TSENG, KWEI-WEN 曾奎紋

he/him, DSP Engineer kweiwen.tseng@gmail.com https://github.com/kweiwen +886-921833641

EDUCATION

National Chiao Tung University **Sep, 2013 – Jun, 2016**
Music Technology, Master of Science **Hsinchu, Taiwan**

National Sun Yat-sen University **Sep, 2009 – Jun, 2013**
Marine Biotechnology, Bachelor of Science **Kaohsiung, Taiwan**

INDUSTRY EXPERIENCE

Waves Audio **Aug, 2024 – Present**
Senior DSP Engineer **Taipei, Taiwan**

- Designed a real-time source separation with low RAM usage and MCPS[A1], for karaoke applications, achieving 3.7/0.99 M parameters, 17/15 dB SI-SDR, and 406ms latency.
- Designed and implemented novel XTC algorithms[A2].
- Maintained immersive and spatial audio algorithms based on decomposed HRTFs and head tracking.

Li Auto **Dec, 2021 – Jun, 2024**
Senior DSP Engineer **Shanghai, China**

- Oversaw development of audio algorithms and a tuning tool.
- Ported algorithms from x86 to Hexagon DSP (8155/8295) and SHARC DSP (21569).
- Developed RoFormer source separation, Neural Reverberator[A3], and Upmixer[A4]; optimized via lookup tables, block processing, and SIMD intrinsics, reducing computational cost by **30%**.
- Conducted preliminary research on FxLMS-based active engine noise control algorithms, granular synthesis for engine sound design, and VBAP-based spatial audio.
- Provided mentorship to junior engineers.

Lava Music **Jan, 2021 – Dec, 2021**
Senior DSP Engineer **Guangzhou, China**

- Led audio algorithm and system development[A5] on SHARC DSP (21479).
- Developed AEC, CNN-based SED, time-varying FDN, PVOC, time-varying morphing filters, and synthesis algorithms (Karplus-Strong, FM); optimized via Taylor/Padé approximations, reducing cost by **40%**.

Tymphany Acoustic Technology **Aug, 2019 – Jan, 2021**
DSP Engineer **Taipei, Taiwan**

- Designed **Flow DSP**[A6], which were adopted by **Bose**, **Google**, **Jabra**, **LG**, **Marshall**, **NTT**, and **Rivian**.
- Implemented LMS Filter, Schroeder Reverberator, Linkwitz–Riley Filter.

Tymphany Acoustic Technology **Sep, 2018 – Aug, 2019**
Electrical Engineer **Taipei, Taiwan**

- Designed schematics, PCB layouts, and test fixtures; managed BOM.
- Developed an automated test and measurement platform using PyVISA.
- Developed a user-hardware interface using CPLD.

Positive Grid **Oct, 2017 – Feb, 2018**
Junior Electrical Engineer **Taipei, Taiwan**

- Designed schematics, PCB layouts, fixtures, and managed BOM; conducted failure sample analysis and hands-on troubleshooting.

PUBLICATIONS

- Kweiwen Tseng. "Building an Open Source Modular Synthesizer Based on the Integration of Raspberry Pi, Python and Pure Data". In: Workshop on Computer Music and Audio Technology, 2015

PATENTS

- Kweiwen Tseng. "Even Order Harmonic Compressor Using Pitch-Shifting, Full-Wave Rectifier and Amplitude Modulation". CN.202311220680.3. Sept. 2023
- Kweiwen Tseng. "Novel Audio Tuning Strategy Using Neural Network Stem Separation with VBAP and U-Net". CN.202311189701.X. Sept. 2023
- Kweiwen Tseng. "Surround Algorithm Based on Time-Domain Filterbank and Voltage Control Amplifier". CN.202311181121.6. Sept. 2023
- Kweiwen Tseng. "A HCI Interface and Spatial Room Model with Perceptual Acoustic Properties". CN.202311177268.8. Sept. 2023
- Kweiwen Tseng. "Novel Calibration Implementation Using Least-Squares IIR Filter on Mel-Scale". CN.202211612831.5. Dec. 2022

TEACHING ASSISTANT

- *Pure Data, VCV Rack, and Generative Music*, Music In Creation, Shanghai, 2023.
- *Introduction to Music Technology*, Music In Creation, Shanghai, 2022.
- *Performing Practice of Digital Orchestra*, NCTU, Hsinchu, 2014.
- *Computer Media: MSP*, NCTU, Hsinchu, 2013.
- *Real-time Digital Audio-visual Synthesis II*, NCTU, Hsinchu, 2013.

PRESENTATIONS

- *Neural Networks in Audio: Beyond Fancy Filters!*, IRCAM, Taipei, 2025.
- *Reverse Engineering 101: Buchla 700 as a Case Study*, IRCAM, Taipei, 2025.
- *The Theory and Implementation of Algorithmic Composition*, NTHU, Hsinchu, 2025.
- *From Design to Implementation: The Complete Guide to VST Development*, Digilog, Taipei, 2025.

GITHUB PROJECTS

- K700, a VST remake of the Buchla 700.
- REMORA, implementing audio processing and control interfaces on the SHARC DSP (21489).
- Puannhi, an asynchronous, time-variant feedback delay network for reverberation.
- Vermilliad, an open-source platform for building digital modular synthesizers. (Master's Thesis)

PRODUCTION & PUBLIC ADDRESS EXPERIENCE

- Zhao Lin –Pot Band Show, Mixing/Mastering, 2025.
- Morio Agata, Live Sound, Acoustic Guitar & Vocals, 2016.
- Lim Giong, Live Sound, DJ Set, 2016.
- Panai Kusui, Live Sound, Acoustic Guitar & Vocals, 2015.
- Ayugo Huang, Live Sound, Full Band, 2015.

SKILLSETS

- Programming: C, C++, LaTeX, Python.
- IDE: CCES, Jupyter Notebook, PyCharm, Visual Studio, Xcode.
- API and Framework: JUCE, PortAudio, PySide, VST SDK, WebAudio.
- Machine Learning: PyTorch, PyTorch-Lightning.
- Audio / Music Programming: AudioMulch, Max/MSP, Pure Data, SuperCollider, VCV Rack.
- Electrical Design: Altium Designer, EAGLE, LTSpice.

APPENDIX 1

INTRODUCTION

At Waves Audio, we developed a real-time Music Source Separation (MSS) system under strict resource constraints defined by the product team: low latency, low RAM consumption, and low MCPS budget. These requirements ruled out many offline or high-complexity architectures and motivated a purpose-built design.

Our solution combines Pseudo-Quadrature Mirror Filter (PQMF) sub-band decomposition, a UNet architecture operating in the STFT domain, and a decoupled complex mask prediction. A hybrid bottleneck module enables cross-channel and cross-subband interaction without introducing look-ahead latency. The overall pipeline is illustrated in FIG 1, and the internal structure of the UNet is detailed in FIG 2.

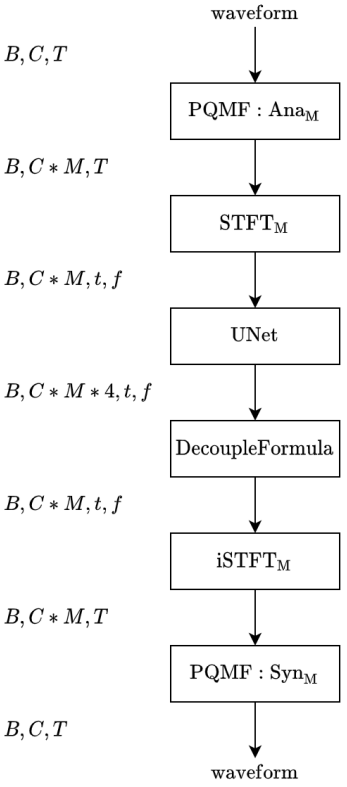


Figure 1: Overall Pipeline

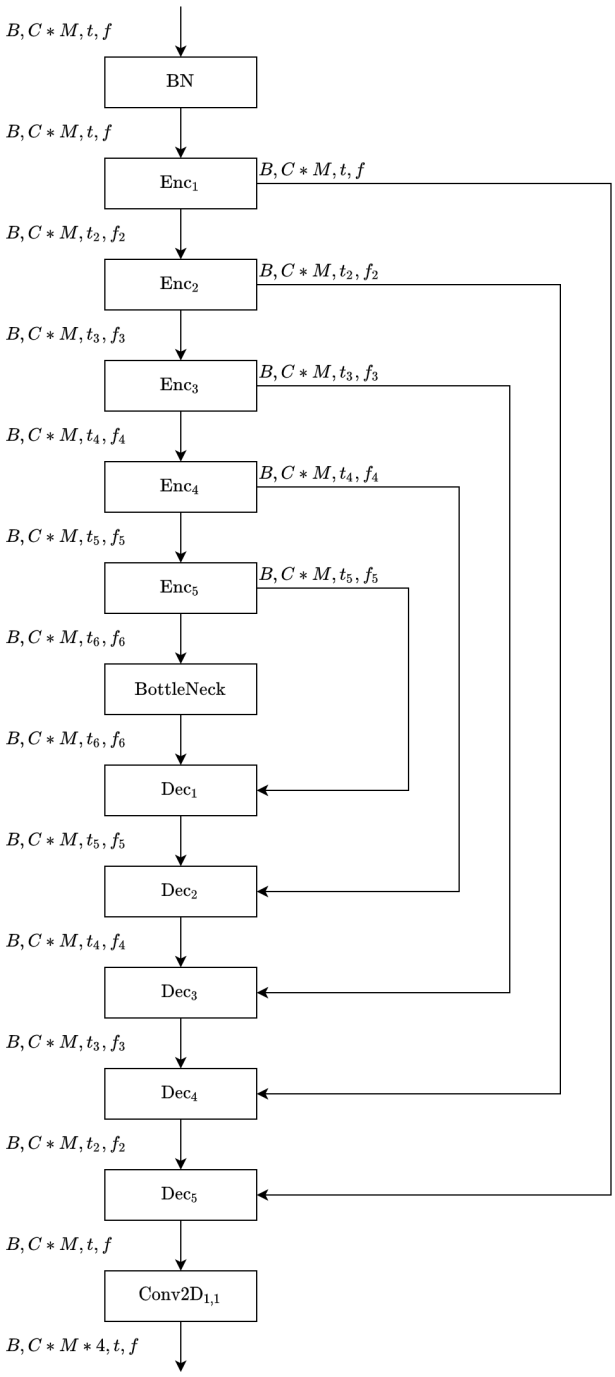


Figure 2: UNet Architecture

PQMF Sub-band Decomposition

To reduce the computational cost of subsequent processing, we apply PQMF analysis to decompose the input waveform into M sub-bands, reducing the time-domain sample rate by a factor of M in each band. The filter bank is designed with a Kaiser window, whose roll-off and stop-band attenuation are controlled by the β parameter. This design satisfies the near perfect reconstruction condition, ensuring minimal information loss through the analysis–synthesis pipeline.

