Ultrasonic Frequency Pitch Shifter

Team Static Shock!

John Michael DiDonato, Anurag Garikipati, Dustin Klingensmith

Table of Contents

Executive Summary

Human beings are severely limited in the range of frequencies they are capable of hearing. Humans can perceive frequencies in the range of 20 Hz to 20 kHz, whereas animals like dogs, bats, and whales can hear frequencies well above 20 kHz. The frequencies north of 20 kHz are known as ultrasonic frequencies, and they can help people better understand the world. The military and medical industries already utilize these frequencies in their respective industries to better accomplish their goals. The aim of this project is to introduce ultrasonic frequencies to everyday people. Using the LPCxpresso54608 microcontroller, we develop a device that maps ultrasonic frequencies existing between 24 - 36 kHz to 0 - 12 kHz, a range almost exclusively within human hearing. Our solution hinges on a simple frequency pitch shifter algorithm which isolates the frequency range 24 - 36 kHz via bandpass filtering and shifts these frequencies down to the 0 - 12 kHz range through intentional aliasing. To accomplish this design, the LPCxpresso54608's DMIC samples at 96 kHz, the DMA controller manages data transfer to relieve the board's CPU, an 81-tap FIR bandpass filter designed using the Parks-McClellan algorithm isolates the frequencies in the range 24 - 36 kHz, and the resulting signal is decimated to shift those frequencies down to the 0 - 12 kHz range. Our group successfully shifted an ultrasonic version of the Mario theme song back to a range perceivable to humans.

Problem Statement

Humans can typically perceive noises with frequencies in the range of 20 Hz - 20 kHz. Ultrasonic noises consist of sounds that exist north of 20 kHz, a point beyond human hearing. This leaves people without the ability to detect noise sources or interpret the sounds contained in the ultrasonic range. The defense and medical industries, as well as various animals in nature, use these ultrasonic frequencies to better understand the world around them. Military submarines utilize sonar technologies to create a picture of an underwater environment, similar to whales and dolphins. Medical ultrasound is used to help doctors see the development of babies in the womb, ensuring the healthy delivery and growth of the child. Being able to hear and visualize these sounds offers a chance to better understand our surroundings. It also provides a basis for many of the more complicated ultrasonic technologies such as medical ultrasound and military sonar applications.

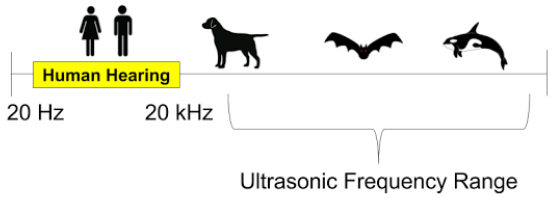

Figure 1: The Limitations of Human Hearing

Scaling ultrasonic frequencies down into the human hearing range will expose individuals to the ordinary noises surrounding them every day that they are unable to detect like the ultrasounds produced by cracking knuckles. Hearing these frequencies will help people determine the sources of ultrasonic noises, experience a world they didn't know existed, and open the door to using ultrasonic noises to help people navigate the world, especially those who are disabled. As stated the military and medical industries already use ultrasound to help visualize the world around them. While visualization is beyond the scope of a senior design project, being able to comprehend ultrasound is an important step in developing technologies to help people sense the entire world surrounding them.

Proposed Solution

We will build a device capable of capturing noise outside the range of human hearing, transforming those frequencies into the audible range, and playing the sound back for the user on an external speaker. The device will be capable of real-time performance, so that the user can listen to the ultrasonic sounds as they are being produced nearby.

