “SeamLess: Distributed Spatial Audio Rendering on the Linux Audio Stack”. In: *Proceedings of the 20th Linux Audio Conference (LAC-26)*. Maynooth, Ireland, June 2026.
- [6] Maxime Popoff et al. *Audio DSP to FPGA Compilation: The Syfala Toolchain Approach*. Technical Report RR-9507. Univ Lyon, INSA Lyon, Inria, CITI, Grame, Emeraude, 2023.
- [7] 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 (SMC-24)*. Porto, Portugal, 2024.
- [8] Benjamin Quiédeville, Romain Michon, Tanguy Risset, and Stéphane Letz. “The Space Bar: An Embedded

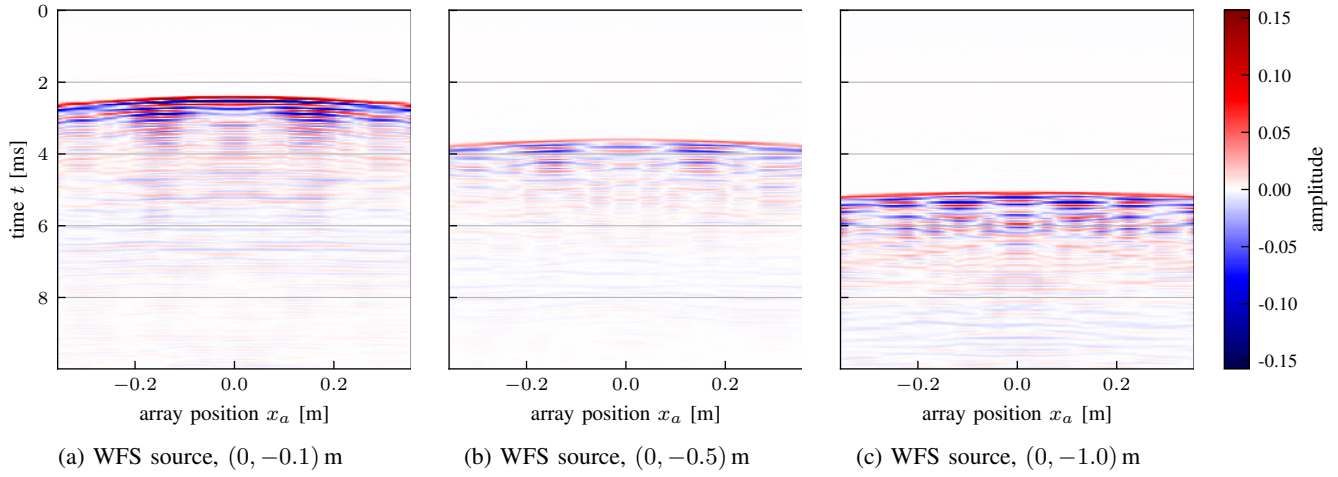


Fig. 6: Spatial impulse responses of a WFS virtual source at $x_s = 0$ m, for increasing source distance.

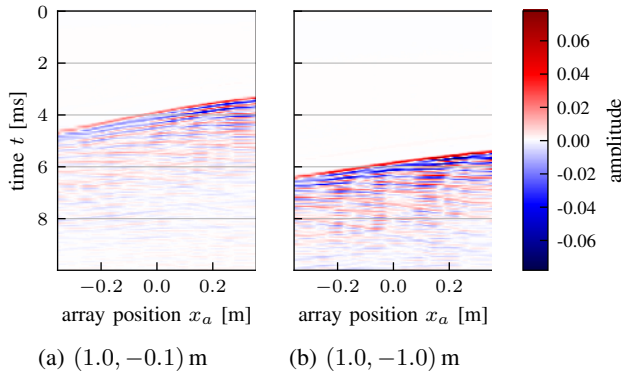


Fig. 7: Spatial impulse responses of a WFS virtual source at $x_s = 1$ m, for increasing source distance.

WFS Sound System”. In: *JIMLAC 2025 - Journées de l’Informatique Musicale*. Lyon, France, 2025.