PQMF-based sub-band processing is well established in neural vocoders such as MelGAN[2], where processing in the sub-band domain significantly reduces the computational load while preserving perceptual audio quality. We adopt the same analysis–synthesis pipeline here to compress the UNet's input dimensionality.

UNet Architecture

Motivated by the results of Kong et al.[1] and Liu et al.[3], who demonstrated effective separation by combining PQMF sub-band decomposition with a UNet operating on sub-band spectrograms, we adopt the same PQMF–STFT pipeline as the backbone of our architecture. Operating in the frequency domain allows the model to exploit spectral structure directly.

The core building block is the LSTM Block, which consists of two stacked unidirectional LSTMs with Layer Normalisation and Leaky ReLU activations, connected via residual skip connections. Using unidirectional LSTMs ensures that the model processes audio causally, with no dependency on future frames.

Each encoder block (FIG 3) wraps one LSTM Block followed by average pooling for downsampling along the time–frequency axes. Each decoder block (FIG 4) upsamples via bilinear interpolation, projects the channel dimension with a linear layer, adds the corresponding encoder skip connection, and passes the result through an LSTM Block.

We also evaluated a CausalConv MobileNet variant as an alternative to the LSTM backbone. Although causal convolutions avoid look-ahead, the LSTM-based architecture achieved higher SDR in our ablation, and we therefore adopted it as the primary design.

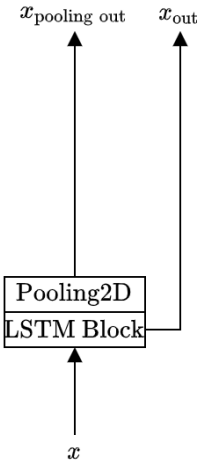


Figure 3: Encoder

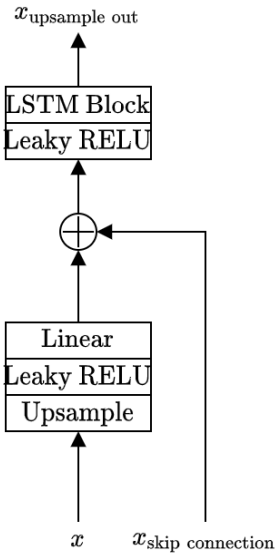


Figure 4: Decoder

BottleNeck

Because each encoder LSTM operates independently per channel, there is no cross-channel or cross-subband information exchange before the bottleneck. Enabling such interaction at this stage is critical for the model to reason about source identity across all frequency bands simultaneously.

We explored two bottleneck designs. A fully connected bottleneck is computationally lightweight and yields acceptable separation quality. As a stronger alternative, we designed a Transformer-based bottleneck (CF-Transformer, FIG 5) that applies self-attention over the $C \times F$ token sequence, where C is the number of

channels and F the number of frequency bins. Crucially, the time dimension T is folded into the batch dimension prior to attention:

$$\mathbf{x} \in \mathbb{R}^{B\times C\times T\times F} \xrightarrow{\text{reshape}} \mathbb{R}^{(B\cdot T)\times(C\cdot F)\times 1} \tag{1}$$

As a result, self-attention operates solely along the $C \times F$ axis and introduces no look-ahead along the time axis, preserving the real-time constraint. Each transformer layer uses LayerScale[9] to stabilise training at initialisation.

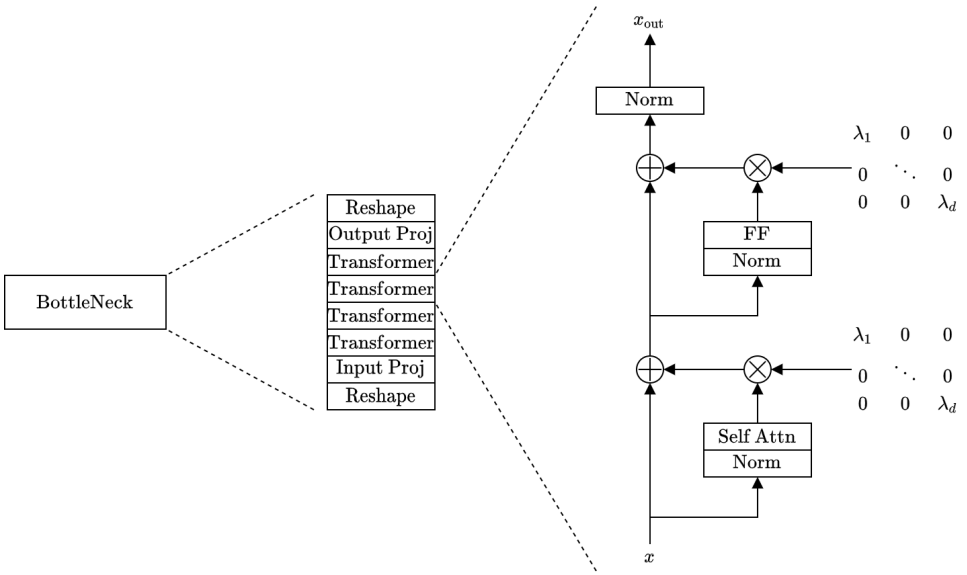


Figure 5: CFTransformer BottleNeck

Decouple Formula

Following Kong et al.[1] and Wang et al.[11], we predict a decoupled complex mask rather than estimating the real and imaginary parts of the spectrogram directly. For each target source, the network outputs four maps: mask magnitude $\hat{M_{\text{mag}}}$, and the real and imaginary components of the mask phase ($\hat{M}_{\text{re}}$, $\hat{M}_{\text{im}}$), from which $\cos\angle\hat{M}$ and $\sin\angle\hat{M}$ are derived, along with a linear residual term Q. The separated spectrogram is then reconstructed as

$$\hat{S} = |\hat{S}|e^{j(\angle\hat{M}+\angle X)}, \tag{2}$$

where $|\hat{S}| = \text{ReLU}(\hat{M_{\text{mag}}} \bigodot |X| + \hat{Q})$ and the output phase is obtained by rotating the input phase by $\angle\hat{M}$. This formulation has been shown to improve SDR and phase estimation compared to real-valued mask prediction.

Dataset

Training data is drawn from three public dataset: MUSDB18-HQ[7][8], MoisesDB[6], and Slakh2100[5]. To improve coverage of real-world production material, we supplement these with two sources: in-house stems generated using a Roformer-based separation pipeline[10][4], and a licensed dataset acquired from a third-party vendor.

REFERENCES

[1] Qiuqiang Kong et al. "Decoupling Magnitude and Phase Estimation with Deep ResUNet for Music Source Separation". In: *International Society for Music Information Retrieval Conference*. 2021. URL: https://api.semanticscholar.org/CorpusID:237491546.

[2] Kundan Kumar et al. "MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis". In: *Neural Information Processing Systems*. 2019. URL: https://api.semanticscholar.org/CorpusID:202777813.

[3] Haohe Liu et al. "Channel-wise Subband Input for Better Voice and Accompaniment Separation on High Resolution Music". In: *ArXiv* abs/2008.05216 (2020). URL: https://api.semanticscholar.org/CorpusID:221104000.

[4] Wei-Tsung Lu et al. "Music Source Separation With Band-Split Rope Transformer". In: *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2023), pp. 481–485. URL: https://api.semanticscholar.org/CorpusID:261556702.

[5] Ethan Manilow et al. "Cutting Music Source Separation Some Slakh: A Dataset to Study the Impact of Training Data Quality and Quantity". In: *2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)* (2019), pp. 45–49. URL: https://api.semanticscholar.org/CorpusID:202660940.

[6] Igor Pereira et al. "Moisesdb: A dataset for source separation beyond 4-stems". In: *ArXiv* abs/2307.15913 (2023). URL: https://api.semanticscholar.org/CorpusID:260333834.

[7] Zafar Rafii et al. "MUSDB18 - a corpus for music separation". In: 2017. URL: https://api.semanticscholar.org/CorpusID:199578935.

[8] Zafar Rafii et al. "MUSDB18-HQ - an uncompressed version of MUSDB18". In: 2019. URL: https://api.semanticscholar.org/CorpusID:212982970.

[9] Hugo Touvron et al. "Going deeper with Image Transformers". In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)* (2021), pp. 32–42. URL: https://api.semanticscholar.org/CorpusID:232428161.

[10] Ju-Chiang Wang, Wei-Tsung Lu, and Minz Won. "Mel-Band RoFormer for Music Source Separation". In: *ArXiv* abs/2310.01809 (2023). URL: https://api.semanticscholar.org/CorpusID:263608675.

[11] Chun-Hsiang Wang et al. "Improving Real-Time Music Accompaniment Separation with MMDenseNet". In: *2024 27th Conference of the Oriental COCOSDA International Committee for the Co-ordination and Standardisation of Speech Databases and Assessment Techniques (O-COCOSDA)* (2024), pp. 1–6. URL: https://api.semanticscholar.org/CorpusID:270869645.

APPENDIX 2

INTRODUCTION

What is Cross-Talk?

