FPGA Beamforming Final Report

Patrick Haertel
Department of EECS
Massachusetts Institute of Technology
Cambridge, Massachusetts
phaertel@mit.edu

Carlos Villa
Department of EECS
Massachusetts Institute of Technology
Cambridge, Massachusetts
villac@mit.edu

Olivia Stoner
Department of EECS
Massachusetts Institute of Technology
Cambridge, Massachusetts
ostoner@mit.edu

***Abstract*—We present the final design of an acoustic beamforming implementation on an FPGA. Beamforming is a signal interference mitigation technique which spatially filters audio sources using a linear microphone array. We accomplish this objective in three stages: (1) audio sampling, (2) modifying sampled audio using the delay-and-sum algorithm, and (3) outputting the modified audio samples. In doing so, we are able to efficiently amplify a signal coming from a chosen integer direction in an 180 degree space while suppressing audio signals coming from other directions.**

***Index Terms*—field programmable gate array, digital systems, acoustic beamforming, delay-and-sum algorithm**

I. Audio Sampling

In order to capture and transmit audio, we use an array of ICS-52000 MEMS Microphones on a self-designed carrier board. The microphone array comprises four microphones separated by 35 cm each. These microphones daisy-chain together as specified by the TDM protocol detailed below. The construction of our board can be seen in Fig. 1. Using the 100 MHz clock, we sample the audio at the frequency nearest 40 kHz that is easily created with clock dividers. We chose a sample rate of 39.0625kHz formed by dividing the 100 MHz clock by 2560. Critically, this divisor is easily divisible by 32, which is key to the TDM communication protocol.

Fig. 1: Physical design of the linear microphone array

A. ICS-52000 MEMS Microphones

We chose to use ICS-52000 MEMS Microphones for our microphone array because they perform signal processing for us by doing the analog-to-digital conversion and filtering the output, and more importantly can sample audio synchronously. Since the microphones do signal processing for us, we can communicate just by following the microphones' communication protocol as defined by the data sheet. The microphones are only available in a surface mount package, so we developed a carrier board to connect with the microphones and arrange them more easily.

B. Microphone Carrier Board

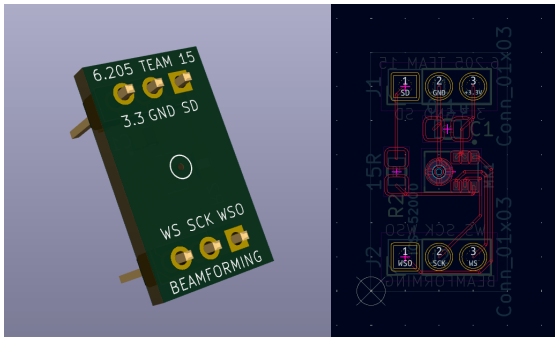

Fig. 2: 3D rendering of the carrier board and PCB layout

The carrier board for the ICS-52000 microphone allows for easy testing and development of our physical microphone array on a breadboard. The carrier board has two 3-pin headers separated by 0.5". This means it fits into a breadboard similar to a DIP IC. The layout, as shown in Figure 2, connects the MEMS microphone to the headers, a bypass capacitor, and an impedance matching resistor. The bypass capacitor (0.15 uF) stabilizes the supply voltage to the microphone. The impedance matching resistor (15 Ohm) attempts to match the output resistance of the data port to that of the transmission line.

C. TDM Communication Protocol

In our system, our FPGA operates as the controller and our microphones act as the peripheral. We implement the Time Division Multiplexing communication protocol to communicate with the ICS-52000 microphones. Time Division Multiplexing, hereby referred to as TDM, is a communication protocol that allows for synchronous sampling and sequential output of these samples defined by a Word Select (WS) selection signal.

We sample audio at 39.0625 kHz which allows us to capture audio well beyond the Nyquist rate of human speech, assuming human speech ranges up to about 8 kHz. To effectively implement beamforming, a sample rate well above the Nyquist rate allows for better signal resolution. This frequency also serves as our system's (WS) frequency, indicating the start of the transmission of audio samples for the microphone array. We chose this sample rate because it covers most human speech and is easily generated by a simple clock

divider, allowing us to stay within the same clock domain. Our controller drives this signal. The microphone array runs on a Serial Clock (SCK), provided by our controller, that follows the formula $n \times 32 \times 39.0625\text{kHz} = \text{SCK}$, where $n =$ power of two equal or greater than # of microphones. Our TDM protocol is parameterized to support different numbers of microphones to facilitate expansion and debugging. For simplicity, our WS and SCK signals are generated via clock dividers.