- [9] Thomas Albert Rushton, Romain Michon, and Stéphane Letz. “A Microcontroller-based Network Client Towards Distributed Spatial Audio”. In: *Proceedings of the Sound and Music Computing Conference (SMC-23)*. Stockholm, Sweden, 2023.
- [10] Thomas Albert Rushton, Romain Michon, Stefania Serafin, Tanguy Risset, and Stéphane Letz. “Networked Microcontrollers for Accessible, Distributed Spatial Audio”. In: *Frontiers in Virtual Reality* 5 (Nov. 2024). ISSN: 2673-4192. DOI: 10.3389/frvir.2024.1391987. (Visited on 12/19/2024).
- [11] Thomas Rushton, Romain Michon, and Tanguy Risset. “All Together Now: A Synchronous Platform for Distributed Spatial Audio”. In: *Proceedings of the 22nd Sound and Music Computing Conference (SMC25)*. Graz, Austria: Institute of Electronic Music and Acoustics, University of Music and Performing Arts Graz, July 2025. DOI: 10.5281/ZENODO.15838463. (Visited on 03/15/2026).
- [12] Thomas Rushton, Romain Michon, and Tanguy Risset. “ANANAS: A Networked Audio System for Distributed Spatial Audio Applications”. Accepted for publication, in press. 2026.
- [13] Sean Devonport and Richard Foss. “The Distribution of Ambisonic and Point Source Rendering to Ethernet AVB Speakers”. In: *Proceedings of ICSA 2019*. Ilmenau, Germany, Sept. 2019.
- [14] Jose A. Belloch et al. “On the Performance of a GPU-based SoC in a Distributed Spatial Audio System”. In: *The Journal of Supercomputing* 77.7 (Apr. 2021), pp. 6920–6935. (Visited on 12/20/2022).
- [15] Jeff Koftinoff. “GPS-Disciplined AVB and Realtime Remote Audio Exchange: A Raspberry Pi 5 Implementation”. In: *Proceedings of the 20th Linux Audio Conference (LAC-26)*. Maynooth, Ireland, June 2026.
- [16] Yann Orlarey, Stéphane Letz, and Dominique Fober. “New Computational Paradigms for Computer Music”. In: ed. by Gerard Assayag. Paris, France: Delatour, 2009. Chap. Faust: an Efficient Functional Approach to DSP Programming.
- [17] Romain Michon, Yann Orlarey, Stéphane Letz, and Dominique 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, May 2019. (Visited on 12/15/2022).
- [18] Claudia Rinaldi and Carlo Centofanti. “The Musical Metaverse: Advancements and Applications in Networked Immersive Audio”. In: *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)*. 2024, pp. 1–7.
- [19] P. Cairns, H. Daffern, and G. Kearney. “Investigation of Server-Based Spatial Audio for Metaverse Concert Distribution”. In: *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)*. 2024, pp. 1–8. DOI: 10.1109/IS262782.2024.10704097.

- [20] 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 2022 Sound and Music Computing Conference (SMC-22)*. 2022.
- [21] Romain Michon, Joseph Bizien, Maxime Popoff, and Tanguy Risset. “Making Frugal Spatial Audio Systems Using Field-Programmable Gate Arrays”. In: *Proceedings of the 2023 New Interfaces for Musical Expression Conference (NIME-23)*. May 2023. (Visited on 12/19/2024).
- [22] alex6679. *teensy-4-usbAudio*. URL: <https://github.com/alex6679/teensy-4-usbAudio> (visited on 07/01/2026).
- [23] Sascha Spors, Rudolf Rabenstein, and Jens Ahrens. “The Theory of Wave Field Synthesis Revisited”. In: *Audio Engineering Society Convention 124*. May 2008.
- [24] Pierre Lecomte. “Ambitools: Tools For Sound Field Synthesis Using Higher Order Ambisonics - V1.0”. In: *Proceedings of the 1st International Faust Conference (IFC-18)*. Mainz, Germany, 2018.
- [25] Antonin Novak, Laurent Simon, and Pierrick Lotton. “Synchronized Swept-Sine: Theory, Application, and Implementation”. In: *Journal of the Audio Engineering Society* 63.10 (2015), pp. 786–798. DOI: 10.17743/jaes.2015.0071.
- [26] Augustinus J. Berkhout, Diemer de Vries, and J. J. Sonke. “Array technology for acoustic wave field analysis in enclosures”. In: *The Journal of the Acoustical Society of America* 102.5 (1997), pp. 2757–2770. DOI: 10.1121/1.420330.