Cross-talk(FIG 6) arises because the sound from each loudspeaker travels to both ears. If L and R denote the left and right channel signals, the acoustic mixing at the ears can be modelled as

$$\begin{bmatrix} L & R \end{bmatrix} \begin{bmatrix} 1 & gz^{-\tau} \\ gz^{-\tau} & 1 \end{bmatrix} = \begin{bmatrix} L + Rgz^{-\tau} & R + Lgz^{-\tau} \end{bmatrix} \tag{3}$$

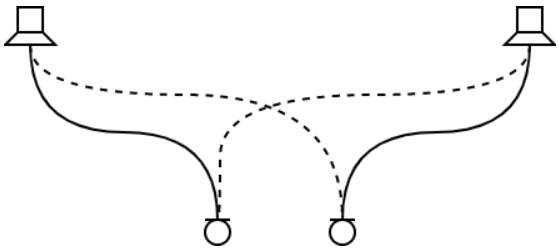


Figure 6: solid: desired signal; dashed: signal to be cancelled

where g is the inter-channel gain (attenuation due to distance and acoustic spreading) and $z^{-\tau}$ represents the delay of sound travelling from one speaker to the opposite ear.

Inverse Filtering

The most direct solution is to design a control matrix H that inverts the acoustic matrix C, satisfying $CH = I$. Solving the 2×2 system gives

$$H = \begin{bmatrix} \dfrac{1}{1-(gz^{-\tau})^2} & \dfrac{-gz^{-\tau}}{1-(gz^{-\tau})^2} \\ \dfrac{-gz^{-\tau}}{1-(gz^{-\tau})^2} & \dfrac{1}{1-(gz^{-\tau})^2} \end{bmatrix} \tag{4}$$

The corresponding ipsilateral (ipsi) and contralateral (cont) transfer functions are

$$H_{\text{ipsi}} = \frac{1}{1-(gz^{-\tau})^2} \tag{5}$$

$$H_{\text{cont}} = \frac{-gz^{-\tau}}{1-(gz^{-\tau})^2} \tag{6}$$

Problems with Inverse Filtering

The naive inverse filter has three practical limitations. First, the denominator $1-(gz^{-\tau})^2$ approaches zero at resonance frequencies, making the matrix ill-conditioned. This produces *coloration*[2] (deviation of $|H_{\text{ipsi}}(i\omega) + H_{\text{cont}}(i\omega)|$ and $|H_{\text{ipsi}}(i\omega) - H_{\text{cont}}(i\omega)|$) and *distortion* (excessive gain drives the loudspeaker into its nonlinear regime). Second, the pure gain–delay model (g, $z^{-\tau}$) ignores the headshadow effect. Third, the filter is optimised for a single listener position, so the sweet spot is typically only a few centimetres wide.

Although combining BACCH with a headshadow model addresses first and second limitation, the BACCH framework itself is computationally demanding: the ipsilateral and contralateral filters must be realised either as long FIR convolutions in the time domain or as block-wise spectral multiplications in the frequency domain. Therefore, a computation efficient method is proposed in the following section.

Head Shadow + Regularization (IIR Approach)

We propose an alternative formulation that integrates Duda's spherical headshadow model[1] directly into a *regularized* XTC framework and implements the result as a cascade of IIR sections.

Let $h = h_{\text{cont}}/h_{\text{ipsi}}$ denote the normalized headshadow filter, so that the inter-channel gain g and delay τ retain their original physical interpretation. The normalized headshadow filter response h and its time-reversed counterpart $\bar{h}$ are

$$h = \frac{b_0 + b_1 z^{-1}}{a_0 + a_1 z^{-1}} \qquad \bar{h} = \frac{b_1 + b_0 z^{-1}}{a_1 + a_0 z^{-1}} \tag{7}$$

Substituting into the XTC framework gives the headshadow-integrated IIR filters:

$$H_{\text{ipsi-hs}} = \frac{g^2\bar{h}^2 - (\beta+1)\, z^{-2\tau}}{g^2\bar{h}^2 - \left[(g^2 h\bar{h} + \beta)^2 + 2\beta + 1\right] z^{-2\tau} + g^2 h^2\, z^{-4\tau}} \tag{8}$$

$$H_{\text{cont-hs}} = \frac{-(g^2 h\bar{h} + \beta)\, g\bar{h}\, z^{-\tau} + gh\, z^{-3\tau}}{g^2\bar{h}^2 - \left[(g^2 h\bar{h} + \beta)^2 + 2\beta + 1\right] z^{-2\tau} + g^2 h^2\, z^{-4\tau}} \tag{9}$$

where the polynomial products h^2, $\bar{h}^2$, and $h\bar{h}$ are expanded as second-order rational functions in z^{-1}, which is independent of the delay τ. Therefore, we could expand the polynomial as a high order IIR.

Unstable Pole Reflection

In addition, the IIR denominator may contain roots outside the unit circle. Any unstable pole p with $|p| > 1$ is replaced by its conjugate-reciprocal $1/p^*$, preserving the magnitude response while restoring stability. Finally, the filter is then simplified as a cascade of second-order sections in direct-form II.

Simulation Setup

All experiments are implemented in Python (NumPy, SciPy, Matplotlib, PyTorch) under an ideal, anechoic, and free-field assumption. A symmetric two-loudspeaker geometry is parameterised by speaker-to-head distance, inter-speaker half-angle, and the integer ITD delay τ in samples. Head shadowing follows Duda's spherical-head HRTF with radius $a = 8.5\,\text{cm}$, fitted as first-order IIR pairs $(h, \bar{h})$; the cross-talk gain g is set by free-field propagation loss. Signals are processed at 44.1 kHz over 20 Hz–20 kHz.

Four variants are benchmarked within this single framework:

(i) naive inverse filtering via FIR truncation ($\beta = 0$);

(ii) BACCH;

(iii) proposed BACCH design with headshadow-aware $\beta(\omega)$.

(iv) proposed IIR design with regularization and headshadow-aware $\beta(\omega)$;

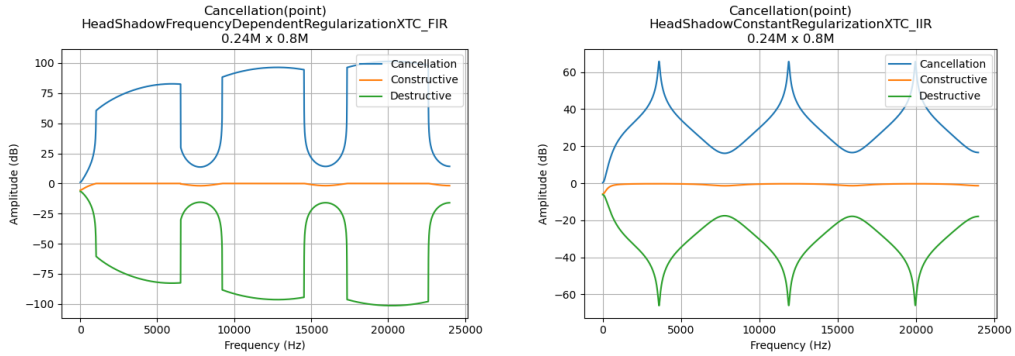


Figure 7: Point cancellation: (iii) vs. (iv)

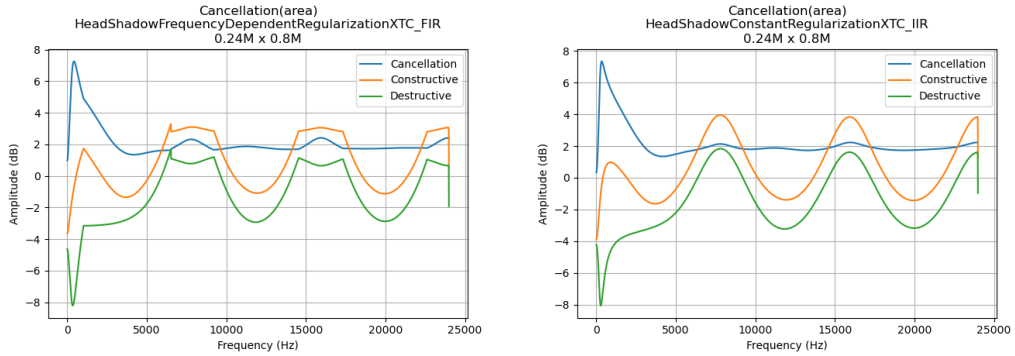


Figure 8: Area-averaged cancellation: (iii) and (iv), evaluated by laterally displacing the listener from −15 cm to +15 cm in 1 cm steps.

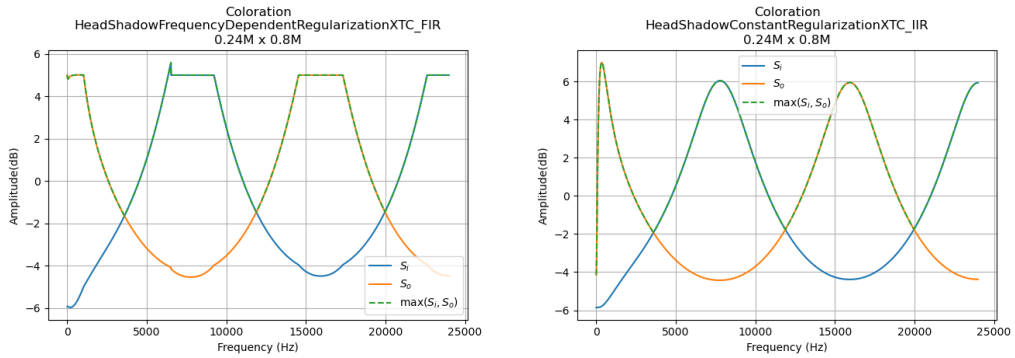


Figure 9: Coloration (S_i, S_o, $\max(S_i, S_o)$): (iii) vs. (iv)

Discussion

Once the headshadow model is incorporated and performance is evaluated over a listening area rather than a single point, the (iv) and (iii) achieve comparable cancellation and coloration levels (FIG 8 and 9). Therefore, the advantage of the (iv) lies not in acoustic performance but in computational efficiency.

REFERENCES

[1] C. Phillip Brown and Richard O. Duda. "A structural model for binaural sound synthesis". In: *IEEE Trans. Speech Audio Process.* 6 (1998), pp. 476–488. URL: https://api.semanticscholar.org/CorpusID:18043708.