To receive our samples, the controller raises WS signal high for a full SCK cycle to indicate to the first microphone that it should begin transmitting data for 32 bits. The current microphone then raises the Word Select Out (WSO) flag to indicate to the next microphone that it is their turn to send data. This process continues for the number of microphones in the system. We only sample the first 24 bits of each 32 bit transmission, because the microphones only send 24 bit two's complement audio in MSB format. Our samples are then processed by our beamforming logic before output.

II. Beamforming Logic

A. Delay-and-sum Algorithm

We implement the delay-and-sum algorithm in our beamforming logic [1]. This equation calculates the time delay necessary to shift each microphone's audio signals to isolate the sound from the intended direction. The following equation models this approach:

$$\Delta\tau_n = \frac{n * d * cos(\theta)}{c} \tag{1}$$

- $\Delta\tau_n$: time delay
- n: microphone index
- d: distance between microphones
- θ: angle to focus beam at
- c: speed of sound

We use zero-indexing for the microphone index. The distance between microphones is measured in meters, and the angle of focus is entered in degrees. We use 343 m/s for the speed of sound. This equation can be used in conjunction with clock frequency to determine the number of samples delay to wait before sending audio stored in mic BRAMs, which is the approach we used in our system.

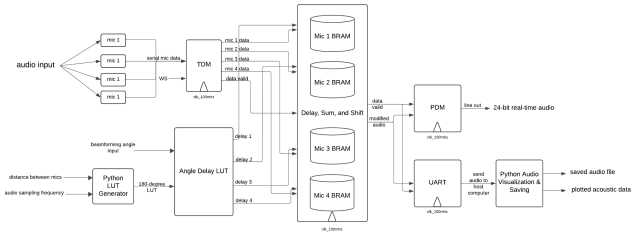

Fig. 3: Block diagram of beamforming implementation

B. Implementation

Fig. 3 provides a high level overview of the implementation of the beamformer. The TDM input and audio outputs of been trimmed from this diagram.

- **Python LUT Generator**: Using the user-defined audio sampling frequency and distance between mics, we use a Python script to generate an 180-degree LUT. The values in this table correlate each angle $\in$ [0, 180] to the number of samples delay necessary to wait before sending audio signals from a second mic.
- **Angle Delay LUT**: We make use of the FPGA's switch array, specifically sw[7:0], to implement the selection of the beam focus angle. The base-10 value of this angle is then displayed on the FPGA's seven segment display. We use the selected angle, in conjunction with the previously-generated LUT, to generate the respective delay for each mic in the linear array.
- **Delay, Sum, and Shift**: Using the mic data from the TDM module, along with the delays generated from the angle delay LUT, we create a BRAM for each mic's received audio data in this module. We keep address pointers such that we only add in the received data from that particular mic to the output signal if the proper number of samples delay has been reached. Because our active BRAM depth corresponds to the number of samples delay, we can keep the memory demands for this module relatively conservative. More specifically, for our 4-microphone linear array, the maximum time delay possible is for the microphone at index 3 with an input angle of 0 degrees, in which case the time delay $\Delta\tau_3 = 3 * 0.35 * cos(0)/343 = 0.0030612s$, which, at our sample rate of 39062.5 Hz., corresponds to 119 samples of delay, or an active BRAM depth of 119. Considering we have four of these BRAMs, each at a lower active depth than this, and each BRAM stores 24-bit audio, the memory usage is minimal.

III. Audio Output

A. Saving to Audio File

- **UART:** In order to rapidly test and preview the results of our beamformer, we utilize UART to transmit audio data from the FPGA to a computer. Although the microphones capture 24 bits of audio, to remain within the capabilities of UART at standard baud rates, the audio samples must be truncated. In addition, to properly reconstruct the audio stream at the UART receiver, each transmission must send an alignment bit. Fig. 4 illustrates the difference in audio with and without this alignment bit. Just before 0.5 seconds in the top waveform, the receiver loses alignment and the audio becomes loud nonsense. After adding the alignment, the full audio stream is received without issue. Finding a fix to this issue was critical to developing a working system, but it also set new requirements for our UART transmission. These requirements constrained our system to sending 14 audio bits and two alignment

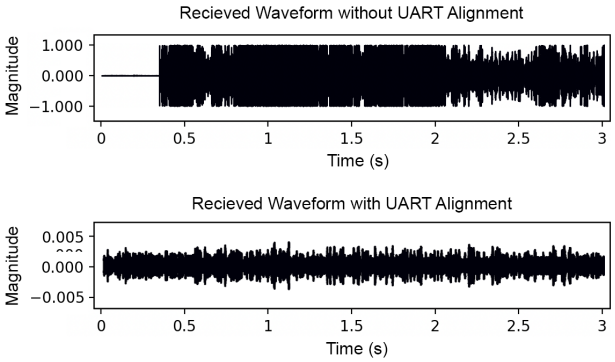

Fig. 4: Audio waveforms before and after UART Alignment

bits across two UART transmissions. Sampling audio at 39.0625kHz requires a baud rate of at least 625,000, so we utilize the nearest standard baud rate, 921,600. Fig. 5 shows the structure of the data bytes for each UART transmission (start and stop bits are ignored in this figure).