Implementation Details

The project was intended to meet the following design requirements:

1. Capture sounds above 20 kHz
2. Shift a specified band of ultrasonic frequencies into the range of human hearing (20 Hz - 20 kHz) using a pitch shifting algorithm
3. Achieve real time performance
4. Play the sound back at audible frequencies using an external speaker

The project was designed using the LPCxpresso54608 development board produced by NXP, and implemented using the Keil Uvision5 toolchain. The development board is pictured below in

figure 2. With a maximum sampling rate of 96 kHz, we found that the built in digital microphone was capable of recording frequencies beyond 40 kHz, with acceptable performance for our needs. This allowed to us to implement the entire project using only the peripherals that come with the board, which helped keep costs low and reduces furthering the complexity of the project

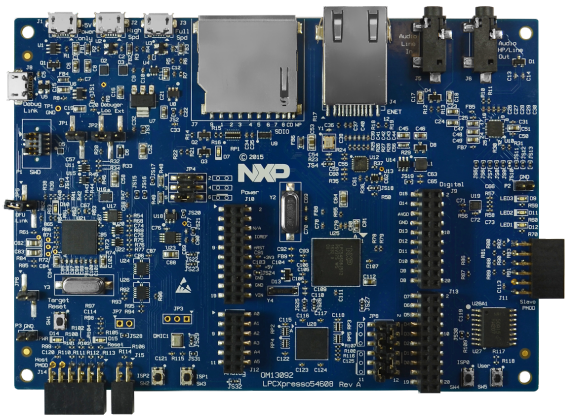

Figure 2: NXP's LPCxpresso54608 Development Board

The original pitch shifting algorithm was intended to be a phase vocoder. After implementing an initial version of a standard phase vocoder in MATLAB for testing, it was discovered sufficient real time performance would be difficult to achieve given the stated design goals and time constraints. Instead a simpler approach was taken. This alternative design includes the use of bandpass filtering and the intentional aliasing of the signal. According to standard communication systems theory, by isolating the desired frequency range using a standard order FIR bandpass filter and decimating the signal appropriately, any signal can be linearly mapped down to a lower frequency range. Typically, this is strongly avoided, however, in our case, it became the central feature of the pitch shifting algorithm. A few additional important project features include the DMA used for managing data movement, decimation to achieve aliasing to shift frequencies, and an FIR filter for bandpass filtering. The following figure provides an overview of the system present in our device.

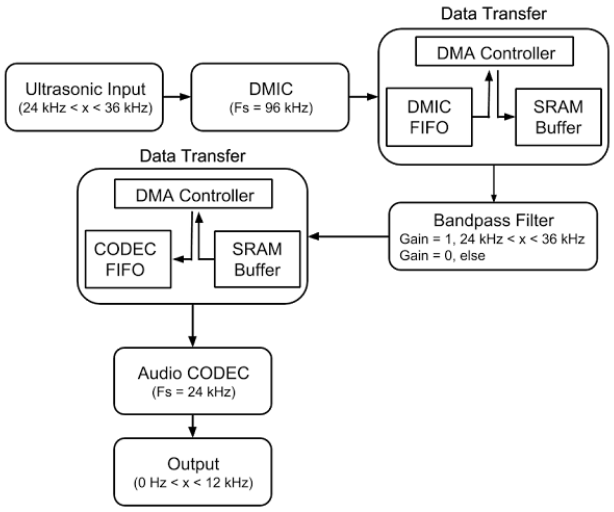

Figure 3: System Overview

The first block of the project involves setting up the digital microphone to capture high frequency sounds. The main clock for the microphone subsystem is produced using the audio PLL at 24.576 MHz. This clock rate is then divided by a factor of four to set the PDM rate at 6.144 MHz. Using the 2FS mode of the microphone, the PCM sample rate is then calculated using the following equation:

$$PCM\ Rate = \frac{PDM\ Rate}{2 * OSR}$$

OSR is the oversample ratio used in the PDM to PCM conversion. In our case, using a sampling frequency of 96 kHz and using the maximum PDM rate of 6.144 MHz, we need an OSR value of 32. Using this low OSR value allows us to capture the needed ultrasonic frequencies, while having the less desireable effect of slightly decreasing the frequency resolution. This is a tradeoff our team was willing to accept as ensuring the capture of ultrasonic frequency was a stated goal of the project's design. The subsequent reduction in frequency resolution, however, still remained at an acceptable level and did not interfere with any of the other project goals..

