EAVESDROP - Enhanced Audio Voice Extraction and Selective Detection for Remote Observation and Processing

Samvit Das
Dept. of Electrical Engineering & Computer Science
Massachusetts Institute of Technology
Cambridge, MA
samvitd@mit.edu

Anand John
Dept. of Electrical Engineering & Computer Science
Massachusetts Institute of Technology
Cambridge, MA
anandj@mit.edu

***Abstract*—Clear listening in a specified direction is a highly desirable feature of microphone systems, especially in noisy environments such as conference rooms and lecture halls. The tight time constraints for real-time processing of rich audio suggest an FPGA-based implementation of a beamforming array audio processing algorithm to spatially filter a signal. In this work, we propose EAVESDROP, an end-to-end beamforming audio processing pipeline that simultaneously isolates a directional audio signal and tracks the motion of an audio source. In particular, we develop and implement a PCB microphone array design and a filtering and tracking pipeline built on an Artix-7 FPGA development board.**

***Index Terms*—Digital systems, Beamforming, Signal processing, Field programmable gate arrays, Audio systems**

I. Introduction

Beamforming and direction-of-arrival (DoA) are well-studied problems in the field of signal processing. A variety of algorithms exist and achieve high performance especially in frequency-specific RF applications, but extensive work has also been done to adapt these algorithms for generic-frequency audio processing. These adaptations have achieved success in speaker diarization, voice recognition in vehicles, and long-range listening in noisy environments. In this paper, we demonstrate that the Generalized Sidelobe Canceller model of beamforming can be implemented in a compute-efficient manner to spatially filter audio in the human vocal range of frequency (100 - 4000 Hz), and provide a simple algorithm to update the state estimate of a speaker's position in real-time. We implement our design on a Field Programmable Gate Array (FPGA). An FPGA is ideal for this time sensitive problem because it is able to process the inputs of several microphones in parallel and optimize the limited set of operations necessary for most beamforming algorithms.

II. Preliminaries

Beamforming algorithms exploit a signal's dependence on the relative position of the transmitter and the receiver. For this reason, most beamforming devices consist of an array of N spaced receivers.

6.205 Digital Systems Laboratory, Massachusetts Institute of Technology

A. Time Delay of Arrival

The phase picked up by a signal with wave-vector $\vec{k}$ over a displacement $\vec{r}$ is $\Delta\phi = \vec{k} \cdot \vec{r}$. The time delay is then $\tau = \frac{\vec{k}}{\omega} \cdot \vec{r} = \frac{\hat{v}}{v_p} \cdot \vec{r}$, where v_p is the propagation speed of the signal and $\hat{v}$ is the unit vector direction of propagation.

Therefore, for a compact microphone array in which a source can be assumed to be at an equal orientation relative to each microphone, the time delays of a signal at microphone i relative to the origin are given by:

$$\tau_i = \frac{1}{v_p}(\hat{v} \cdot \vec{r}_i) \tag{1}$$

To synchronize the signals $x_i[t]$, we must compensate for these delays:

$$x_i[t - \tau_i] \longrightarrow x_i[t] \tag{2}$$

From this point on, we treat the $x_i[t]$ as though they are appropriately synchronized.

B. Delay and Sum

Assuming prior knowledge of the direction of arrival $\hat{v}$, the simplest beamforming algorithm is to simply apply the appropriate delays to each signal and output a weighted sum of the synchronized signals. This is known as a **delay and sum (DAS)** beamformer.

Let $\mathbf{x}[t] \in \mathbb{R}^N$ be the vector of all (synchronized) microphone signals. The output of the DAS beamformer is:

$$y[t] = \mathbf{w}_0^T \mathbf{x}[t] \tag{3}$$

Where $\mathbf{w}_0 \in \mathbb{R}^N$ is a vector of weights that comprises the filter. The underlying idea of the DAS beamformer is that noise and signals from undesired directions will not be appropriately synchronized and thus are likely to destructively interfere. Therefore, the input will be spatially filtered in the direction $\hat{v}$.

C. Generalized Sidelobe Canceller

A drawback of the DAS beamformer is that it has poor resolution. That is, signals relatively far from the desired direction will still contribute to the beamformer. This makes it especially vulnerable to strong incoming signals from undesired directions.

