Aurras: Active Environmental Noise Cancellation Using an FPGA

Matthew Caren
Massachusetts Institute of Technology
Cambridge, MA, USA
mcaren@mit.edu

Bryan Jangeesingh
Massachusetts Institute of Technology
Cambridge, MA, USA
brytech@mit.edu

Jonas Rajagopal
Massachusetts Institute of Technology
Cambridge, MA, USA
jrajagop@mit.edu

***Abstract*—There exist many systems capable of actively reducing outside noise in closed, acoustically-optimized systems, such as headphones and car interiors. However, few systems are capable of reducing noise in arbitrary unfamiliar environments. We introduce a dynamic noise cancellation system capable of significantly reducing environmental noise in any space using a Field-Programmable Gate Array (FPGA). The system is able to calibrate its behavior to the time-domain and frequency-domain characteristics of the space using an impulse. Then, it monitors the environment to ascertain the requisite output from a speaker near the user to destructively interfere with the incoming pressure wave.**

I. Overview

Active noise cancellation is a technique that attenuates sound by introducing a secondary sound source designed to neutralize the original source. Ideally, this secondary source emits a sound wave (an "anti-noise signal") that is the exact inverse of the incoming wave (the "noise signal"), thus achieving complete destructive interference with the original pressure wave. In systems where the secondary source is co-located with the user, such as active noise-canceling headphones, the influence of the surrounding environment can be reasonably ignored, since the point of detection, the point of cancellation, and the point of perception are all very close together. However, this approach relies on positioning sound sources extremely close to the user's ears and is typically effective for only one user. We developed a more versatile system capable of canceling single-source noise in a variety of environments. The system achieves this by measuring and adapting to the acoustic characteristics of the surrounding environment.

The physical setup, as detailed in Fig. 1, consists of an input microphone, output speaker, and feedback microphone, as well as an additional speaker-microphone pair for calibration. All computation is performed on an AMD Spartan-7 FPGA clocked at 98.3MHz, running 24kHz 16-bit audio.

The speaker/microphone pairs are configured colinearly and are fixed at a distance of 34.5cm from each other. To ensure consistency, we constructed an MDF housing that secures the speakers, microphone, audio amplifier, FPGA, and all necessary wiring. A perfboard layout manages signal wiring to the board, and matches the desired pin configuration for each microphone and associated components. To counteract the feedback behavior between the microphone and speaker output, acoustic decoupling foam is used to mount the microphone to the wooden enclosure—the system is capable of a stable output of up to 88 dB of sound intensity output before feedback.

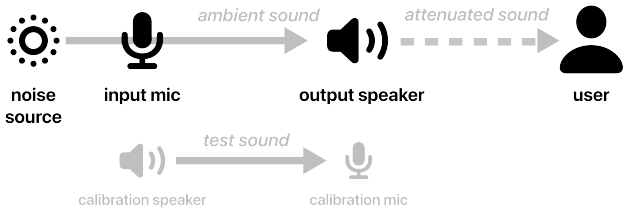

Fig. 1: Overview of physical system layout

The system has two modes: a core noise cancellation mode and a room-adjusted mode. The core noise cancellation mode operates independently of the characteristics of the room. In this mode, the input microphone signal is first preprocessed with a DC-blocker, anti-aliasing filter, and decimator from 48kHz to 24kHz. This signal is then delayed by the time-of-flight for a sound pressure wave to travel from the system microphone to the system speaker. Finally, an inverted (i.e. negated) version of the signal is outputted, to destructively interfere with the original noise.

The room-adjusted mode builds on this by taking into account the environment. In this mode, the microphone input is similarly preprocessed, but the signal is then convolved with a measured impulse response of the room to account for the time-domain and frequency-domain behavior of the surroundings. The signal is then again delayed and negated. Hardware switches on the FPGA board configure the state of the system.

All code is stored in a public GitHub, which can be accessed at *https://github.com/matthewcaren/aurras*.

II. Input

A. Reading Data From Microphone

The microphones used are MEMS microphones[1] interfaced via the I2S protocol over the onboard pmod pins. The microphones are natively driven at a 48kHz sampling rate and output 18-bit samples, which are down-sampled on arrival to 16 bits

[1]https://www.adafruit.com/product/3421

to match the rest of the system. An I2S manager module is responsible for both deserializing the incoming bitstream, as well as managing the outgoing clock and word-select lines. I2S requires a clock to be sent to the microphone at 64 times the sampling rate, where the first 32 cycles correspond to the first channel and the latter 32 to the second channel. For each channel, the first 18 bits represent the 18-bit audio, and the latter 14 bits are not used. In the system, the data line is clocked at 3.072MHz. The channel select pin is grounded since the two microphones in the system operate separately. The word-select pin operates 64-times slower than the data clock and crucially changes on the falling edge of the data clock. The power supply and ground take up the last two pins. A 100kΩ pull-down resistor removes any capacitance that has built up.