The sampling rate was set to 96 kHz, the maximum rate supported by the LPCxpresso54608's digital microphone. This sampling rate provides roughly 10.4 microseconds between samples. In order to improve performance, the DMA controller is set to handle data movement between the digital microphone and a ping pong buffer in memory. Using a buffer length of 1024 input samples, the buffer are filled and ready for processing every 10.67 ms. Further increases in performance were realized by increasing the clock rate of the CPU to 180 MHz, which is as fast as the board used is rated for.

After the ping pong buffer is filled, a DMA callback function is called to set a flag to signal the CPU. Next, an FIR bandpass filter is used to isolate frequencies between 24 - 36 kHz. This is the band that will be aliased down to the range of 0-12 kHz for playback. After considerable filter testing, our group decided to go with an 81 tap FIR filter designed using the Parks-McClellan algorithm. The filter response can be seen in the following figure.

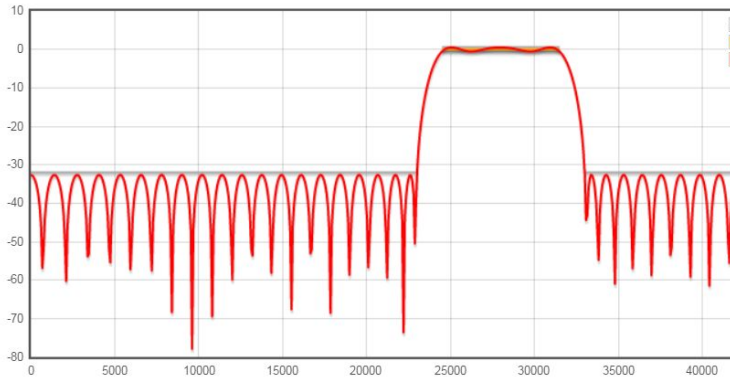

Figure 4: 81-Tap FIR Bandpass Filter Magnitude Response

The filter implemented uses the arm_fir_decimate_f32 floating point function provided in the CMSIS-DSP library. The function takes advantage of block processing to reduce computations, and handles the decimation. In this case, we intend to move from a sampling rate of 96 kHz on the input, to a sampling rate of 24 kHz on the output, where the audio CODEC will turn the samples back into an analog signal. This requires a sampling rate reduction by a factor of four. The sampling is done in the following code snippet.

```
215 //-------------------------
216 //   Decimating Filter
217 //-------------------------
218
219     float32_t *inputBuff  ;
220     float32_t *outputBuff ;
221
222     // Apply the decimating filter using block processing
223     for(int i = 0 ; i < BUFFER_SIZE/BLOCK_SIZE ; i++ )
224     {
225       inputBuff = pfiltBuffin + i*BLOCK_SIZE ;
226       outputBuff = pfiltBuffout + (i*BLOCK_SIZE)/DECIMATION_FACTOR ;
227       arm_fir_decimate_f32(pfiltDecInst,inputBuff,outputBuff,BLOCK_SIZE);
228     }
```

Figure 5: Bandpass Filter and Decimation Code Snippet

The aliasing is easier to visualize using a carpenter's ruler diagram, provided in figure 6. According to the Nyquist Theorem, with an output sampling rate of 24 kHz, or maximum frequency, and the folding frequency for the carpenter's ruler here, will be 12 kHz. This is what provides the linear mapping of the 24-36 kHz band down to the 0-12 kHz band.