UART Data High			UART Data Low	
1'b1	audio[13:7]		1'b0	audio[6:0]

Fig. 5: UART Audio Sample Structure

- **Python Audio Visualization and Saving:** We save the received audio in a wavefile on the user's computer for playback. We also plot the audio samples received for inspection. During our design process, we also used these scripts to save and plot the audio samples from each microphone to perform Python-simulated beamforming and verify that our time-delay approach was viable.

B. Generating Real-time Audio

- **PDM** Our FPGA is designed to accommodate square waves or PWM/PDM signals passed to the audio out port, so we chose to use PDM for the best audio quality [2]. Pulse Density Modulation (PDM) [3] adjusts our audio sample to more accurately represent an analog signal as a stream of bits. Usally, PDM uses a low pass filter, but the bandpass filter built into the board at the audio out port also performs the job of integrating our signal. In order to accurately represent our audio signal we need to perform PDM at an oversampled rate. We choose an oversampled frequency of $128 \times 39.0625\text{kHz} = 5\text{MHz}$ because it aligns with a standard of oversampling either by a factor of 64 or 128 and is conveniently the same frequency as our SCK. At this clock, our audio signal is broken down into a 128 bit bitstream that when averaged by the boards bandpass filter produces high fidelity audio outputs. In simulation we found that our produced output has a $< .01$ difference in density as compared to the original input signal. Fig. 6 shows the transformation of our 24 bit audio samples into their 128 bit bitstream form.

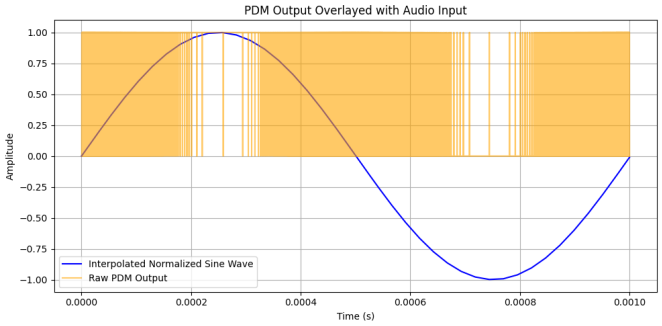

Fig. 6: The Raw PDM output has the same density as the original signal, but doesn't accurately convey how high fidelity the representation is.

When convoluted to simulate a low pass filter, we generate a graph displaying a nearly identically shaped signal although with a smaller amplitude, as displayed in Fig.7.

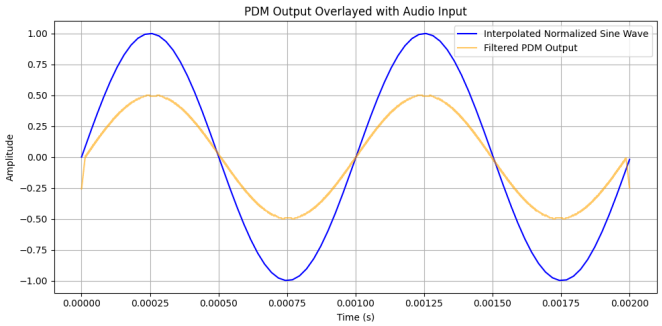

Fig. 7: Simulated convolution of PDM signal to visibly demonstrate likeness to original signal.

On the FPGA we qualitatively judged our output audio. We found the output audio to be audibly clear, although at a lower amplitude, as the simulated figure suggested, but it did have a constant noise. As of now, we believe this noise may be due to some misalignment between our PDM implementation and the FPGA band-pass filter.

C. Audio Playback

- **Off-chip Audio Storage:** A final goal for our system is to implement long-term off-chip audio storage to enable on-button-press recording playback. We intend to implement a modified traffic generator module to write to DDR memory and read from it upon a button-press trigger. We will then use the PDM module to play this audio via line out. Unfortunately, due to time constraints, we were unable to accurately implement our audio playback feature.