B. DC-Blocker

The MEMS microphones used tend to have significant DC offsets. All microphones tested had a baseline of approximately -900 (a normal-volume conversation would produce samples on the order of $100-300$). This baseline is generally very stable and does not drift, but must be addressed to prevent unexpected output behavior, as well as to prevent intractably large DC components in the downstream convolution stage (which will effectively square the DC offset).

To correct for the offset, a module computes the average offset over 2^{15} microphone samples ($\sim 0.68s$) by taking a running sum of the inputs, and then right-shifting the result by 15 bits. All audio samples are thereafter shifted by the offset to neutralize the DC component. This is controlled by `BTN1` on the board.

C. Anti-Aliasing Filter

The microphone is sampled at 48kHz, however, the system audio runs at a sampling rate of 24kHz. To downsample without aliasing, we precede decimation with an anti-aliasing lowpass filter with a cutoff at 11500Hz. The filter was built using Vivado's FIR Compiler IP with coefficients designed via the Parks-McClellan algorithm to have a ripple of less than 0.4dB in the passband.

D. Decimation

Lastly, the audio is decimated from a sampling rate of 48kHz to a sampling rate of 24kHz. Since the input has already been antialiased, we can simply take every other sample from the anti-aliasing filter output.

Once the above steps are complete, the raw input from the Adafruit microphone has now been converted into 16-bit, zero-centered, downsampled 24kHz audio.

III. Delay Module

The delay module is used to align the computed response of the input signal at the microphone (the "anti-noise signal") with the incoming original noise. Without introducing this delay, the input-to-output latency would be just a few clock cycles on the FPGA, and the output of the anti-noise signal would preempt the arrival of the corresponding noise signal by almost the entire time-of-flight from the input microphone to the output speaker.

The delay module's design leverages the inherent capabilities of the Block RAM (BRAM) of the FPGA. We configure the BRAM in dual-port mode, which offers concurrent access through two independent sets of data and address lines, and therefore simultaneous read and write operations.

We treat the BRAM as a circular buffer. The delay module accepts samples in real-time from the input microphone, which are immediately written into memory. Simultaneously, an access pointer offset by a specified delay value reads from the BRAM and outputs the corresponding sample. The delay parameter is calculated to be the desired delay time (measured in audio-clock cycles) offset by the latency introduced by the memory operations to achieve sample-correct accuracy. It can be adjusted in real-time to modify the time interval between the original sound and its delayed reproduction.

In the system, the delay value is set to be equal to the time-of-flight measured by using the distance between the input microphone and output speaker, 34.5cm, and dividing by the speed of sound in air, giving a flight time of 1.01ms, which corresponds to 24 audio samples. The delay is programmed using `SW15-SW10` on the board.

IV. Room-Adjusted Noise Cancellation

This set of modules is only used when the system is in "room-adjustment" mode. The high-level objective is to anticipate how the input sound will interact with the surroundings, and incorporate this into the predicted 'anti-noise' sound. Since each room responds differently, the system must be calibrated each time it is used in a new location.

A. Impulse Response

To measure how a sound behaves in a space, an impulse is played from the calibration speaker, and the resulting impulse response (IR) is recorded using the calibration microphone. This provides a model of the transfer function of the space.

First, an impulse generator module outputs an impulse from the speaker by bringing the output from 0 to a large value for a few cycles and then pulling it back down to 0. It also generates internal trigger signals to achieve sample-correct timing alignment with other modules.

After the impulse is played, the system waits for the number of cycles specified by the delay module to account for the time it takes the impulse to travel from the calibration speaker to the calibration microphone. Then, a 1-second IR (which is sufficient for most common spaces) is recorded from the calibration microphone and stored in the IR memory buffer subsystem. The IR generation and measurement process is triggered by `BTN3` on the board.