The **Generalized Sidelobe Canceller (GSC)** beamformer is an adaptive algorithm that applies a time-varying filter $\mathbf{w}[t]$. The idea behind the algorithm is to minimize the signal power P while constraining the filter coefficients to eliminate the trivial solution $\mathbf{w} = \mathbf{0}$. For M_c linear constraints, we can form the constrained optimization problem:

$$\min |\mathbf{w}^T \mathbf{x}[t]|^2 \quad \text{s.t.} \quad \mathbf{C}^T \mathbf{w} = \mathbf{c} \tag{4}$$

Where $\mathbf{C} \in \mathbb{R}^{N \times M_c}$ and $\mathbf{c} \in \mathbb{R}^{M_c}$ define the constraints on the filter coefficients. We may decompose $\mathbf{w} = \mathbf{w}_0 + \mathbf{w}_a[t]$. Where $\mathbf{w}_0$ belongs to the column-space of $\mathbf{C}$ and $\mathbf{w}_a$ belongs to its left null-space, e.g., $\mathbf{C}^T \mathbf{w}_a = 0$. We can then vary $\mathbf{w}_a$ without changing the value of $\mathbf{C}^T \mathbf{w}$ as long as $\mathbf{C}^T \mathbf{w}_0 = \mathbf{c}$. For this reason, we call it the adaptive filter.

Let $\mathbf{B} \in \mathbb{R}^{N \times (N - M_c)}$ be a matrix such that $\mathbf{C}^T \mathbf{B} = \mathbf{0}$. Then for any vector $\mathbf{a} \in \mathbb{R}^{N - M_c}$, $\mathbf{Ba}$ belongs to the left null-space of $\mathbf{C}$. Substituting into Eq. (4), our optimization problem is now:

$$y[t] = (\mathbf{w}_0 + \mathbf{B}\mathbf{a}^*[t])^T \mathbf{x}[t] \tag{5}$$

$$\mathbf{a}^*[t] = \min_{\mathbf{a}[t]} |(\mathbf{w}_0 + \mathbf{B}\mathbf{a}[t])^T \mathbf{x}[t]|^2 \tag{6}$$

Taking the gradient of the objective:

$$\frac{\partial P}{\partial \mathbf{a}[t]} = 2 \cdot \mathbf{B}^T \mathbb{E}\left\{\mathbf{x}[t]\mathbf{x}[t]^T\right\} (\mathbf{w}_0 + \mathbf{B}\mathbf{a}[t]) \tag{7}$$

Setting this equal to zero, we have a closed form solution for the optimum $\mathbf{a}[t]$, or the Wiener solution:

$$\mathbf{a}[t]^* = -\mathbb{E}\left\{\mathbf{x}^T \mathbf{B}\mathbf{B}^T \mathbf{x}\right\}^{-1} \mathbb{E}\left\{\mathbf{w}^T \mathbf{x} \cdot \mathbf{B}^T \mathbf{x}\right\} \tag{8}$$

III. Algorithm Implementation

A. Beamforming

We implement the GSC beamformer algorithm with the following choice of constraints:

$$\mathbf{C}^T = \begin{bmatrix} \mathbf{I}, & \ldots, & \mathbf{I} \end{bmatrix} \tag{9}$$

$$\mathbf{c} = \delta_0 \tag{10}$$

Where $\mathbf{c}$ is a unit impulse. Note that we use an identity operator $\mathbf{I}$ over 1, because we act over the L-sample wide window of the signal. These constraints are motivated by the assumption that we work in free-field propagation. The constraint of $\mathbf{C}$ is motivated by the fact that the desired signal is the same at every microphone since we assume that $x[t]$ is appropriately time delayed. Thus, $\mathbf{C}$ describes the . The unit impulse corresponds to the identity filter since we do not want to distort the signal. We can then choose the fixed beamformer component as:

$$\mathbf{w}_0^T = \frac{1}{N} \begin{bmatrix} \delta_0, & \ldots, & \delta_0 \end{bmatrix} \tag{11}$$

In other words, a simple average over the N components of $\mathbf{x}$.