IV. Design Evaluation

A. Preliminary Evaluation

Before building the full system, we tested beamforming on a smaller two-microphone array. We simulated beamforming by applying delays in Python to audio streams from each

microphone. These tests showed a slight attenuation of the audio when the simulated beam was not directed at the angle the audio source was positioned from the array. These results gave us confidence to move forward with the project and led to the results of the full system detailed below.

B. Audio Sweep at Fixed Beam Angle

To test that the beamformer amplifies audio at the specified beam angle, we performed a sweep with a constant audio source from around 90 degrees to 0 degrees at a constant distance. We set the beam angle to 0 degrees. Fig. 8 shows the audio waveform that resulted from the sweep. There is clear amplification of the signal at the end of the waveform as it enters the beam formed by the beamformer at 0 degrees. There are other smaller amplifications throughout the sweep which are consistent with side lobes. Side lobes are an expected artifact in delay-and-sum beamformers, which are caused by aliasing. Thus, qualitatively, we see the beamformer successfully amplifies audio at the desired beam angle.

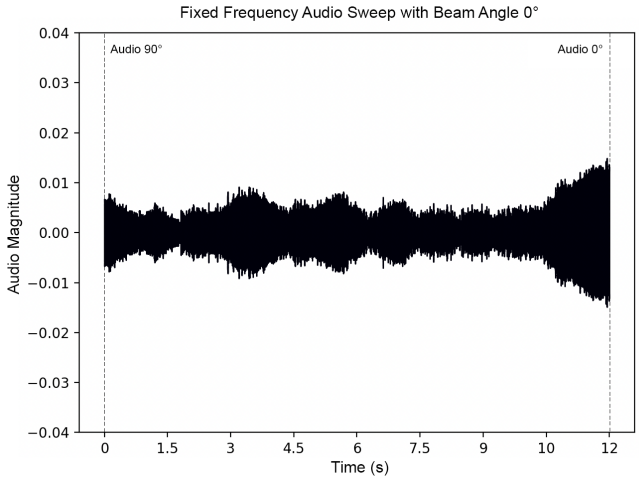

Fig. 8: Audio waveform of fixed frequency audio swept from 90 deg to 0 deg across 12 seconds with the beam angle set to 0 degrees for the duration of the sweep.

C. Human Speech and Fixed Frequency Audio Filtering

Our primary goal in building an acoustic beamformer was to spatially filter audio; in particular, we aimed to attenuate a fixed frequency ”noise” signal which was spatially separated from human speech. To setup for this test, we positioned a speaker playing a 1kHz signal at 90 deg and about 8.5 ft from the microphone array. We then had a subject speak 10 deg from the microphone at the same distance. There are two evaluations here, first qualitatively that the subject’s speech is clearer than the noise signal. Second, quantitatively, changing the beam angle towards the subject (away from the noise) attenuates the noise signal. This test was then replicated by swapping the position of the subject and the noise signal to show the system works in multiple orientations.