B. Convolution Outline

This process involves performing mathematical convolution of the audio input with the measured IR. This requires an audio input buffer of equal size to the IR buffer, so we maintain a

running buffer of the last 24000 audio input samples. Then, in the time limited by the audio sampling period, we perform the required multiplications and additions: each audio sample, denoted as the ith sample, is multiplied with the corresponding IR value indexed at $23999 - i$, followed by a sum of all the resulting values. In other words, the convolutional result at any given timestep is the dot product of the input buffer with the reversed IR buffer. Because the audio sample rate is 24 kHz and the system is clocked at 98.3 MHz, each sampling period contains 4096 clock cycles, in which all 24000 required multiply-add operations must be completed. This mandates a more complex approach than simply iterating and completing one multiply-add per cycle. However, the constraints of the dual-port memory preclude parallelizing more than two computation streams, so this demands a more complex memory system.

C. IR Memory Management

To meet the timing constraints, we designed a memory architecture for storing the IR buffer using 4 Xilinx Dual-Port memory buffers operating in unison, allowing for 8 reads per clock cycle. Each sample of the IR is written into memory only once in reverse order to enable homogenous indexing of both buffers in the convolution logic. Each buffer stores a contiguous $1/4$ of the IR data, where the first of the four buffers stores the last 6000 values and so on. This allows for a streamlined reading process: in each cycle, two adjacent indices are read from each of the four buffers, with the buffers being accessed sequentially. This method allows the system to read the entire set of 24000 IR values in 3000 clock cycles, comfortably less than the 4096 cycles available within the audio sampling period. This design not only optimizes memory access, but also aligns with the system's processing capabilities and time constraints.

D. Audio Buffer Memory Management

To successfully perform the convolution within the requisite period, our audio buffer must also support 8 read operations per clock cycle. This design is inherently more complex than the IR memory, as it requires updating with each new audio sample. From an algorithmic perspective, every time we receive a new sample it becomes the last value in the buffer, and we shift each existing value by one position to make room for it (discarding the oldest sample in the process). Reading and rewriting all 24000 values upon receiving every audio sample is infeasible, so we must use a more efficient strategy.

The audio buffer, similar to the IR memory, is segmented into four sub-buffers, each containing 6000 consecutive audio samples. Buffer 3 represented the newest 6000 audio samples and buffer 0 represented the oldest 6000 of the 24000 samples. However, now we maintain a pointer to the most recent value and treat the memory block as a circular buffer, where the newest value of each buffer is located at the pointer and the oldest at the index just after it, wrapping around the end of the buffer. For example, if the pointer is at 2500, then index 2500 is where the new value gets added. Indices 2501 to 5999 of

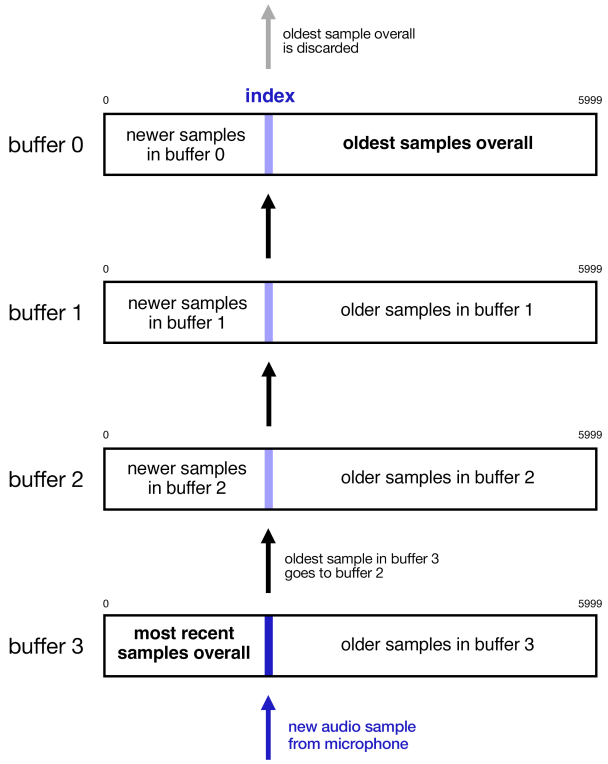

Fig. 2: Diagram of the live audio buffer memory management structure

buffer 3 represent the oldest 3499 samples of the third audio buffer. Index 2501 will be overwritten in the next cycle, and the value in its location will become the newest index of the next buffer up. Index 2499 contains the audio sample which was received one cycle previously.