The blocking matrix $\mathbf{B}$ is not uniquely determined; the choice of the following is a valid solution:

$$\mathbf{B} = \frac{1}{N} \begin{bmatrix} -\mathbf{I} & \cdots & -\mathbf{I} \\ (N-1)\mathbf{I} & \ddots & \vdots \\ \vdots & \ddots & -\mathbf{I} \\ -\mathbf{I} & \cdots & (N-1)\mathbf{I} \end{bmatrix} \tag{12}$$

Note that by our choice, $\mathbf{B} \in \mathbb{R}^{N \times (N-1)}$ and $\mathbf{a}[t] \in \mathbb{R}^{(N-1)}$. The structure of these matrices allows for memoization.

Define $x_{B,i'}[t]$ to be the i-th row of $\mathbf{B}^T x[t]$ for $i \in \{1, \ldots, N-1\}$. Applying the matrix:

$$x_{B,i}[t] = \sum_{j=i+1}^{N} x_j[t] - \mathbf{w}_0^T \mathbf{x}[t] \tag{13}$$

$$x_{B,i}[t] = x_{i+1}[t] - \frac{1}{N} \sum_{j=1}^{N} x_j[t] \tag{14}$$

$$x_{B,i}[t] = x_{i+1}[t] - \mathbf{w}_0^T \mathbf{x}[t] \tag{15}$$

In our implementation, we directly compute the optimal $\mathbf{a}[0]$ upon activation. This requires computing the matrix inverse $\mathbb{E}\left\{\mathbf{x}^T \mathbf{B}\mathbf{B}^T \mathbf{x}\right\}^{-1}$. At all other times, we update our estimate of $\mathbf{a}[t+1]$ by running a gradient descent initialized at $\mathbf{a}[t]$.

B. Direction of Arrival

Upon activation, we require that the steering vector is manually specified by the user. This is done to prevent a computationally expensive search over the entire hemisphere. We expect that most use cases will have priors on the general direction of the desired signal.

At all other times $t > 0$, for estimated direction of arrival $\hat{v}[t]$, the DoA algorithm predicts the next direction as:

$$\hat{v}[t+1] = \text{argmax } P(\hat{u}) \text{ s.t. } |\hat{v} - \hat{u}| \leq \epsilon \tag{16}$$

The idea being that the desired signal's power will be greatest when the directions are perfectly aligned.

After running the GSC beamformer algorithm on the current direction estimate $\hat{v}[t]$, we estimate the power $P(\hat{u})$ by first applying the appropriate time delays and then running a gradient descent to estimate $\mathbf{a}_{\hat{u}}[t]$ for each $\hat{u}$.