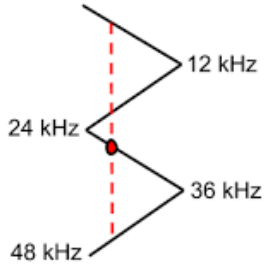

Figure 6: Intentional Aliasing

With the data filtered and waiting in a temporary buffer, it is ready to be sent to the audio CODEC and turned into an analog signal. This process is also handled using the DMA controller. Because the CODEC provides an interrupt to the DMA controller whenever the transmit FIFO register reaches a specified amount in order to begin the data transfer, a larger set of buffers were required on the output in order to avoid memory collisions. The CODEC DMA callback function is executed whenever the end of the current buffer is reached, and at that time, the pointers for the buffers used for the DMA transfer, and for the CPU to store the filtered data are swapped. A set of four buffers were implemented here. There is a buffer of data between the buffer being filled with the most recently filtered data, and the one being used to supply the CODEC, in both directions. This allows the pointers to be swapped with less concern of what the CPU is currently accessing. It also adds an extra 10 ms delay, reducing real-time performance slightly. The performance was still found to be more than acceptable though, especially considering the fact that humans can not hear the original noise source in the first place.

```
91  // Rotate the pointers around to the appropriate buffer
92  if(i2s_TxTransfer.data == &pingOUT[0])
93  {
94    i2s_TxTransfer.data   = &pongOUT[0] ;  // Pointer for CODEC DMA transfer
95    codecBuffPtr          = pong2OUT ;     // Pointer for CPU filtered data output buffer
96  }
97  else if(i2s_TxTransfer.data == &pongOUT[0])
98  {
99    i2s_TxTransfer.data   = &ping2OUT[0] ;
100   codecBuffPtr          = pingOUT ;
101 }
102 else if(i2s_TxTransfer.data == &ping2OUT[0])
103 {
104   i2s_TxTransfer.data   = &pong2OUT[0] ;
105   codecBuffPtr          = pongOUT ;
106 }
107 else
108 {
109   i2s_TxTransfer.data   = &pingOUT[0] ;
110   codecBuffPtr          = ping2OUT ;
111 }
```

Figure 7: Data Management Code Snippet

Conclusion/Results

In order to verify the successful implementation of our design, a piezo buzzer was used to generate ultrasonic sounds. Due to the inherent nature of ultrasonic noises, relying on naturally occurring ultrasounds would be difficult to verify as humans can not hear them. For this reason, the team preferred using a piezo buzzer capable of generating specific ultrasonic frequencies in the range from 24 kHz to 36 kHz. It is important to note, any naturally occurring ultrasonic noises that existed in this frequency band were also shifted with the frequencies generated by the piezo buzzer. Piezo buzzers possess a piezoelectric element which elicits a mechanical response i.e. vibration when an electrical signal is applied. The electrical source can be tailored by changing the voltage and current of the signal to produce vibrations at specified frequencies. Using an arduino to control and modify the electrical source sent to the piezo buzzer, we were able to get the piezo buzzer to ‘play’ the mario theme song by vibrating the buzzer at different frequencies in a specified order. We then shifted all the notes in the mario theme song by 24 kHz so that the song played in the ultrasonic range.

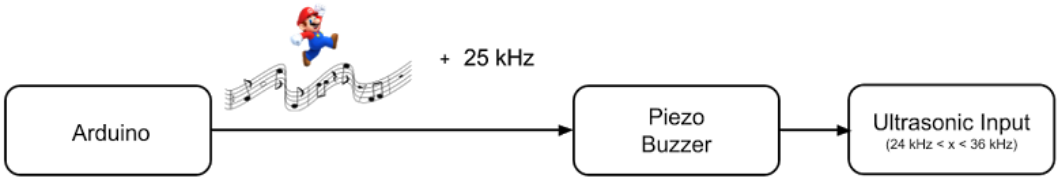

Figure 8: Ultrasonic Frequency Input Generation Process

Using the device described earlier, our group successfully shifted the ultrasonic version of the Mario theme song back into the range of human hearing. We were able to successfully implement our design. The following figure depicts our project setup.