This means that when a new sample is received, the value in the spot of the pointer in buffer 3 is transferred to the same index in buffer 2. This represents moving what was the last sample from the third buffer - the most recent 6000 samples - into the buffer representing the audio samples from 12000 cycles ago to 6000 cycles ago. Similarly, the previous value at the pointer index in buffer 2 is written to the same spot in buffer, and so on as the previous samples 'cascade' upwards. When the first buffer is reached, the sample from the pointer index is removed from the buffer since this is the audio sample from 24000 cycles ago. Once the convolution is complete, the pointer is incremented by 1. This process is outlined in Fig. 2.

This structure requires only three reads and four writes when the system receives a new audio sample while still keeping track of the 24000 most recent audio samples. It also enables more streamlined convolution computation, since two samples can be read at a time from each buffer starting at the index one after the pointer, since we start convolving with the oldest values.

E. Convolution Module

Using the memory structures outlined above, the convolution process becomes much more simplified. Each time the system receives a new audio input, the module spends roughly 3000 cycles to compute the convolved output. Since the impulse buffer is written in reverse order, the convolution becomes the dot product of the two stored buffers with the audio buffer starting at one more than the index of the pointer - the oldest value in the buffer. Each cycle, we read 8 addresses from the audio buffer and 8 addresses from the IR buffer. Two adjacent indices are read from each buffer. Three cycles later, we receive the corresponding values, where we multiply them and add each result to one of eight running intermediate sums. This summing structure enables more efficient operations (since adding the 8 values from a single cycle together cannot be synthesized in parallel). This process is then run with eight multiply-add blocks completely in parallel for the 3000 cycles. Once all 24000 multiplies have been computed, the module spends seven more cycles summing the eight intermediate sums to produce a final convolved output. We take the 13th to 28th bits of this result as the 16-bit output to maintain a signal magnitude similar to the input audio (these values can be tweaked, but were experimentally found to be effective, and a similar effect can be achieved by changing the gain of the amplifier).

F. Convolution Example

Let's say the system is in a state where the IR has already been recorded and the audio buffer pointer is at 2500. First, the 2500th element of buffers 3, 2, and 1 are read. Then these values are shifted "up" a buffer, so they now represent the 2500th elements of buffers 2, 1, and 0, respectively. The new audio sample becomes element 2500 in buffer 3, and the pointer indicates index 1000 is the start index. This is represented by the upward arrows in 2.

The convolution iterates from oldest to newest, so the elements from indices 2501 and 2502 of each audio buffer are multiplied by the elements from indices 0 and 1 of the corresponding IR buffer. Each of these eight products is added to a separate intermediate sum. Then, each of the indices is incremented by 2 and this repeats until the 2499th and 2500th element of each audio buffer is multiplied with the 5998th and 5999th element of the impulse buffers. Then, the eight intermediate values are summed, producing the result of the convolution, and the audio buffer pointer is incremented by 1. The system waits for the next audio sample and repeats the same process.

V. Output

To output the anti-noise pressure wave, we are using 4-inch full-range speakers[2] driven by a low-latency stereo digital amplifier (speaker selection is performed by simply writing to the left or right channel). The output signal is synthesized through pulse-density modulation (PDM) via an onboard Delta-Sigma modulator at 6MHz. The calibration speaker solely outputs the impulse and the system speaker outputs either an anti-noise signal (for either operating mode), a pass-through signal from either microphone or a test tone depending on the configuration of `SW2-SW8`.

We use a first-order PDM to synthesize audio. We noticed that an undesirable chirping noise was present if the DC component was close to zero. To combat this, we offset the output by a constant value to avoid this noise. A higher-order PDM may also fix this issue.

[2]https://www.daytonaudio.com/product/1534/tcp115-4-4-poly-cone-midbass-woofer-4-ohm

A. Output Phase Correction

The phase responses of speakers are highly nonlinear, which must be accounted for to achieve successful destructive interference with the sound source. This is accomplished with an all-pass FIR filter that estimates the inverse of the phase response of the speaker. Finding these FIR filter coefficients is highly non-trivial; an extensive discussion of this problem and the developed solution is included in Appendix A.

VI. System Evaluation

The system runs on the provided RealDigital FPGA. The synthesized result meets timing with a slack time of 3.283ns (compared to the clock period of 10.17ns, it meets timing by a significant margin). The longest propagation delay for the logic portion of a single cycle is only 3.856ns. The system uses only 13 of the 150 DSP blocks and 5.28% of the LUTs on the board. The system does use a considerable amount of memory, using 50 out of 75 of the on-board BRAMs. However, about 1/3 of the memory used by the system is used for a debugging module which outputs audio delayed by a full second, which could eventually be removed if the system requires more memory.