[2] Edgar Choueiri. "Optimal Crosstalk Cancellation for Binaural Audio with Two Loudspeakers". In: 2010. URL: https://api.semanticscholar.org/CorpusID:16069611.

APPENDIX 3

INTRODUCTION

We developed a user-configurable neural reverberator that decodes room acoustics into real-time FDN parameters, allowing users to customize and share room settings through a community platform, including dimensions, moisture, temperature, wall material, and virtual speaker/audience position.

ISM/RT-based RIR convolution is computational intensively for real-time use; conventional FDN reverberators lack intuitive control over spatial characteristics. Our solution addresses this via two threads: the first (FIG 10) handles UI and neural network-driven RIR analysis; the second (FIG 11) handles real-time audio via a hybrid convolution–FDN technique[6][2].

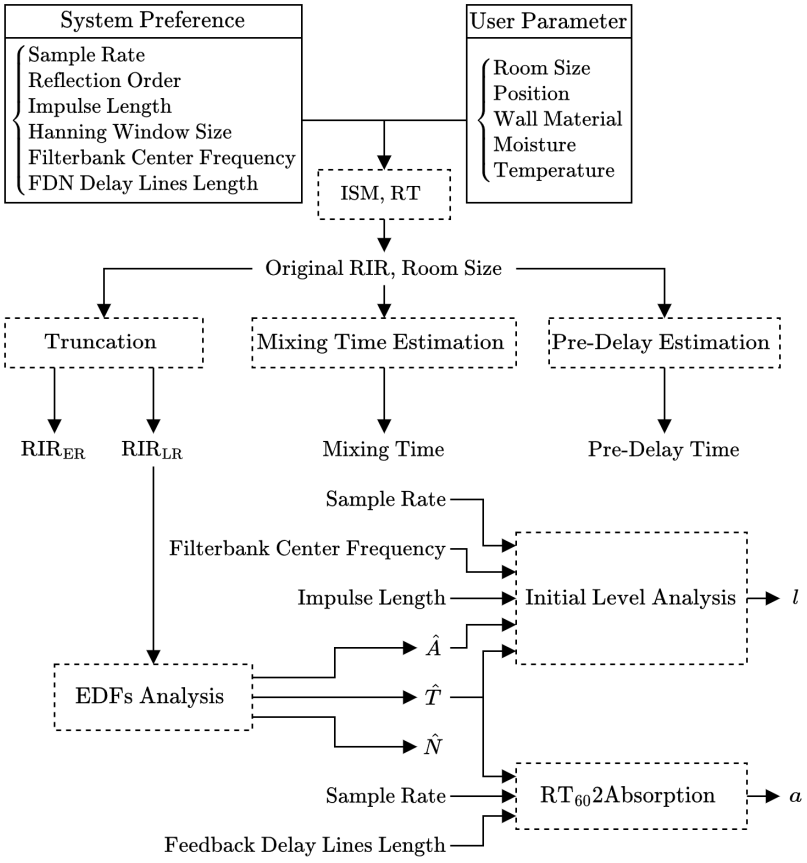


Figure 10: User Interface, Estimation, and Analysis

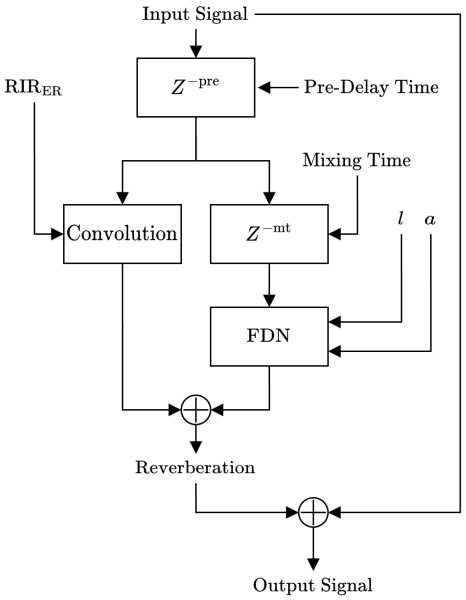


Figure 11: Fast Convolution and FDN

To optimize performance, the RIR is split into early reflections, processed via FIR convolution, and late reverberation, represented as Energy Decay Functions (EDFs). Bayesian analysis decodes the EDFs into frequency response and reverberation time parameters aligned with the FDN delay lines, enabling controllable reproduction of acoustic characteristics.

The implementation builds on PyRoomAcoustics[10] with several modifications to improve RIR accuracy: negative coefficients[7] are included, and RIR Sinc Windows in a clustered data structure are used to discriminate between reflection orders for more precise truncation.

In the truncation function, mixing time is estimated from room size[6] and used as a boundary between early reflections and late reverberation. The RIR is divided by reflection cluster number (FIG 12) rather than by sample-domain mixing time, yielding more precise truncation and frequency response.

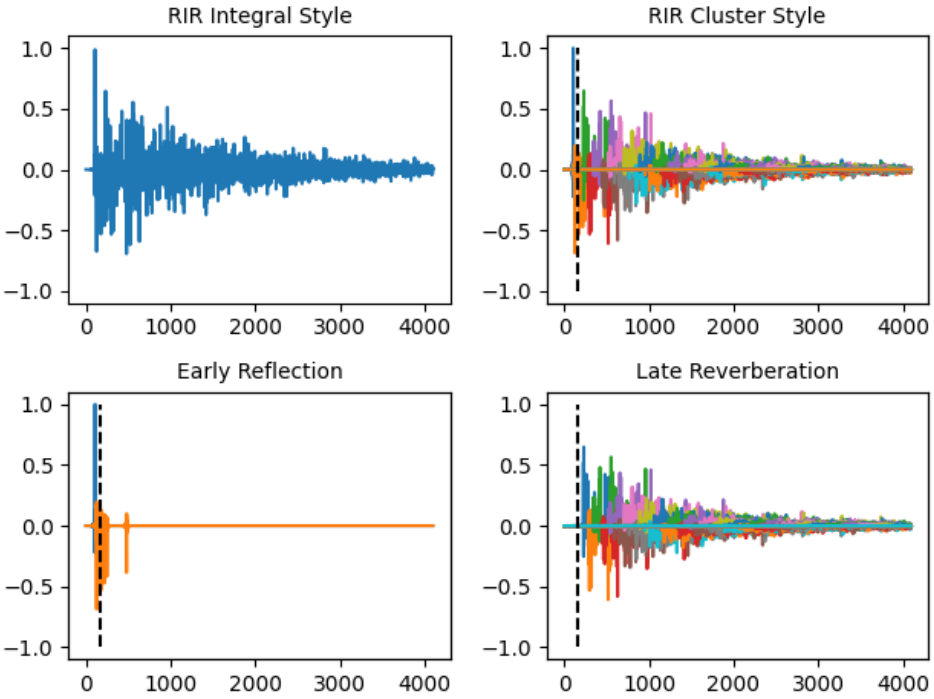


Figure 12: Truncation Result, dash line suggests the mixing time derived from room size.

To reduce computational cost, initial zero-valued RIR coefficients are removed by identifying the minimum speaker–audience distance, with delay compensation applied at the start of processing (See FIG 11).

Late reverberation RIR is converted to EDFs via Bayesian analysis[1], estimating the per-band decay parameters $\hat{A}$ (initial level), $\hat{T}$ (reverberation time), and $\hat{N}$ (noise floor). As an alternative, we integrate DecayFitNet[5], a neural network trained to perform multi-exponential decay analysis directly from the energy decay curve.

DecayFitNet takes a sub-band EDF as input and regresses $\hat{A}$, $\hat{T}$, and $\hat{N}$ in a single forward pass, replacing the iterative Bayesian fitting with a fixed-cost inference step. This is particularly advantageous in our pipeline since the ISM produces single-slope decays, making the network's output well-conditioned for subsequent FDN parameter mapping. Both estimation paths are managed via FDNTB[11]. Absorption filter coefficients and output filter initial levels are then computed using least-squares optimization[1] based on the estimated $\hat{T}$ and FDN delay line length.

In the FDN process, I have designed a FDN (FIG 13) based on Complementary AllPass Filters[3][4] as an alternative to graphic equalizer (GEQ) FDNs[8][9] (FIG 14). Since each filter bank band operates independently, its gain can be adjusted without affecting the feedback matrix input gain, eliminating the need for separate output filters.

[1] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.lsq_linear.html`

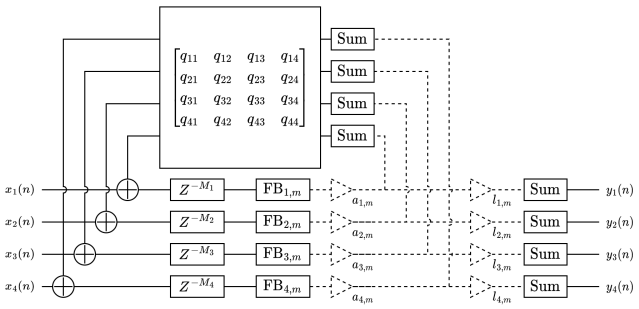


Figure 13: Filterbank FDN

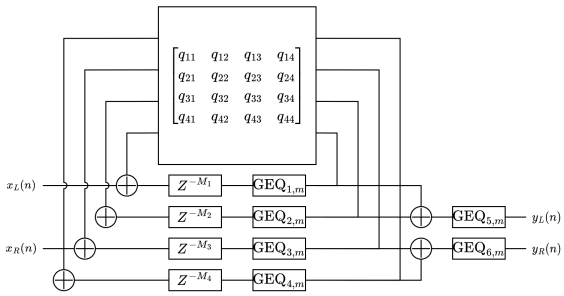
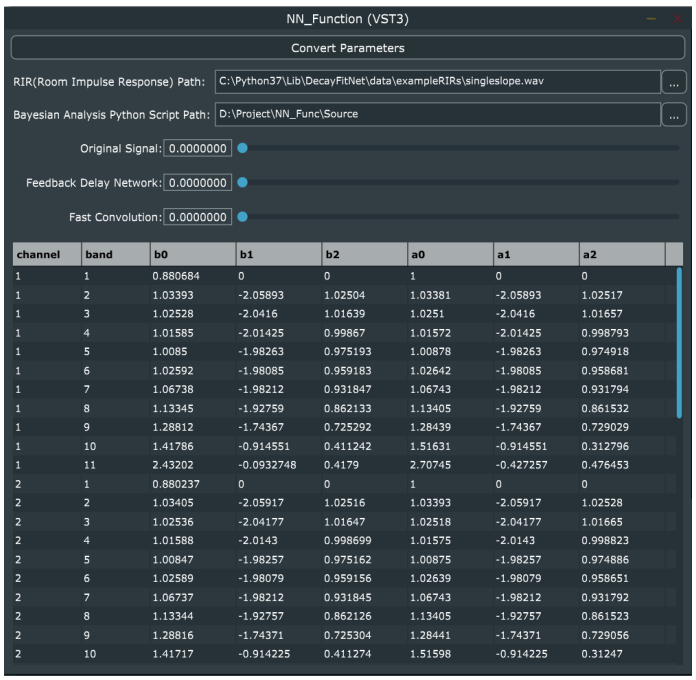


