Audio Beamforming using “True-Time Delay” utilizing Digital FIR Filters on an FPGA

The traditional Beamforming approach used with RADAR is completed with analog phase shifters, which are not very accurate which would destroy the noise quality if used here. Therefore, if you were to delay in time instead of phase you would be able to get a high quality “beam-steer” on the audio. The beam steer equations should be the same as in an antenna array, just with the speed of sound rather than the speed of light.

As Seen below an 8-element array with an element separation of .3m each, would allow for a good 60-degree steer here

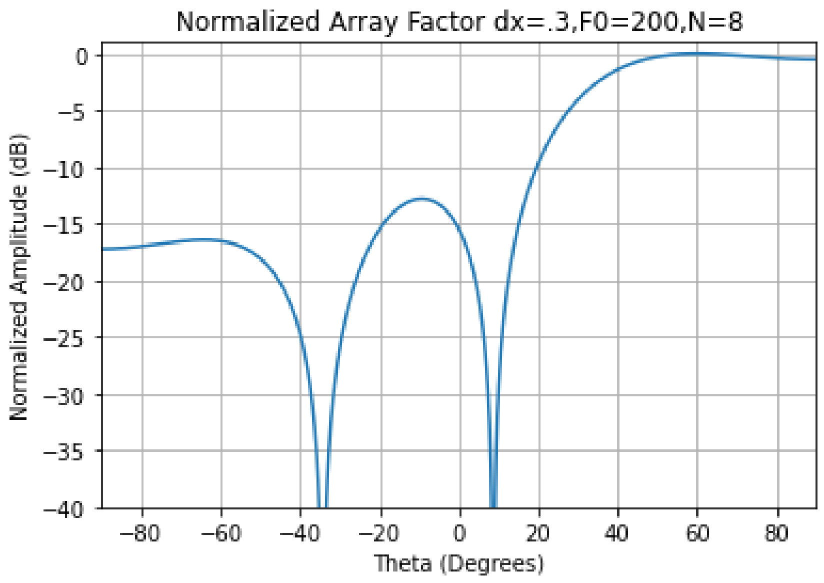


Fig. 1. Example of plugging into Array Beam steering equations for c=343m/s

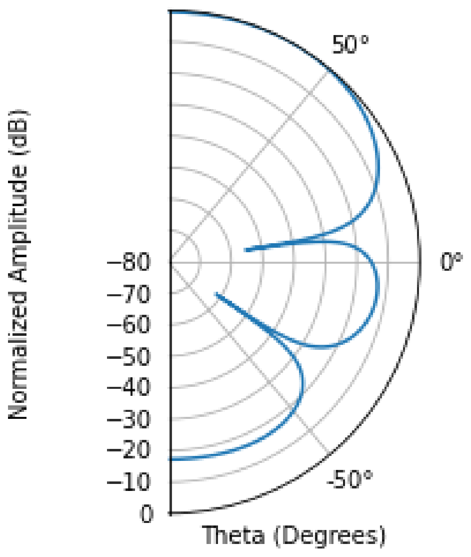


Fig. 2. Polar Equivalent Plot

The neat thing is, its much easier to get a time delay instead of a phase delay with simple trigonometry as you can easily find the extra distance the sound wave must travel with the equation below.

$$delay = \frac{dx * sin(steerAngle)}{343m/s}$$

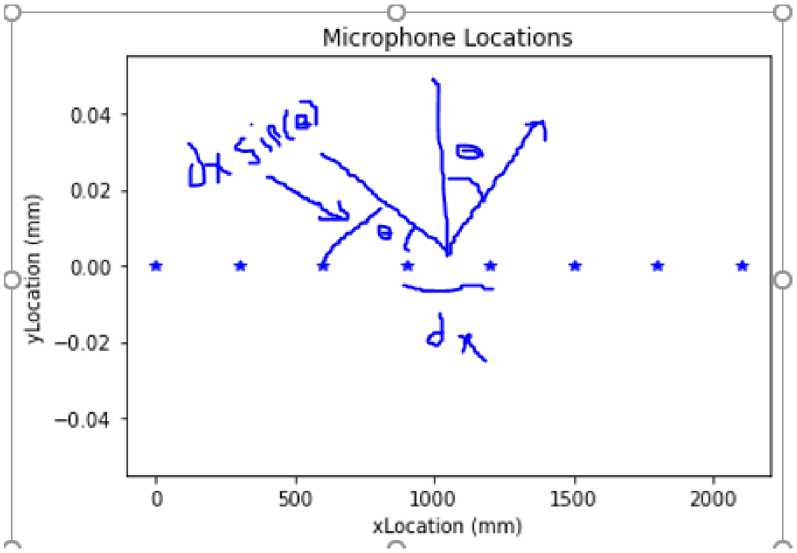


Fig. 3. Distance Equation

Converting Time delay to FIR Coefficients makes more sense to think in terms of samples rather than time.

$$\text{Sample Delay} = \text{timeDelay*Sampling Frequency}$$

Ex: time delay = 1.866ms, Fs = 44.1 kHz, then sample delay is 82.2989 samples.