Considering these metrics, the system could feasibly run at an audio rate of 48kHz. This would require twice the memory usage since both buffers would store 48000 samples. The convolution would also have half as many clock cycles between audio samples, 2000 cycles to do 48000 multiplications. So, it would need to do four times as many operations, 32 multiplies per convolution cycle instead of 8. We believe this is possible, but it would be pushing the limit of the onboard memory.

VII. Performance

We measured the system in the controlled environment of an acoustically-treated studio. Our setup consisted of a static sound source placed opposite a Shure SM57 dynamic microphone, with the system placed collinearly in between. In each trial, we played a constant sound and measured a 10-second average baseline sound pressure level from the microphone. Then, we engaged our system and measured the resulting steady-state attenuation. The results for the level reduction of sinusoidal test tones by frequency are cataloged in Fig. 3. The overall level reduction of the system over the frequency range of 100Hz–3000Hz (measured with the same setup) is plotted in Fig. 4. We were able to consistently reduce

Frequency (Hz)	Level Reduction (dB)	Acoustic Power Reduction
50	0.0	0.0%
100	1.5	29.2%
200	7.7	83.0%
300	7.6	82.6%
400	4.4	63.7%
500	15.5	97.2%
600	6.2	76.0%
700	7.2	80.9%
800	5.7	73.1%
900	12.1	93.8%
1000	6.5	77.6%
1500	7.1	80.5%
2000	4.4	63.7%
2500	1.0	20.6%
3500	0.0	0.0%

Fig. 3: Measured dead-room noise reduction by frequency

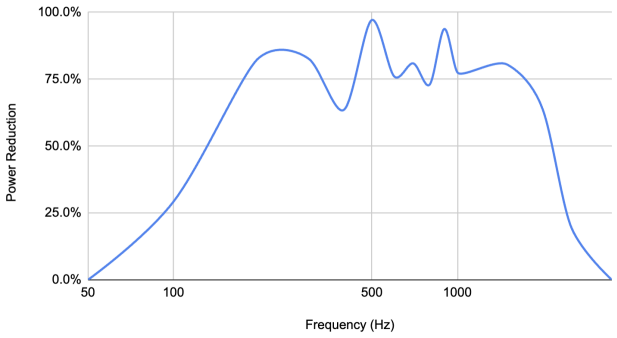

Fig. 4: Plotted dead-room noise reduction by frequency

the acoustic power level of incoming noise by about 75% in the target frequency band of 200–2000Hz, with especially good performance in the "sweet spot" of 500–900 Hz. At peak performance at 500Hz, the system was consistently able to cancel over 15dB of noise despite the speakers adding some noise to the system—which means the system is reducing the power of the sine wave by over 97%.

In a fairly reverberant space (a dorm room with tile floor and windows), we used a sound level meter on a mobile phone as a portable setup to perform a similar test of noise levels with and without our system engaged, and in both modes. Even in the significantly non-ideal conditions, we achieved between 3 and 12 dB noise cancellation at 500Hz in room-adjusted mode. This corresponds to anywhere from a 35% to 93.7% decrease in acoustic power. In many trials, it even performed a few dB better in "room-adjustment" mode compared to the core mode despite the impulse response buffer inducing a lot of noise into the system.

The system's behavior is more inconsistent in room-adjusted mode because the impulse response measurement process adds

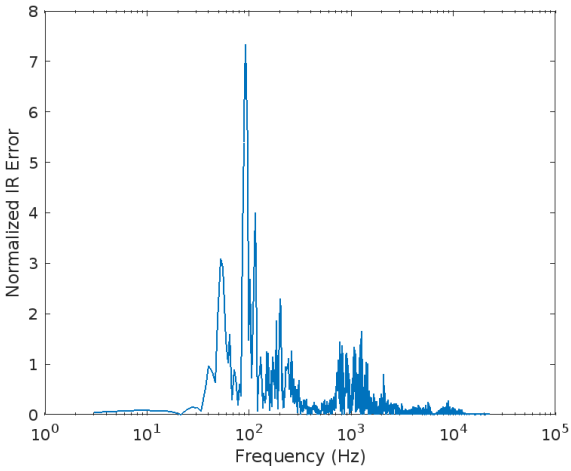

Fig. 5: Frequency-domain error of IR modeling

noise and variability, so getting a good impulse is crucial. It sometimes takes a few impulses to get a sufficiently good measurement. Despite this, the system performs about 5dB better in room-adjusted mode than in its core mode in a room that was not acoustically-treated. This shows that using the IR-based modeling to predict how the input sound will interact with the environment in real-time helps the system cancel more noise.