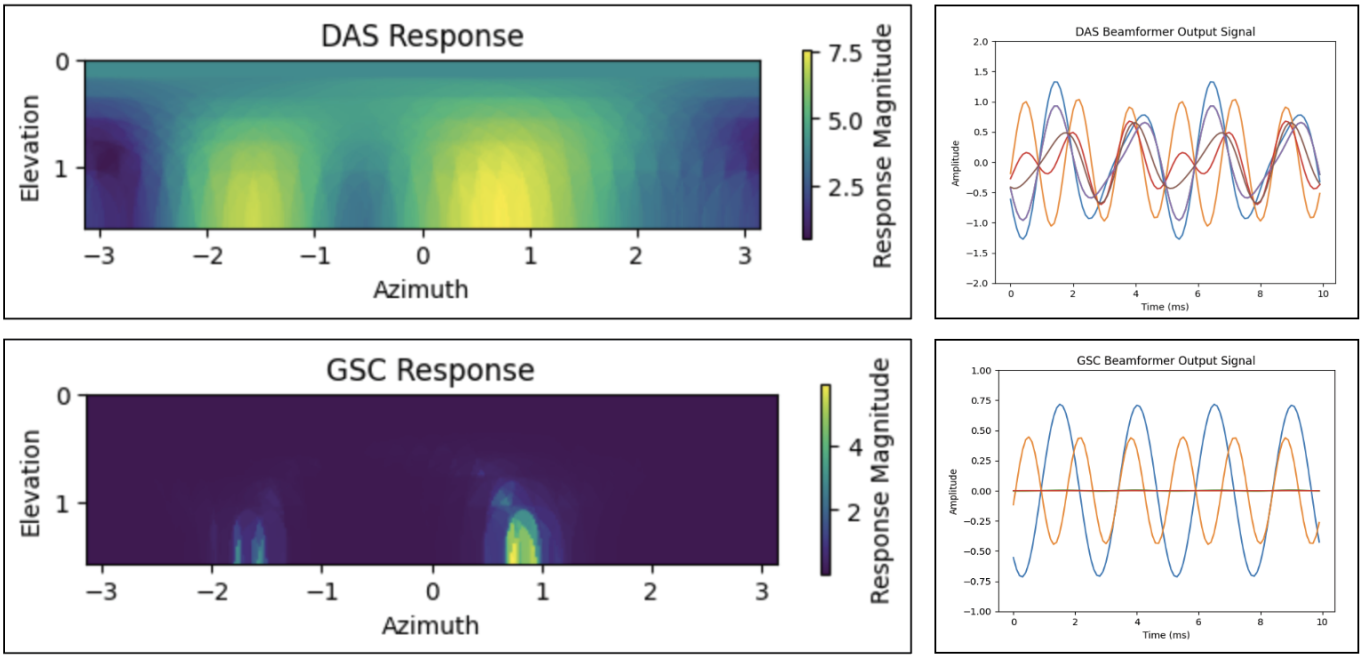

Fig. 1. Resolution tests: sources with amplitude 1, frequencies 400 Hz and 600 Hz placed at azimuths $-\frac{\pi}{2}$ and $\frac{\pi}{4}$ respectively.
Left: Plot of beamformed output power with respect to angular position.
Right: Plot of beamformed output signal at various steering directions. Blue and orange signals correspond to steering directly at placed sources.

C. Algorithm Evaluation

We evaluate our GSC beamformer implementation via a Python simulation. We tested against the baseline of a pure DAS algorithm on the metrics of sensitivity to changes in steering direction and resolution between separate sources.

Figure 1 displays both the improved sensitivity and resolution of the GSC algorithm. The difference between a steering in the direction of a source and a steering in any other direction is much more pronounced. The direct output signals also show much improved suppression of the source signals when the steering is not aligned.

We note that the GSC algorithm tends to suppress the source signals even when the steering is aligned to a lesser degree. This effect is more pronounced as the frequency increases.

IV. Hardware & Architecture Design

The overall device hardware is shown in figure 2 (a) and the architecture is described in the block diagram in figure 2 (b). We will now explain the componenents in more detail.

A. Hardware Overview

The device is centered around the NEXYS A7 FPGA Development board with an Artix A100T FPGA. The board interfaces with a custom built microphone array to gather input as well as via UART to a computer and via I2S to an audio DAC to output it's processed audio.

B. Microphone Array

The microphone array consists of 16 microphones arranged in a circular pattern of diameter 20 cm. Each microphone outputs data over its own separate wire in pulse density modulation (PDM) format at a user-specified clock rate of 3 MHz. The clocks are shared between all the microphones such that all the microphones sample at the exact same instant (almost, half of the microphones sample on the rising edge and the other half on the falling edge of the clock so that they can share a data line). These sampling outputs are connected to the FPGA via the PMOD ports.

C. Microphone Input

The PDM input needs to be low-pass filtered and decimated to determine the actual sound measured by the microphone. Since the microphone can only measure sound accurately up to 10 kHz and this is the upper limit on human speech, the incoming audio will be low-passed down to 10 kHz. More precisely the overall filter will have a pass band till 6 kHz and a stop band starting at 10 kHz.

This is done in the following manner. A cascaded integrator-comb (CIC) filter is used to do the initial low pass filtering and bring the microphone data from 1 bit at 3 MHz to 18-bit at 192 kHz. Then 3 stages of half-band FIR filters decimate it down to 18-bit at 48 kHz. It is then low-pass filtered through a conventional FIR low-pass filter to return the final 16-bit 48 kHz audio that will used in the rest of the processing. There are also additional amplifier stages integrated inbetween the filters controlled by switches on the FPGA to compensate for the low input volume. Since the microphone bases its output signal on its acoustic overload point of 120 dB, and the normal audio the microphone measures is in the 60 - 80 dB range, significant amplification is required to make the signal heard.