Now the 82 samples of delay can be trivially handled by an internal buffer in the FPGA, so we really care about the fractional .2989 sample delay. This is non trivial, but the equations are already known and can be used as below.

```
for j in range(filterLen):
    x = j-fracDelay[i]
    sinc = sin(np.pi*(x-centerTap))/(pi*(x-centerTap))
    window = .54-.46*np.cos(2*pi*(x+.5)/filterLen)
    coeff[j] = window*sinc
```

Now this will still have a group delay of (Ntaps-1)/2, but every channel will have this delay, so it will be irrelevant. You could model the whole delay with this group delay calculation, but you would run into problems of having too many taps which would utilize too much of the FPGA Fabric.

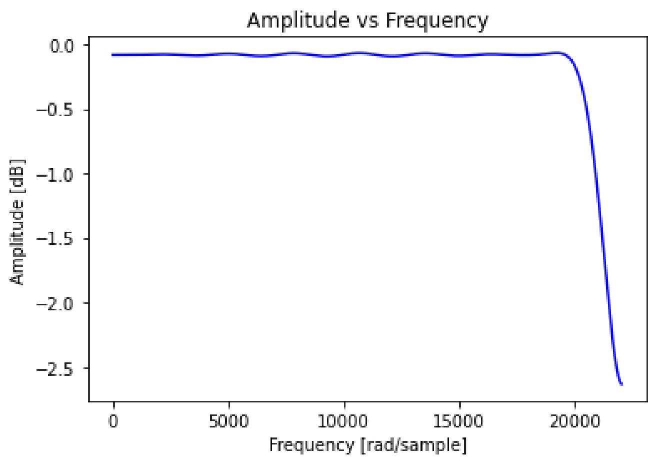


Fig. 4. Frequency response example of digital filter (When sampling at Fs = 44.1kHz)

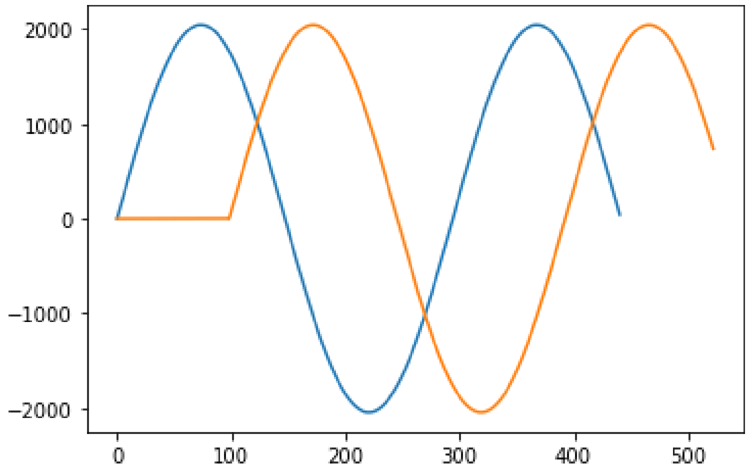
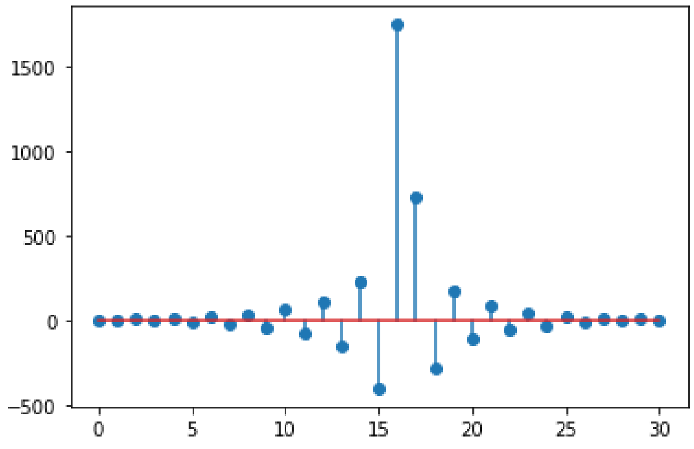


Fig. 5. Convolution of example Sin wave with this digital filter (Samples vs. Amplitude was scaled to be more like the digital signed 12 bit sample would be)



FIR Coeff’s for a 31 tap filter

- Things that would need to be designed:
    1. PCB with 8 speakers or microphones + 8 DAC’s or ADC’s and whatever other sub circuits are needed
    2. Convert the Python Code to C so that we can run it on an ARM Processor on a relatively cheap (130$) SoC like Zynq’s or Pynq’s by Xilinx that encompasses the FPGA and Processor (An FPGA is necessary, because I don’t think a microcontroller can measure 8 ADC’s/DAC’s simultaneously)
    3. Verilog/VHDL code for representing the FIR Filter and taking in the DAC/ADC Samples
    4. Dynamic Coefficient Loader for placing the FIR Coeff’s in the FPGA FIR Filters
- Some Stretch Goals
    1. Add a Computer Vision Targeting system to figure out what angle to steer to target someone
    2. Have everything internally handled by the SoC (Without an external Control/Helper Computer)
- Reasons why this process benefits from being digital
    1. Should allow you to dynamically pick out speakers in a room by just changing the steer angle/FIR Coeff’s
        - Or in reverse of dynamically pointing sound to a location in a room
    2. Should allow for keeping a high quality on the audio since the digital filter is just a low-loss low pass filter
    3. If you handle the FIR Filter processing off chip then you could theoretically pick out multiple speakers and clean up their audio by changing the FIR Coefficients you process the audio with.