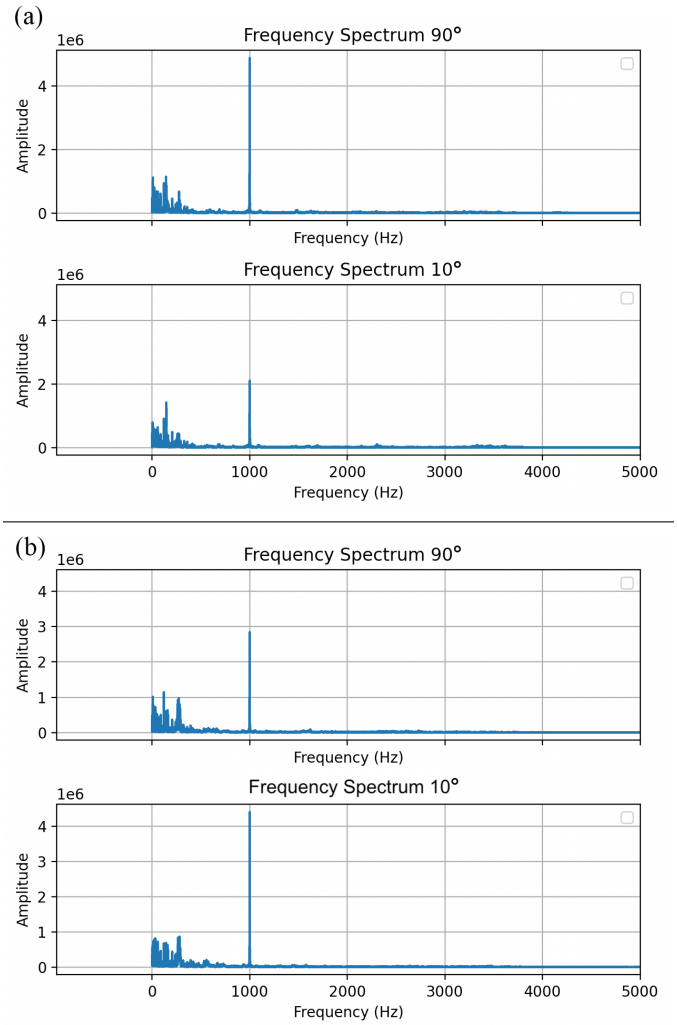

Fig. 9: Frequency spectrum of captured audio at beam angles of 90 deg and 10 deg. Plot a) shows results when noise at 90 deg. Plot b) shows results when noise at 10 deg.

Fig. 9 shows the frequency response of the audio captured by our beamforming system. When the beam angle is set to the angle of the speaker, there is a clear attenuation of the 1kHz noise signal in comparison to when the beam angle is set towards the noise source. The attenuation in Fig. 9a. is over 50%. The attenuation in Fig. 9b. is over 25%.

V. Retrospective

In developing this system we had to make implementation decisions about whether we wanted to operate in the time domain or the frequency domain, how to maintain algorithmic assumptions with hardware constraints, and what aspects of the system were necessary to demonstrate the functionality of our system.

We decided to perform our beamforming using the delay and sum algorithm which operates in the time domain. This decision was motivated by a desire to avoid moving between the time and frequency domain, such that we could focus more

on the concept of beamforming itself. While we were able to develop our system to meet the goals we had set, operating in the frequency domain to perform beamforming may have shown better results. Additionally, in inspecting our results we wondered if we may have been neglecting differences that may have been caused by changes in frequency. To this end, experimentation using a frequency based beamforming algorithm would be interesting to see.

With the delay and sum algorithm, we were operating under the assumption that our signal would be hitting each microphone roughly in parallel. This is a fair assumption when signals come from a farther distance, as would happen in most real-world beamforming use cases. However, our microphones weren't especially sensitive to picking up an audio signal from such long distances, and so we had to balance staying a far enough distance from the microphones to maintain our assumption and staying a close enough distance to actually capture audio signals. We iterated on the physical design of our system, initially using a single breadboard and phone audio to capture audio data, and then finally landing upon a larger board with mics spaced 35 cm. apart to capture audio data from speakers 1-2 meters away from the microphones. The larger distance between microphones made our delays more substantial allowing for greater differences in the output audio when applying the delay and sum algorithm.

In addition, we used speakers to try to create more evenly distributed noise signals when performing the tests. Additional hardware constraints included our PDM being bounded by the onboard filter provided by the FPGA and potential parasitic introduced by long wires in our final linear microphone array.

To reliably test our system we had to develop lots of infrastructure to output our signals. Much of our time was dedicated to creating a reliable connection between our FPGA and our computer to run tests and perform visualizations of our output signals. For our real time output, we didn't have long connections to our audio out port, so we had to become creative in ways to listen to our output audio without also listening to our input audio. Due to work in these areas we ultimately did not have time to implement other desired features like on board recording and playback, but those would be our intended next steps.

VI. Code

Our team's code [redacted]

VII. Contributions

- Patrick: Designed the PCB and physical setup. Wrote the top-level system and the UART module and module tests. Wrote sections of report for these as well as the Design Evaluation section. Also contributed to Python visualization scrips.
- Carlos: Wrote modules and tests for TDM and PDM. Did significant research on both protocols. Wrote the report for these modules as well.
- Olivia: Wrote the Delay, Sum, Shift module. Simulated time-domain beamforming and scripts to help visualize results in Python. Edited the final video. Wrote sections of the paper as well.
- Thank you to Joe for all the help during this project. The idea for the UART alignment bit significantly helped our performance.

References

[1] AudioLabsErlangen, "A Gentle Introduction to Beamforming." Oct. 05, 2021. Available: https://www.youtube.com/watch?v=etTcru7CrWU

[2] "Welcome to Real Digital," Realdigital.org, 2024. https://www.realdigital.org/doc/822e17a669a05f748c80af2274478bb5

[3] Kite, T. "Understanding PDM Digital Audio". Available at: Utexas.edu, 2024. https://users.ece.utexas.edu/~bevans/courses/realtime/lectures/10 Data Conversion/AP Understanding PDM Digital Audio.pdf