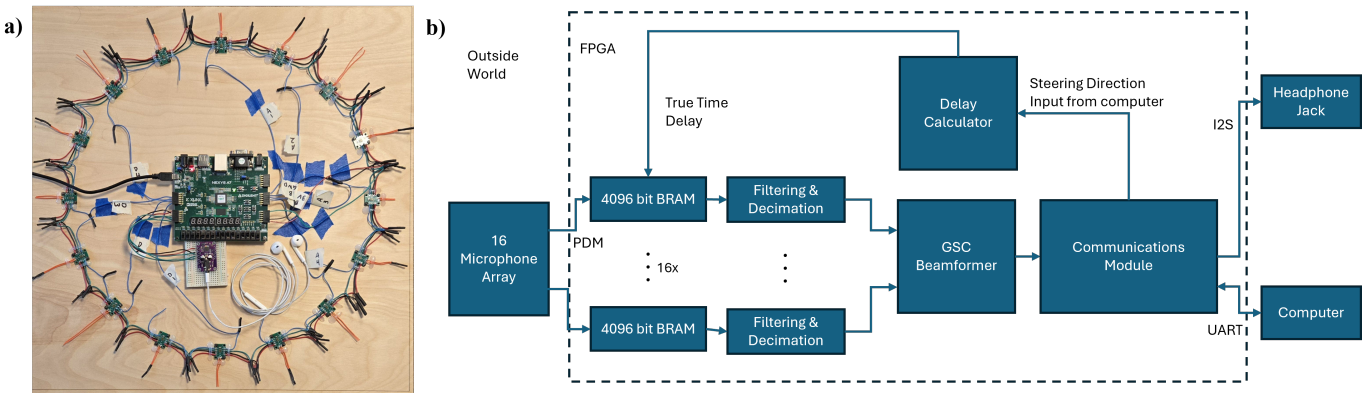

Fig. 2. Final device design. (a) Constructed microphone array with audio jack and UART output. (b) Overarching block diagram

We choose this architecture because processing the raw 1-bit audio is challenging due to the high 3 MHz sample rate. Although the CIC filter has poor stop band attenuation and pass band characteristics, its computational efficiency, especially as compared to more conventional FIR filters, makes it a good choice for this step. In addition, because the output of the filter retains a high frequency (192 kHz), almost all aliased noise falls into the higher frequency band which will then be removed by the better filters later in the pipeline.

The rationale for the choice of the half-wave filter is similar. It has improved performance compared to the CIC filter while being less efficient. However, it is still more efficient compared to a conventional FIR filter and only aliases sound into a frequency band that is removed by the next filter in the pipeline.

The last stage is the conventional FIR filter. This finishes the processing with the result being a clean filtered signal but with a much higher computational efficiency as compared to directly applying a conventional FIR filter to the input. In fact simulations showed that this pipeline resulted in better stop band attenuation and pass band characteristics than simply applying one FIR decimating filter to the input.

D. True Time Delay

Prior to the PDM microphone output being processed by the filtering and decimation step, it is passed through a large BRAM buffer. The index of the element that is outputted from the buffer to the next module is determined based on the direction the device is seeking to beamform in. That is we true time delay the inputs while it's still in the PDM stage. This gives us the advantage of having a very fine resolution on the delays applied to each signal since the PDM is a much higher frequency signal, 3.072 MHz, compared to the audio after the filtering step which is at a much lower 48 kHz. To see the effect this decision had on precision, consider how far sound travels in the distance between two samples. In the case of the 3.072 MHz frequency, it travels $343\ \text{m/s} \cdot 1\text{s}/3.072 \times 10^6 = 0.11\text{mm}$. In the case of the 48 kHz frequency, a similar calculation shows that it travels 7.1 mm. This can be much larger than the distance between some of the microphones for some steering directions and defeats the purpose of the high precision used in the construction of the array.