We also tested in an extremely reverberant space (a large multi-level mezzanine), where performing a similar test in room-adjusted mode yielded approximately 2dB of noise cancellation compared to the baseline. This trend of more reverberation resulting in less ideal cancellation behavior is expected, as the acoustic characteristics of a space become increasingly complex and more difficult to accurately model.

To measure the efficacy of the impulse response-based acoustic modeling itself, we measured the system's simulated response to an external impulse and compared it to the actual space's response in the 50–10,000Hz frequency band. The system's simulated response was able to model the frequency-domain characteristics to 94.87% accuracy, with an RMS error between the two normalized frequency responses plotted in Fig. 5. Note the particularly low error in the "sweet spot" of 500–900 Hz.

Though we initially considered performing explicit frequency-domain analysis of the environment with a frequency sweep, the high accuracy of the IR-based modeling rendered it unnecessary.

VIII. Conclusion & Next Steps

Overall, the system operates very effectively in its core noise canceling mode, and achieves significant—though sometimes inconsistent—results in reverberant spaces as well. Using the core noise cancellation mode in an acoustically-treated studio, the system is able to reduce the power by up to 97% and by 75% across a broad range of frequencies. In a reverberant

room, the system is able to reduce the noise level by 3–12dB when in "room-adjustment" mode. The system generally performs better in "room-adjustment" mode when it is in a reverberant room because it is able to predict how the sound will interact with the room and counteract this in the anti-noise output.

With the current architecture, we could still tune the delay to be more precise in order to better align the anti-noise signal and achieve more ideal destructive interference.

One next step we would like to pursue is a tool to visualize the measured IR and cancellation metrics using the HDMI output from the FPGA. This could involve showing the sound before and after cancellation, a temporal diagram of the measured IR of a space, or a frequency-domain plot of either the audio before and after cancellation or the IR itself. While this tool would not increase the system's functionality, it would create an engaging data tool that helps to monitor the status and effectiveness of the system.

Upgrading to higher-quality equipment would also help cancel more of the input noise. With the current setup, the speakers and mics both introduce a significant amount of noise, which not only adds undesirable sound to the system's operation but also contaminates the IR since the recorded impulse response also contains noise. The data shown in 3 would show even larger decibel drops if the speakers did not add additional noise to the system. We could also automatically measure the distance between the system microphone and the system speaker by sending an impulse and measuring the number of cycles until it is received. This would add flexibility, however in the current system, the speaker and microphone are fixed so delay never changes and this is unnecessary.

Given the relatively low memory, LUT, and flip-flop usage, the system could presumably be improved by running the audio at 48kHz and modifying the memory systems to run faster, as well as the convolution algorithm to do 32 multiply-adds per cycle. As mentioned above, this is feasible, but would push the onboard memory to its limit.

The DC-blocker algorithm could also be modified to be time-adaptive instead of being a calibration that happens when turning the system on. This would help if the microphone's baselines drifted, and could be implemented with a one-pole IIR filter. However, we did not find this to be necessary as the microphones have steady baselines.

Lastly, a big upgrade to the system would be the implementation of a self-adjustment mechanism via a feedback microphone next to the user. For example, this could be used to automatically tune the output audio level of the system instead of doing this manually (as currently done). It could also be used to tune the IR algorithm based on what the users hear and even adjust specific frequencies using a gradient-descent algorithm. This would make the system more flexible as it would learn the optimal parameters for operation in real-time.

ACKNOWLEDGMENTS

The authors extend their heartfelt gratitude to the course staff of 6.205 for their invaluable assistance and support. Special thanks go to Joe Steinmeyer, whose guidance during office hours and on Piazza was invaluable. We would also like to express our sincere appreciation to Adam Hartz and Alan Oppenheim for their advice with the design and implementation of the all-pass filter.

This project was a remarkable example of collaboration, where each author contributed significantly to each aspect of the system. Because of the highly integrated nature of the system, each author was equally involved in designing the system.

APPENDIX

A. Allpass FIR Coefficient Calculation

This section deals with the problem of finding FIR coefficients for an arbitrarily-specificied allpass filter; that is, for a specified phase response $\phi(\omega)$, we wish to find a function with the spectrum specified by

$$\hat{H}(\omega) = 1e^{j\phi(\omega)}.$$

We encountered this problem while accounting for the phase response of the speaker, which is non-negligible and does not match any common analytical functions at a glance. The target phase response of our speaker correct (which is the opposite of the phase response of the speaker) is plotted in Fig. 6.