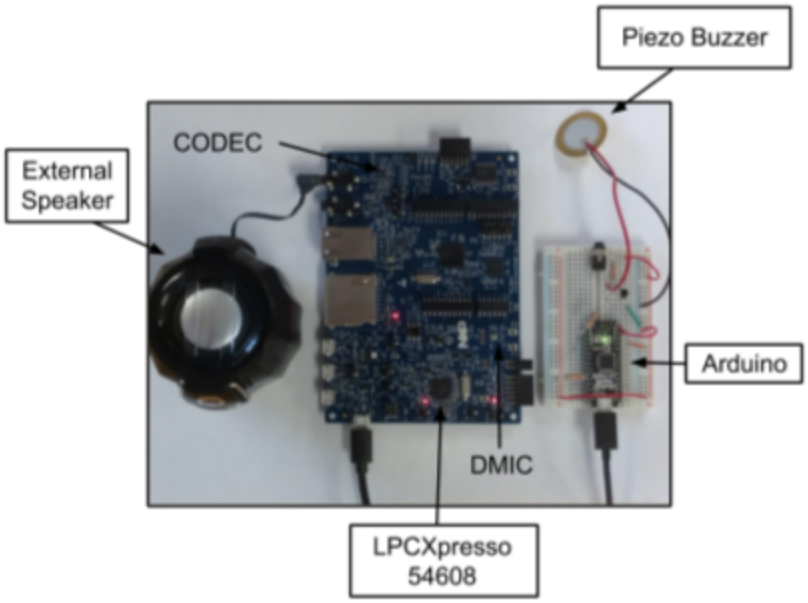

Figure 9: Hardware Configuration

We were able to verify the success of our design by shifting the ultrasonic version of the Mario theme song back down to a frequency range where humans can here. The following figure depicts the digital input at 25 kHz and the analog output at 1 kHz.

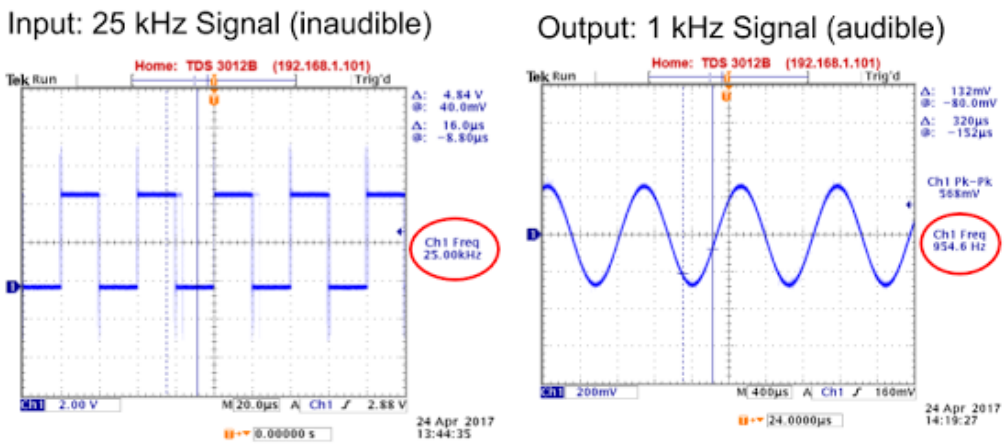

Figure 10: Ultrasonic Pitch Shifter Results

The team was able to accomplish the four stated goals of the project. Frequencies above 20 kHz were captured by sampling the board's DMIC at 96 kHz. Ultrasonic frequencies were shifted down into the human hearing range by using both bandpass filtering and intentional aliasing. We were able to achieve near real time performance with only a small 10 ms delay. Finally, the shifted frequencies were successfully played back using an external speaker. The design could be improved in the future by using a more advanced pitch shifting algorithm like the phase vocoder we initially explored using. Additionally, the device could display the power spectrums of both the ultrasonic input and down shifted output to better depict the frequency shifting the device is performing.