E. GSC Beamformer

This module implements the GSC beamforming algorithm explained in great detail in section III. Our implementation decouples the gradient descent iteration of the beamforming filter from the collection of data such that it is *always* updating even while waiting for a new audio sample. The pipeline can be described in three steps: computing the audio covariance matrix $\mathbb{E}\{\mathbf{x}[t]\mathbf{x}[t]^T\}$ (in expectation) of the audio data, calculating the gradient and objective with respect to the adaptive element $\mathbf{a}$, and applying the descent step to the current filter value. In parallel, the current estimate of the beamforming filter is applied to incoming data. Under this configuration, there is no need to pipeline the GSC algorithm and the dataflow and unnecessarily slow the optimization process.

To easily store and update the expectation value of the covariance matrix, we store each new vector of audio samples in N FIFO queues with lengths $L = 1024$. This corresponds to a 'window' of 20 ms which is adequate for frequencies $>$ 100 Hz. On a time-step t, we compute the covariance matrix of data entering the queue $\mathbf{x}[t]\mathbf{x}[t]^T$, and leaving the queue $\mathbf{x}[t-L]\mathbf{x}[t-L]^T$. Their difference is then added to a running estimate. The FIFOs are stored in BRAM for their size that is too large for registers but manageable. Each queue uses 1 RAMB18 unit.

Because of the structure of the blocking matrix in Equation 12, its product with a vector is a simple operation – one only needs to compute the average value of the vector and subtract from each entry. To apply the non-trivial matrix vector products of the covariance matrix, we implement a parallelized vector dot product module. This module is used to compute the gradient and power values. Note that the power value is not necessary for the optimization algorithm but is helpful to see during debugging. To reduce resource utilization, we serialized part of the dot product calculation, trading off the fully parallelized latency for a more easily built design.

Ultimately, our calculation pipeline used 138 DSP48E1 blocks with $N = 8$ active microphones.

F. Communications Module & Output

To determine which way to point the beamformer, the communications module listens for input from the computer which communicates with it over UART. This is then set over to the Delay Calculator block which computes the proper true time delay for the input.

After going through all the filtering and beamforming steps, the audio is outputted through 2 channels. One is through UART upon which the audio is saved to the computer. The other is through I2S to a digital analog converter (DAC) which connects to a realtime headphone output. The audio outputted over UART is 14-bit 48 kHz audio at a baud rate of 1,000,000. The reason for 14 bits instead of 16 is that the first two bits are used to indicate which byte is the more significant one which is used to decrease errors in transmission.

V. Design Results & Evaluation

We tested both the DAS and GSC beamforming algorithms on our design. With $N = 16$ DAS beamformer performed about as expected, with a 3 dB difference between a signal received from the steering direction as compared to from $90°$ away. This corresponds to about half of the response magnitude. Comparing to Figure 1, this matches our expectations for the non-adaptive true time delay beamformer.

With $N = 8$, the GSC beamformer unexpectedly returned nearly zero audible output. It is unclear whether this result is due to algorithmic oversights or issues in the hardware pipeline. Upon further review, it appears that the GSC algorithm is known to suffer from target signal cancelation, especially when adapting concurrently with data, due to the inevitable correlation of the target signal with interfering signals (by reverberations, steering error, etc.). [1]. This potentially explains the discrepancy between the realized output and simulation results, where, for example, the ideal source model is incapable of modeling the correlations between target and interference signals by reverberation. The idealized sinusoidal outputs also may be more easily decorrelated than generic voice audio.

Optimizations for the GSC algorithm include a time-variant step-size descent, otherwise known as the implicitly controlled least mean squares algorithm. Further work on this project would explore this idea. However, the resource constraints of the FPGA may make more complex algorithms (involving division) infeasible.

A. Hardware Design Evaluation

Our initial design with $N = 16$ microphones suffered from extreme over-utilization issues. Specifically, Vivado reported a requirement of double the available 63,400 'LUT as Logic' blocks. In addition, 236 of the 240 available DSP48E1 blocks were utilized. We implemented several optimizations to cut down on usage. There were also several timing violations.