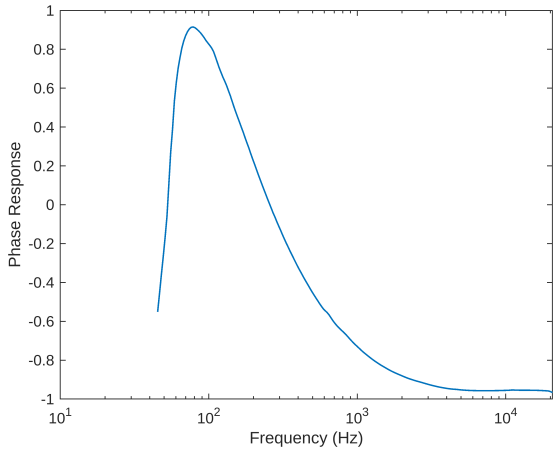

Fig. 6: Target allpass phase response (opposite of measured speaker phase response)

It is intuitively tempting to approach this problem by taking inverse discrete Fourier transform of the desired response. However, this does not result in the desired results, as it forces a linearly increasing phase and matches spectrum coefficients by phase-wrapping (i.e. phasing the input signal by a full 2π radians between each matched phase response in a naive

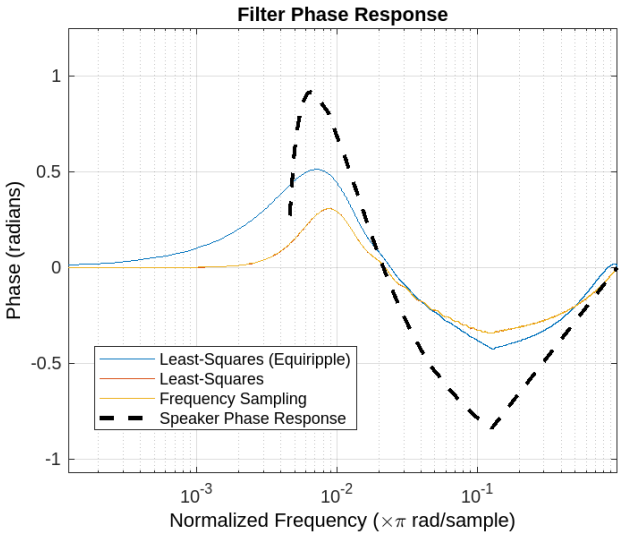

Fig. 7: Comparison of filter phase responses for various coefficient-estimation algorithms

implementation, though more complex methods can be used to avoid this, usually introducing other artifacts in the process).[3]

The Parks-McClellan algorithm, which is often used to design real FIR filters, offers an analytical solution for coefficients for frequency-domain responses of stopbands and passbands. It is efficient and widely implemented, but strictly produces linear-phase results, which are also ill-suited for this purpose.

It turns out that no one has yet discovered a satisfactory analytical solution for this problem, so in applications it is solved on a case-by-case basis. One approach is to apply the Remez exchange algorithm by separating the complex problem into two real ones—this is essentially an extension of the Parks-McClellan algorithm into complex filters—but it suffers from several failure modes and only solves a constrained subset of the problem space. In our specific application, the multiple zero-crossings in the target response made it infeasible.

The best general solution, which is what we used to find the coefficients for the filter in our system, is by using parameter-estimation algorithms like least-squares—where parameters are the FIR coefficients and the cost function operates over the magnitude and phase of the frequency responses.

At this point, we should also note the secondary consideration of transient preservation, which we would also like to optimize in the design of our filter. Because we are introducing a significant amount of nonlinear-phase distortion, signal components in different frequency bands are delayed by different amounts, causing them to be pushed out of alignment with each other and "smearing" sharp transients. While this is inconsequential in some applications, this was an important consideration in our system, as too much phase misalignment would result in the system "missing" transients and creating unpleasant sharp noises at the onset of sounds.

This results in a trade-off between transient behavior and correcting of the phase response of the speaker. The error function we used encourages optimization towards both the inverse phase response (full phase correction) and perfect transient behavior (zero phase / group delay) equally, and is weighted by a factor of 6 in our target frequency band of 200-2000Hz.