Figure 14: GEQ FDN



channel	band	b0	b1	b2	a0	a1	a2
1	1	0.880684	0	0	1	0	0
1	2	1.03393	-2.05893	1.02504	1.03381	-2.05893	1.02517
1	3	1.02528	-2.0416	1.01639	1.0251	-2.0416	1.01657
1	4	1.01585	-2.01425	0.99867	1.01572	-2.01425	0.998793
1	5	1.0085	-1.98263	0.975193	1.00878	-1.98263	0.974918
1	6	1.02592	-1.98085	0.959183	1.02642	-1.98085	0.958681
1	7	1.06738	-1.98212	0.931847	1.06743	-1.98212	0.931794
1	8	1.13345	-1.92759	0.862133	1.13405	-1.92759	0.861532
1	9	1.28812	-1.74367	0.725292	1.28439	-1.74367	0.729029
1	10	1.41786	-0.914551	0.411242	1.51631	-0.914551	0.312796
1	11	2.43202	-0.0932748	0.4179	2.70745	-0.427257	0.476453
2	1	0.880237	0	0	1	0	0
2	2	1.03405	-2.05917	1.02516	1.03393	-2.05917	1.02528
2	3	1.02536	-2.04177	1.01647	1.02518	-2.04177	1.01665
2	4	1.01588	-2.0143	0.998699	1.01575	-2.0143	0.998823
2	5	1.00847	-1.98257	0.975162	1.00875	-1.98257	0.974886
2	6	1.02589	-1.98079	0.959156	1.02639	-1.98079	0.958651
2	7	1.06737	-1.98212	0.931845	1.06743	-1.98212	0.931792
2	8	1.13344	-1.92757	0.862126	1.13405	-1.92757	0.861523
2	9	1.28816	-1.74371	0.725304	1.28441	-1.74371	0.729056
2	10	1.41717	-0.914225	0.411274	1.51598	-0.914225	0.31247

Figure 15: JUCE Plug-In

REFERENCES

[1] Jamilla Balint et al. "Bayesian decay time estimation in a reverberation chamber for absorption measurements." In: *The Journal of the Acoustical Society of America* 146 3 (2019), p. 1641. URL: https://api.semanticscholar.org/CorpusID:203923814.

[2] Thibaut Carpentier, Markus Noisternig, and Olivier Warusfel. "Hybrid Reverberation Processor with Perceptual Control". In: *International Conference on Digital Audio Effects*. 2014. URL: https://api.semanticscholar.org/CorpusID:12728324.

[3] Alexis Favrot and Christof Faller. "Complementary N-Band IIR Filterbank Based on 2-Band Complementary Filters". In: 2010. URL: https://api.semanticscholar.org/CorpusID:259106136.

[4] Alexis Favrot and Christof Faller. "Designing Sets of N Doubly Complementary IIR Filters". In: *Journal of The Audio Engineering Society* (2011). URL: https://api.semanticscholar.org/CorpusID:108673591.

[5] Georg Götz et al. "Neural network for multi-exponential sound energy decay analysis". In: *The Journal of the Acoustical Society of America* 152 2 (2022), p. 942. URL: https://api.semanticscholar.org/CorpusID:248887595.

[6] Jean-Marc Jot, Laurent Cerveau, and Olivier Warusfel. "Analysis and Synthesis of Room Reverberation Based on a Statistical Time-Frequency Model". In: *Journal of The Audio Engineering Society* (1997). URL: https://api.semanticscholar.org/CorpusID:14686355.

[7] Eric A. Lehmann and Anders M. Johansson. "Prediction of energy decay in room impulse responses simulated with an image-source model." In: *The Journal of the Acoustical Society of America* 124 1 (2008), pp. 269–77. URL: https://api.semanticscholar.org/CorpusID:20701623.

[8] Karolina Prawda. "FLEXIBLE REAL-TIME REVERBERATION SYNTHESIS WITH ACCURATE PARAMETER CONTROL". In: 2020. URL: https://api.semanticscholar.org/CorpusID:226310273.

[9] Karolina Prawda, Vesa Välimäki, and Sebastian J. Schlecht. "Improved Reverberation Time Control for Feedback Delay Networks". In: 2019. URL: https://api.semanticscholar.org/CorpusID:208342905.

[10] Robin Scheibler, Eric Bezzam, and Ivan Dokmanić. "Pyroomacoustics: A Python Package for Audio Room Simulation and Array Processing Algorithms". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2017), pp. 351–355. URL: https://api.semanticscholar.org/CorpusID:11855208.

[11] Sebastian J. Schlecht. "FDNTB: THE FEEDBACK DELAY NETWORK TOOLBOX". In: 2020. URL: https://api.semanticscholar.org/CorpusID:226313046.

APPENDIX 4

INTRODUCTION

To bridge the gap between Dolby Atmos content[2][3] and stereo music content[4], we developed a stereo upmix algorithm for Li Auto's cockpit (up to 21 speakers).

In the beginning, we used a passive decoder approach inspired by Dressler's work[3] to map decoded signals to speakers. However, passive decoding does not effectively separate sources within the stereo signal, motivating an "active" decoder.

Frequency-domain upmix approaches using mid-side decomposition[8] or coherence/similarity measures[1] produce favorable results but are computationally expensive due to FFT/IFFT and prone to artifacts. Time-domain filter banks[9], such as QMF[5] or Complementary AllPass Filters[4][5], offer an efficient alternative. Dolby Pro Logic II[3] exemplifies this approach; Gundry[6] describes how steering via feedback servo systems, comparators, level measurement, and VCAs can be combined with a filter bank plus matrix gain/delay at the output stage to achieve an immersive surround experience.

[2] https://professional.dolby.com/music/dolby-atmos-for-cars/li-auto/#gref
[3] https://professional.dolby.com/music/dolby-atmos-for-cars/nio/#gref
[4] https://support.apple.com/HT212182
[5] https://ccrma.stanford.edu/~jos/sasp/Quadrature_Mirror_Filters_QMF.html

To improve efficiency, the IIR coefficients of a Butterworth filter are decomposed into coupled all-pass sections[6]. Aligning the filter bank's center frequencies to the mel scale[10] was verified through internal subjective testing to improve perceived quality.

A forgetting factor[7] was initially used to smooth the panning parameter, but caused audible glitches under rapid steering changes. It was replaced with an IIR filter, which supports a lower cutoff frequency than a single-coefficient filter.

Three VCA strategies are supported: Linear (See ALG 1), Square-Law (See ALG 2), and Sine-Law[2] (See ALG 3). Vectorization is enabled by eliminating control statements and guarding against divide-by-zero errors.

In the signal flow diagram (See FIG 16), solid lines represent mono-channel signals and dashed lines represent multichannel signals, where channel count matches the Butterworth filter bank's total subband count. Multichannel outputs from each decoder are summed via a mixer (Σ). Delay units and FDN are incorporated to enhance reverberation and expand the surround field to additional speaker positions such as the roof bar. A secondary decoder extracts information from both the raw input and primary decoder output, following Vickers' strategy[11] for simultaneous multi-decoder operation.

Matrix gain and matrix delay (See FIG 17) provide final signal routing and parameter presets —fidelity, surround, dynamic, and more —within the cockpit environment.

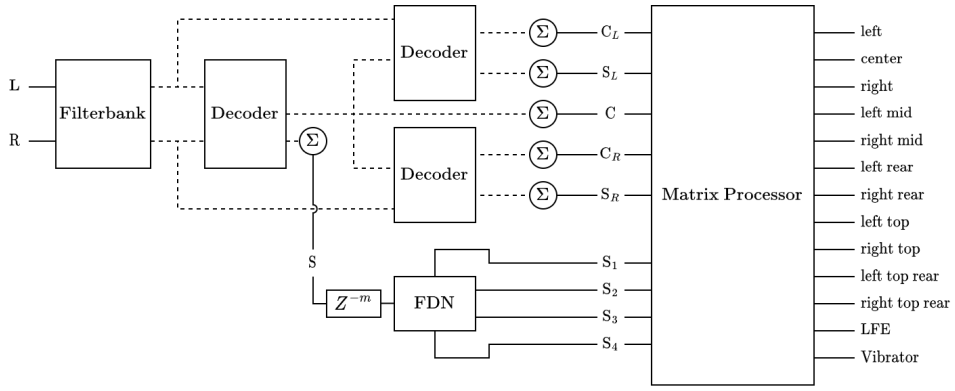


Figure 16: Diagram of Proposed Approach

		Input Channel										
		L	C	R	C_L	C_R	S_L	S_R	S_1	S_2	S_3	S_4
Pre-Output Channel	left	0dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB
	center	− inf dB	0dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB
	right	− inf dB	− inf dB	0dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB
	left mid	− inf dB	− inf dB	− inf dB	0dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB
	right mid	− inf dB	− inf dB	− inf dB	− inf dB	0dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB
	left rear	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	0dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB
	right rear	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	0dB	− inf dB	− inf dB	− inf dB	− inf dB
	left top	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	0dB	− inf dB	− inf dB	− inf dB
	right top	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	0dB	− inf dB	− inf dB
	left top rear	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	0dB	− inf dB
	right top rear	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	0dB
	LFE	0dB	0dB	0dB	0dB	0dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB
	Vibrator	0dB	0dB	0dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB	− inf dB

Figure 17: Matrix Gain, and Matrix Delay can be implemented using similar approach.