Firstly, we pipelined and serialized the dot product computation module. This increased the number of clock cycles per dot product output from 1 to N, but significantly reduced the number of LUT and DSP blocks needed for combinational multiplications and additions. This included both limiting one multiplication per clock cycle and replacing all indexing operations with a simpler bit-shift on each cycle.

We used several to compute average values of filters and in turn find their matrix products with the blocking matrix B. We replaced these with pipelined tree adders, which resulted in further lowered utilization of LUTs as Logic and DSPs, as well as a positive WNS and WHS.

We further pipelined/serialized the calculation expectation value of the covariance matrix on each new data read. Because of the decoupling of the data pipeline and the adaptive filter pipeline, this choice came at effectively no cost to the algorithm.

With the optimizations, the design met utilization and timing requirements for $N = 16$ microphones. However, the compiler was unable to successfully place all shapes onto the FPGA – we suspect that this is because of many large unpacked array registers that are difficult to route together. For example, the covariance matrix and other auxiliary values were stored in $N \times N \times 16$ registers. Though LUT as Memory utilization was not exceeded, the requirements for placement/routing would be difficult to satisfy.

In the interest of time, we opted to reduce the number of microphones to $N = 8$ in order to comfortably satisfy all requirements and successfully build the design. We propose solutions to the mentioned placement issues that would be implemented in further work: rather than using registers for the covariance matrices, store them in BRAM Memory. This can either be stored as a singular N^2 entry deep BRAM, or N distinct N entry deep BRAMs. This will incur a cost proportional to the depth upon every update of the covariance matrix. However, because the time between data reads is infrequent compared to the 100 MHz clock, this comes at little cost.

The matrix-vector product module will need to be restructured to take serial inputs of the rows of the matrix. In this way, the entries are loaded from BRAM and then multiplied with some vector, row-by-row. This also incurs a cost proportional to the depth. Because BRAM is more easily routed than large registers, we expect that this would solve most placement issues.

VI. Author Contribution

A. Individual Contributions

Samvit researched and benchmarked beamforming algorithms in Python and implemented the core algorithm modules for the DAS and GSC algorithms, including the true time delay module, gradient calculation module, and gradient descent module. Most of his work was done test-benching the beamforming algorithm modules and ensuring they matched simulation results. In addition, he also verified the efficacy of

the microphone array design parameters (radius, sample rate, etc.) in simulation.

Anand designed and fabricated most of the microphone array, including the PCB design, microphone choices, and array layout. He developed the filtering pipeline including the CIC and low-pass FIR filters. He also wrote the UART and I2S communication modules in order to interface with the computer and a headphone jack.

VII. Acknowledgements

We would like to thank Justin Htay for guidance in understanding signal processing algorithms and Ben Wang for advice in completing a microphone array beamforming project. We would also like to thank the 6.205 staff, specifically Jan Park for unmatched responsiveness and encouragement with design advice, and Joe Steinmeyer for providing hardware resources and guidance throughout the project.

VIII. Project Code

References

[1] J. Bourgeois and W. Minker, Time-Domain Beamforming and Blind Source Separation. Springer Science & Business Media, 2009.

[2] H. L. Van Trees, Optimum array processing. New York Wiley, 2002.

[3] S. Jin, D. Kim, Hy. Kim, C. Lee, J. Choi and J. Jeon, "Real-time sound source localization system based on FPGA," 2008 6th IEEE International Conference on Industrial Informatics, Daejeon, 2008, pp. 673-677, doi: 10.1109/INDIN.2008.4618186.

[4] T. Verbeure, "Design of a Multi-Stage PDM to PCM Decimation Pipeline," Electronics etc…, Dec. 20, 2020. https://tomverbeure.github.io/2020/12/20/Design-of-a-Multi-Stage-PDM-to-PCM-Decimation-Pipeline.html (accessed Nov. 27, 2024).

[5] R. Lyons, "A Beginner's Guide To Cascaded Integrator-Comb (CIC) Filters", DSP Related.com, https://www.dsprelated.com/showarticle/1337.php (accessed Nov. 27, 2024).

[6] B. Wang, "Phased Array Microphone," Ben Wang's Blog, Feb. 26, 2023. https://benwang.dev/2023/02/26/Phased-Array-Microphone.html (accessed Nov. 27, 2024).