A comparison of the filter responses generated with this least-squares process, a nearly-identical filter with added coefficients to encourage equiripple behavior, a filter designed via filter sampling, and the actual phase response of the speaker is given in Fig. 7. We ended up using the least-squares result, which provides a good balance between effective phase-correction and transient preservation.

B. Block Diagram

See Fig. 8 on the following page.

[3]It's worth noting that the *frequency sampling* method for FIR filter design does something similar to this by estimating the response for specific frequency buckets without any sort of explicit optimization—its result is included in Fig. 7, but was not optimal for our case.

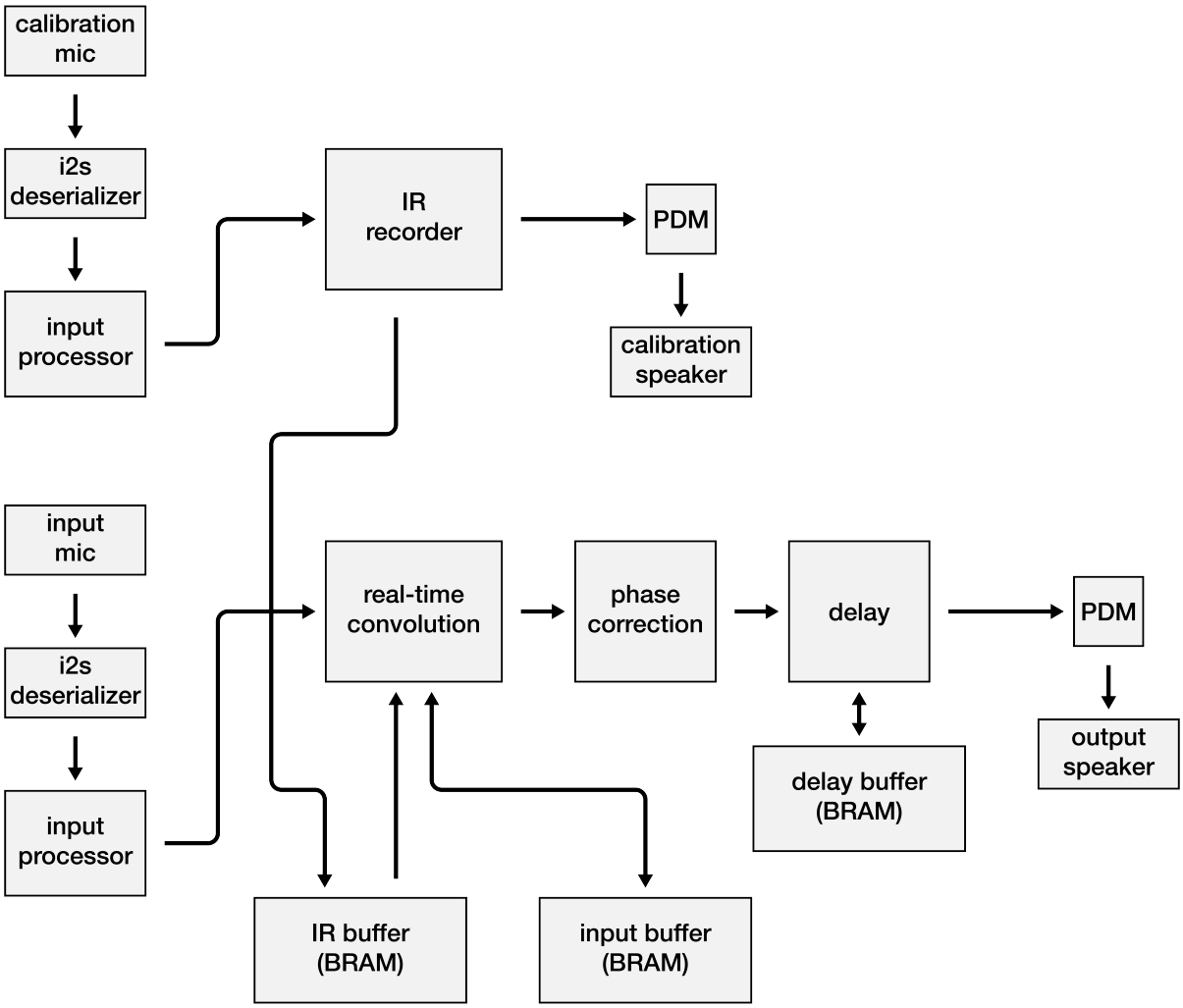

Fig. 8: System block diagram