[6] `https://ww2.mathworks.cn/help/dsp/ref/ca2tf.html`

Algorithm 1 Extract Center And Surround: Linear

Input: L, R
Output: C, S

Initialize:
    $P \leftarrow 0.5$
function SURROUND(L, R)
    $V_L \leftarrow L \cdot P$
    $V_R \leftarrow R \cdot (1 - P)$
    $C \leftarrow V_L + V_R$
    $S \leftarrow V_L - V_R$
    $F \leftarrow |V_L| * |V_R| > 0$
    $P \leftarrow (1 - F) \cdot \frac{|V_R|}{|V_L|+|V_R|} + F \cdot P$
    $P \leftarrow \text{SmoothProcess}(P)$
    return C, S
end function

Algorithm 2 Extract Center And Surround: Square-Law

Input: L, R
Output: C, S

Initialize:
    $P \leftarrow 0.5$
function SURROUND(L, R)
    $V_L \leftarrow L \cdot \sqrt{P}$
    $V_R \leftarrow R \cdot \sqrt{(1 - P)}$
    $C \leftarrow V_L + V_R$
    $S \leftarrow V_L - V_R$
    $F \leftarrow |V_L| * |V_R| > 0$
    $P \leftarrow (1 - F) \cdot \frac{|V_R|}{|V_L|+|V_R|} + F \cdot P$
    $P \leftarrow \text{SmoothProcess}(P)$
    return C, S
end function

Algorithm 3 Extract Center And Surround: Sine-Law

Input: L, R
Output: C, S

Initialize:
    $P \leftarrow 0.5$
function SURROUND(L, R)
    $V_L \leftarrow L_k \cdot \sin\left(\frac{\pi}{2} \cdot P\right)$
    $V_R \leftarrow R_k \cdot \sin\left(\frac{\pi}{2} \cdot (1 - P)\right)$
    $C \leftarrow V_L + V_R$
    $S \leftarrow V_L - V_R$
    $F \leftarrow |V_L| * |V_R| > 0$
    $P \leftarrow (1 - F) \cdot \frac{|V_R|}{|V_L|+|V_R|} + F \cdot P$
    $P \leftarrow \text{SmoothProcess}(P)$
    return C, S
end function

REFERENCES

[1] Carlos Avendaño and Jean-Marc Jot. "Frequency Domain Techniques for Stereo to Multichannel Upmix". In: 2002. URL: https://api.semanticscholar.org/CorpusID:54586220.

[2] H. A. M. Clark, Gilbert F. Dutton, and P. B. Vanderlyn. "The "Stereosonic" recording and reproducing system". In: *Ire Transactions on Audio* (1957), pp. 96–111. URL: https://api.semanticscholar.org/CorpusID:96470231.

[3] Roger Dressler. "Dolby Surround Pro Logic II Decoder Principles of Operation". In: 2000. URL: https://api.semanticscholar.org/CorpusID:115234803.

[4] Alexis Favrot and Christof Faller. "Complementary N-Band IIR Filterbank Based on 2-Band Complementary Filters". In: 2010. URL: https://api.semanticscholar.org/CorpusID:259106136.

[5] Alexis Favrot and Christof Faller. "Designing Sets of N Doubly Complementary IIR Filters". In: *Journal of The Audio Engineering Society* (2011). URL: https://api.semanticscholar.org/CorpusID:108673591.

[6] Kenneth James Gundry. "A New Active Matrix Decoder for Surround Sound". In: 2001. URL: https://api.semanticscholar.org/CorpusID:62028664.

[7] Emmanuel C. Ifeachor et al. "Digital Signal Processing: A Practical Approach". In: 1993. URL: https://api.semanticscholar.org/CorpusID:54140500.

[8] Sebastian Kraft and Udo Zölzer. "STEREO SIGNAL SEPARATION AND UPMIXING BY MID-SIDE DECOMPOSITION IN THE FREQUENCY-DOMAIN". In: 2015. URL: https://api.semanticscholar.org/CorpusID:201052389.

[9] Sebastian Kraft and Udo Zölzer. "TIME-DOMAIN IMPLEMENTATION OF A STEREO TO SURROUND SOUND UPMIX ALGORITHM". In: 2016. URL: https://api.semanticscholar.org/CorpusID:140119537.

[10] S. Umesh, L. Cohen, and D. Nelson. "Fitting the Mel scale". In: *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*. Vol. 1. 1999, 217–220 vol.1. DOI: 10.1109/ICASSP.1999.758101.

[11] Earl Vickers. "Frequency-Domain Two- To Three-Channel Upmix for Center Channel Derivation and Speech Enhancement". In: *Journal of The Audio Engineering Society* (2009). URL: https://api.semanticscholar.org/CorpusID:15520279.

APPENDIX 5

INTRODUCTION

Lava Music is a company that creates a smartguitar, with a body made of fiber. This smartguitar has both a piezo and an acoustic actuator along with features like a tuner, metronome, sheet, effects, loopers, and educational applications.

However, as our company grows and expands its scope we face challenges in software development, embedded systems design, and DSP development. These challenges arise due to the need for refactoring in our software infrastructure.

To begin addressing these challenges, we require a mechanism to handle audio and message events separately. Audio events involve aspects like Serial Ports (SPORT), chained direct memory access (DMA) interrupts, interrupt handlers, and audio callback functions. Message events involve SPI and UART communication between peripherals like flash memory, SDRAMs, and SOC.

In our initial framework implementation, we have linked the handling of audio events, with message events. However, this approach can potentially lead to longer processing times than necessary(See FIG 18). If the processing time exceeds the frame size of the callback function[7] audible glitches may occur.

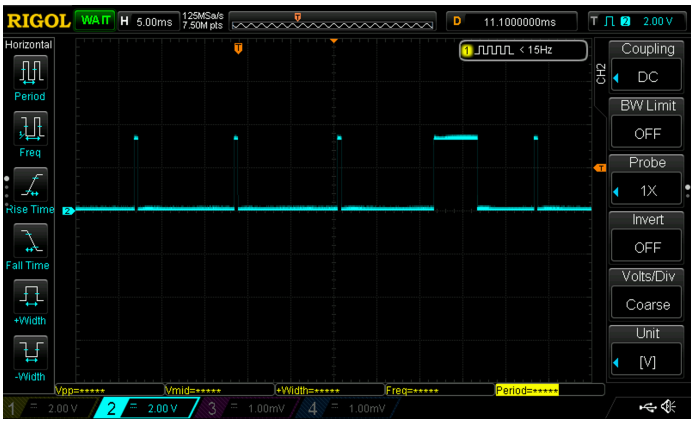


Figure 18: Fluctuating Audio Frame Time

```
int delay_set(int index, int value)
{
    switch(index)
    {
        case ID_subdivision:
            break;
        case ID_tempo:
            break;
        case ID_feedback:
            break;
        case ID_mix:
            break;
        case ID_highcut:
            break;
        case ID_lowcut:
            break;
    }
}
```

Figure 19: Manual Implementation

Secondly, the development of DSP is complicated by the absence of a conventional structure. This means that DSP developers need to handle algorithms and memory usage through syntax cautiously. Another issue we face is the implementation of the control interface(See FIG 19). This practice slows down the delivery process and reduces the reusability of algorithms. Moreover, while static memory management is advantageous, in situations it restricts our ability to cascade audio algorithms, such as using two distortions or multi-band EQ derived from a single parent object.

Lastly, the upgrade function is closely tied to the user application making it more challenging to manage both audio events and message events. Ideally, a bootloader could select an application from flash memory as an approach to task management[1]. Additionally, in this environment, we can create a calibration application that records data from the factory onto flash memory. This recorded data can then be utilized by the corresponding calibration algorithm to ensure output across products. To address these challenges mentioned above we have developed a solution known as "REMORA"[8](See FIG 20).

[7] https://www.youtube.com/watch?v=SJXGSJ6Zoro
[8] https://github.com/kweiwen/remora

We have devised a mechanism that separates the process function, from the interrupt handler. The handler that manages the flag(See FIG21) uses a built-in type called volatile[9] in C. To prevent any issues we have implemented a function called ProcessingTooLong(). In this function, if the program exceeds a timing threshold it enters a while loop. Sets a hardware GPIO to indicate its current behavior. This approach allows us to test combinations of algorithms as well as user stories. The flag serves as an indicator of whether the corresponding function(See FIG 22) is locked or not. Additionally, we ensure that interrupts must; (i) Not update any global data, such as errno. (ii) Not write to (or maintain) any private static data, thereby run-time function is interrupt-safe. In the task loop, we can use a while loop with an if-else statement(See FIG 24) to handle each process function based on its associated flag. This method effectively separates each handler from its process function.

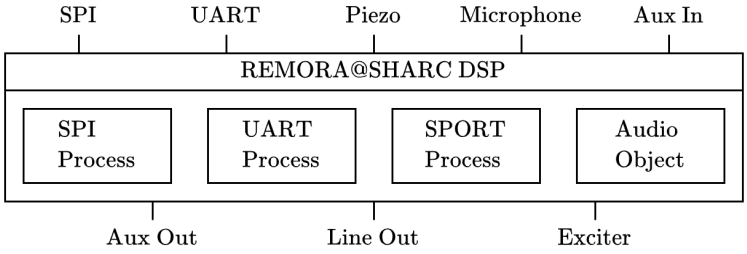


Figure 20: REMORA

```
// SPORT1_isr.cpp
void ProcessingTooLong(void)
{
    while(1);
}

void TalkThroughISR(uint32_t iid, void*
    handlerArg)
{
    if(isProcessing)
    {
        ProcessingTooLong();
    }
    /* Increment the block pointer */
    buffer_cntr++;
    buffer_cntr %= 2;
    inputReady = 1;
}
```

Figure 21: Thread Function

```
// blockProcess_audio.cpp
void handleCodecData(unsigned int
    blockIndex)
{
    /* Clear the block ready semaphore */
    inputReady = 0;
    /* Set the processing active semaphore
        before starting processing */
    isProcessing = 1;

    /* Float, ADC data from codec */
    floatData(output, input, index,
        blockSize);

    /* Place the audio processing algorithm
        here. */
    process_audioBlocks();

    /* Fix, DAC data to codec */
    fixData(output, input, index, blockSize
        );

    /* Clear the processing active
        semaphore after processing is
        complete */
    isProcessing = 0;
}
```

Figure 22: Process Function

By configuring the SPORT, SPI and UART registers the DSP can handle interrupt events. For example, if both SPORT and SPI are interrupted at the time SPORT will be prioritized first followed by SPI and UART.

In the while loop, it is crucial to arrange each process in an order. This allows us to measure the time taken for each process by using an oscilloscope to examine the output of a GPIO. By measuring the processing time, for

[9] https://en.wikipedia.org/wiki/Volatile_(computer_programming)

message events, we can determine the processing time for audio events in critical situations. These situations involve combinations of algorithms and large control message chunks. There are methods to minimize the processing time for message events, such as dividing messages into blocks of size and using checksums on the DSP side to confirm message completion. At the time we aim to maximize the processing time for audio events. This helps us make use of computational power since DSPs are more expensive than general SOC or MCU processors. As a result, we achieve decoupling of SPORT, SPI, and UART tasks(See FIG 23).

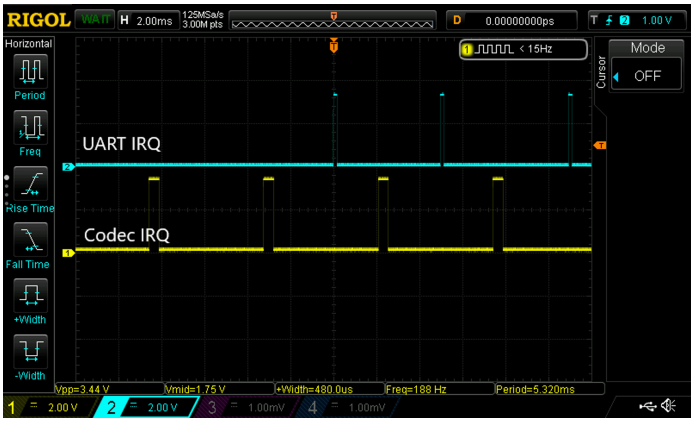


Figure 23: Independent Thread

```
// main.cpp
void process_loop(void)
{
    volatile int count = 0;
    while(1)
    {
        if(inputReady)
        {
            count++;
            handleCodecData(buffer_cntr);
        }
        else if(UARTCommandReady)
        {
            handleUARTCommandData(
                uart_buffer);
        }
        else if(SPICommandReady)
        {
            handleSPICommandData(spi_buffer
                );
        }
    }
}
```

Figure 24: Process Function

A boot kernel or bootloader is a program that loads an operating system's kernel into memory on a DSP. It starts running after powering on the DSP. Ensures that the main operating system is correctly loaded into internal memory. The bootloader should be stored in volatile memory like flash memory so that it can be executed during system startup.

As mentioned in the introduction, the boot kernel implementation allows for applications(See FIG 25). One of these applications is the Upgrade Application, which belongs to App1 in Sector_0. Its main task is to erase flash memory from Sector_2, to Sector_n and write OTA data into it. This enables the upgrading of App2, App3 and so up to AppN.

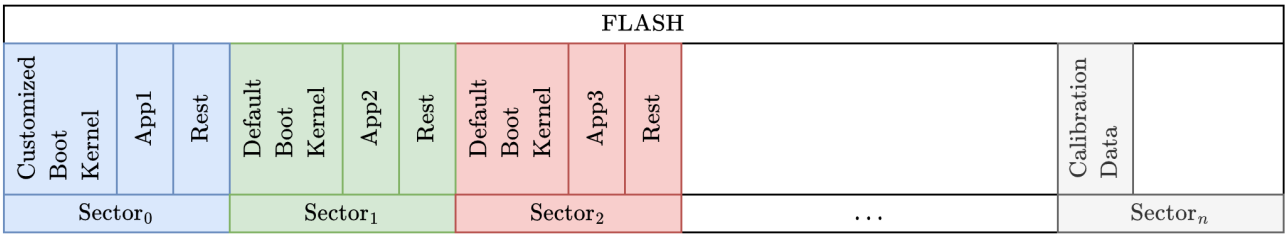


Figure 25: Multiple Applications

The Calibration Application belonging to App2 in Sector_1 writes data of acoustic components into flash memory. By using the IEEE-754[2] standard we can store floating point numbers as 4-byte sequences in memory. When needed this data can be easily. Converted back into its floating point representation. It acts as a calibration reference to compensate for variations during the assembly process, fiber body, piezo, and actuator. Additionally, the Calibration Application provides a talk-through function that allows the manufacturing department to carry out testing plans during assembly procedures. This helps maintain product quality during mass production.

The User Application(See FIG 26) belonging to App3 in Sector_2 is considered the most crucial application. It enables users to customize their effect rack by combining effects. And we create a Dynamic Pool(See FIG 26) that acts as a container, for managing algorithms. Additionally, we maintain a lookup table that keeps track of the computational power required by each audio algorithm. On the other hand, the UI application, which runs on an Android-supported SOC refers to this lookup table. Deactivates certain running processes if a specific combination exceeds the maximum computational power.

The User Application also offers background music (BGM) features. The Aux In(See FIG 26) represents the source of the background music stored in Android. Furthermore, users can enable looper function by manipulating recorded data within Android through the Aux Out(See FIG 26) providing them with performance options. Moreover, an educational application utilizes an on-set detection algorithm(See FIG 26) to evaluate users' playing performance and measure time consumption. In addition, the User Application supports four performance modes.

General Mode: The actuator's sound blends with the acoustic sound.

Enhanced Mode: This mode bypasses the DSP effects while keeping the actuator in operation.

Line Out Mode: This mode turns off the actuator when there is detection of a line output receptacle ring.

Acoustic Mode: This mode turns off both the line output and actuator. However, the functionality of recording, playback, and file rendering remains available.

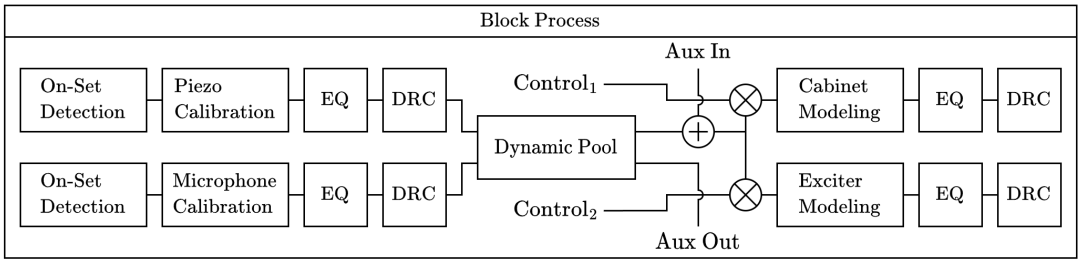


Figure 26: User Application

As mentioned earlier, we have chosen to implement an **AudioObject**(See FIG 27) in C++ instead of C. This decision simplifies managing structures in C and allows for integration between different derived classes. Additionally, we have incorporated an **AudioProcessor Class** to JUCEs implementation[10]. This abstract class defines behaviors like setting and retrieving parameters, and memory management. We can integrate derived classes into a single container based on their shared abstract class. Furthermore, by implementing functions in abstract classes, we ensure that DSP developers fully comprehend crucial behaviors and meet all requirements before successfully compiling the source code.

During the process of SPI or UART command execution we have the ability to control parameters by utilizing the index of the **Derived Instances** within the container and the index of parameters related to those **Derived Instances**. It is worth noting that even if the SPORT remains active we can effectively manage and control the initialization and release of each **Derived Instance** within the container. This means that any audio-related processes will **remain unaffected on-the-fly**. By carefully handling procedures such as fade-in and fade-out as well as managing initialization and release of **Derived Instances**. We can arrange new combinations **without causing memory leaks** or **introducing audible glitches**.

Furthermore, in our setup, we have implemented heaps within external memory using CrossCore Embedded Studio[11] (CCES) from Analog Devices. Each heap is organized as an array specifically designed for storing delay lines. When a **Derived Class** is instantiated it can be assigned to a heap index. This approach ensures that each individual **Derived Instance** which may consume an amount of memory resources has its dedicated access, to an independent heap. Such efficient memory management optimizes our existing setup while also reducing hardware costs.

[10] `https://github.com/juceframework/JUCE`

[11] `https://www.analog.com/en/designcenter/evaluationhardwareandsoftware/software/adswtcces.html`

```
// AudioObject.h
class AudioObject
{
public:
        AudioObject(unsigned int size)
        {
                if(size == 0)
                {
                        ParameterSize = 0;
                        LocalParameter = NULL;
                }
                else
                {
                        ParameterSize = size;
                        LocalParameter = new AudioParameter * [ParameterSize];
                }
        }

        virtual ~AudioObject()
        {
                Release();
        }

        void PrepareToProcess()
        {
        }

        virtual void Process(float* buffer, uint32_t audio_block_size) = 0;
        virtual void Reset() = 0;
        virtual void Release();
        void ParameterCtrl(unsigned int index, unsigned int value);
        unsigned int ParameterDisp(unsigned int index);

protected:
        void addParameter(AudioParameter* parameter);

private:
        unsigned int ParameterSize;
        AudioParameter** LocalParameter;
};

void AudioObject::addParameter(AudioParameter* parameter)
{
        int index = RemoraUtilities::HexsToInt(parameter->paramID);
        assert(index >= 0);
        assert(index < (int)ParameterSize);
        LocalParameter[index] = parameter;
}

void AudioObject::ParameterCtrl(unsigned int index, unsigned int value)
{
        *(LocalParameter[index]) = value;
}

unsigned int AudioObject::ParameterDisp(unsigned int index)
{
        return LocalParameter[index]->cc_value;
}

void AudioObject::Release()
{
        delete[] LocalParameter;
        LocalParameter = NULL;
}
```

Figure 27: Abstract Class of Audio Object

REFERENCES

[1] Mitesh Moonat. "Boot Kernel Customization and Firmware Upgradeability on SHARC® Processors". In: 2009. URL: https://api.semanticscholar.org/CorpusID:62752201.

[2] Dan Zuras et al. "IEEE Standard for Floating-Point Arithmetic". In: 2008. URL: https://api.semanticscholar.org/CorpusID:61507968.

APPENDIX 6

INTRODUCTION

Analog Devices and DSP Concepts are both prominent companies in the audio industry, providing efficient solutions for developing, tuning, and encryption DSP solutions. Their applications such as SigmaDSP[12] & SigmaStudio, and Audio Weaver[13] & AWE Designer, utilize an on-chip solution and graphical user interface (GUI) to accelerate the development of audio products.

This business model aligns well with Tymphany's strong background in supply chain management, manufacturing, and engineering. As a result, we have developed our own solution: **flowStudio**(See FIG 28), a GUI that supports x86 architectures, and **flowEngine**, a DSP library compatible with mainstream embedded systems including Airoha, AMLogic, Nordic, Raspberry Pi, SHARC DSP, STM32, and others. This innovative solution has already been embraced by leading companies like **Bose**, **Google**, **Jabra**, **LG**, **Marshall**, **NTT**, and **Rivian**.

Our solution enables acoustic and DSP engineers to design algorithms that lower the difficulty or develop complete audio solutions without the skill for programming or the need to align with hardware development, thereby increasing development efficiency. It also enables different companies to collaborate with disclosure by sharing the compiled project files.

Once the processors are connected by the flash programmer, one can control the parameters of the algorithm on-the-fly. Also, this control interface can be accessed by external devices, such as embedded systems. This flexibility enhances the deployment of an audio solution and supports a variety of user stories.

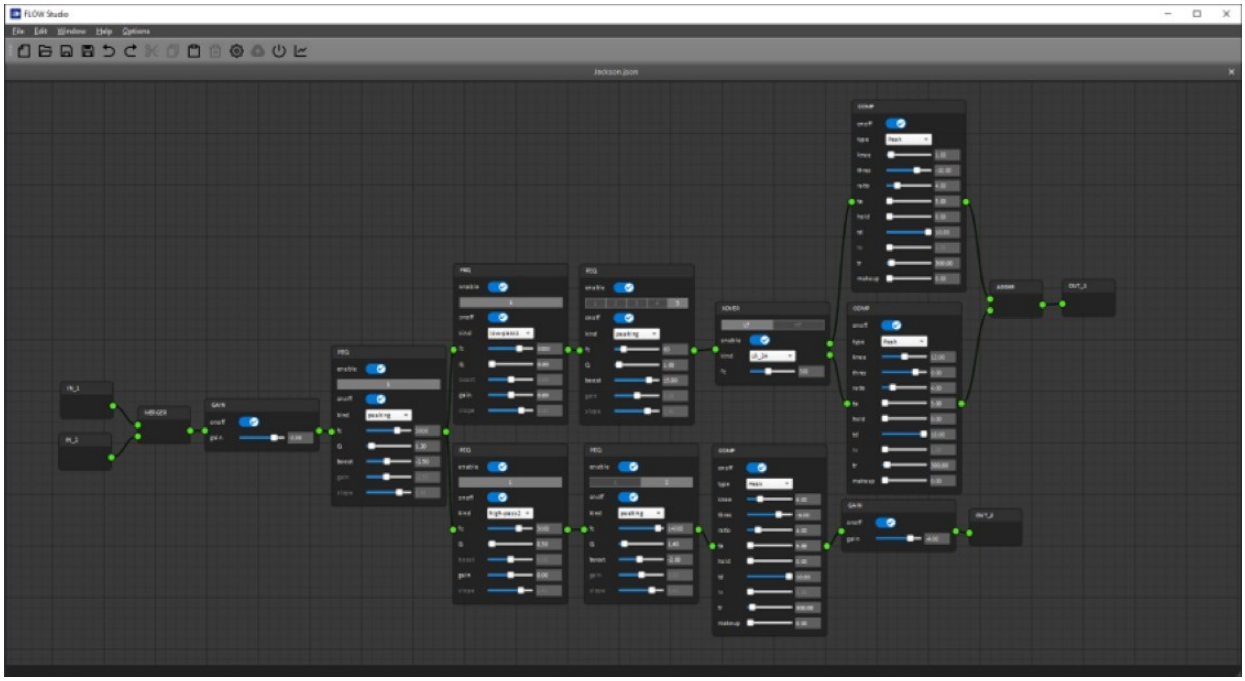


Figure 28: **flowStudio** Editor Window

[12] https://www.analog.com/en/design-center/evaluation-hardware-and-software/software/ss_sigst_02.html
[13] https://w.dspconcepts.com/audio-weaver

flowStudio, a software application developed using PySide2[14] and PyQtNodeEditor[15], offers an editor similar to Max/MSP[16] and Pure Data[17]. Users can define audio processing flow by creating or removing nodes and the insertion or deletion of edges. Basic editing operations such as copy, paste, cut, redo, and undo are also implemented. It supports the control of signal flow and algorithm parameters while connecting to the **flowEngine**. In addition, activating the timer within the widget in the GUI enables real-time data monitoring inside the nodes of the DSP library, such as spectrograms or volume meters.

flowEngine, a DSP library incorporating the C language algorithms in a functional programming style[1], offering the capability to be compiled on multiple platforms from x86 machine to embedded system. This library handles memory initialization, memory release, audio processing, parameter setting, and getting, etc.

In **flowEngine**, we also utilize a directed acyclic graph (DAG)[18] data structure to manage audio processing flow, where algorithm instances are treated as nodes and connections between algorithms are considered edges. This DAG structure ensures the creation of necessary buffers and organizes the audio processing flow for each buffer.

When **flowStudio** is connected to **flowEngine** deployed on an x86 machine, it utilizes the PortAudio API[19] to manage audio input and output, communicating via TCP/IP. Conversely, when connected to **flowEngine** deployed on an MCU/DSP, it handles audio input and output based on the codec and hardware configuration, communicating via UART.

There are three operational Modes in **flowEngine** and **flowStudio**: Offline Mode(See FIG 29), Online Mode(See FIG 30), and Deployment Mode(See FIG 31).

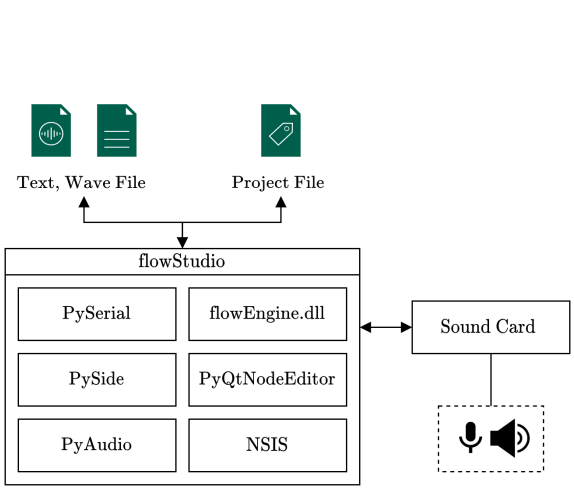


Figure 29: Offline Mode

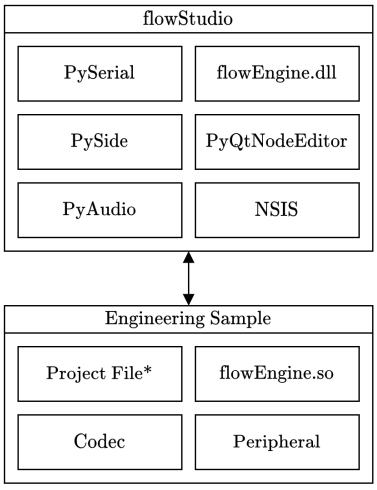


Figure 30: Online Mode

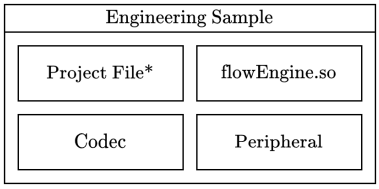


Figure 31: Deployment Mode

Offline Mode(See FIG29), provides the ability to simulate an algorithm flow within a laptop:

- **flowEngine** is compiled into a .dll format, allowing Cython[20] in **flowStudio** to handle the dynamic library loading.
- Read and write features enable recording of simulation results in .txt or .wav formats. allowing users to analyze their algorithm flow before deployment on the embedded systems.
- ADC and DAC node offer real-time audio input/output capabilities.

Online Mode(See FIG 30), provides the ability to tune engineering samples from a laptop:

[14] https://wiki.qt.io/Qt_for_Python/GettingStarted
[15] https://gitlab.com/pavel.krupala/pyqt-node-editor
[16] https://cycling74.com/
[17] https://puredata.info/
[18] https://github.com/RustAudio/dasp
[19] http://www.portaudio.com/
[20] https://cython.org/

- **flowEngine** is pre-compiled into a .a format, enabling MCU/DSP compilers to incorporate this static library.
- If the MCU/DSP lacks a file-based system, the project file is converted into a series of commands to initialize the algorithm flow and its parameters.
- This mode requires a flash programmer to be connected to the engineering samples for controlling the algorithm flow and its parameters.

Deployment Mode(See FIG 31), provides the ability to ship product:

- MCU/DSP can manipulate the parameters using the private control interface.
- It allows clients to customize user stories, for instance:
  - MUC collects the data of the environment from sensors, such as temperature and humidity, and the **calibration node** in **flowEngine** computes the parameters that adapt to the current environment.
  - The **BPM node** in **flowEngine** calculates the BPM of current sources. It allows MCU to manipulate the animation of the LED ring on the outer side of the speaker.
  - The **KWD node** in **flowEngine** recognizes a particular keyword from an input voice, and it will trigger virtual assistants in the SOC.

REFERENCES

[1] Axel-Tobias Schreiner. "Object oriented programming with ANSI-C". In: 1993. URL: https://api.semanticscholar.org/CorpusID:118634402.