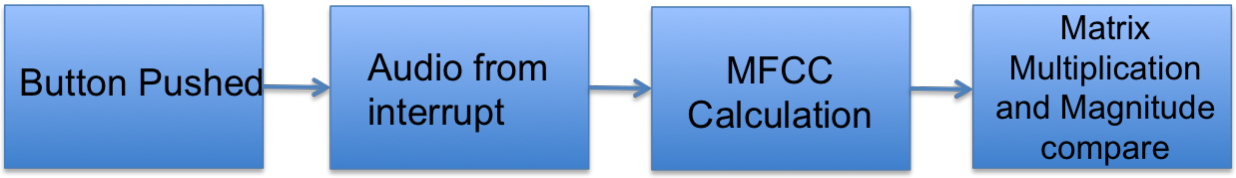
Brendan Larkin
Pavan Mehrotra

EE109 Final Project: Speech Recognition

Introduction

For our project, we created a speech recognition system on the Altera DE2-115 FPGA. The system used standard speech signal processing and machine learning to distinguish "Yes" and "No," with over 90% accuracy

System overview:



Implementation

Our system is based on a music visualizer project from last year by Axel Sly and Travis Geis, whose project in turn was based on Lab 3. We decided to start from there since it already had a working FFT module integrated. This module, called bel_fft and created by Frank Storm, was used in our project to perform hardware FFTs. From the QSys file, we removed the TLDA module (used for the visualization part) and rewired parts of the circuit. We experimented with FFT sizes, but decided to leave the current FFT module from the old project unaffected. This module performed 128 point FFTs in hardware. We changed the input to the ADC, which was originally set to line in, to the microphone in. We did this since we would would be using a microphone to speak, and the microphone input usually has an amplifier since it is made to deal with the low input signals from a microphone. We also lowered the sampling rate from 48 kHz to 32 kHz, the lowest hardware setting, since we did not need that much fidelity. Since this sampling rate was also too high, we decreased the sampling rate to 8kHz Hz by downsampling in software.

After fixing a couple bugs in the audio ISR from the previous project, we implemented the "brute force" version of our system, which was later changed into a machine learning version. The brute force version can be toggled on in the code by changing the global boolean "bruteForce" in main.c. The brute force version originally functioned by prompting the user to say "yes" and then "no" for a baseline. The ADC values were recorded for just over one second (8192 samples) and then transferred over in 128-point frames to the hardware FFT module to create a spectrogram of the samples. For each frame, the result was normalized by dividing the entire spectrum by the square root of the total energy. These baseline spectrograms were then stored separately for comparison on subsequent trials, to distinguish "yes" from "no."

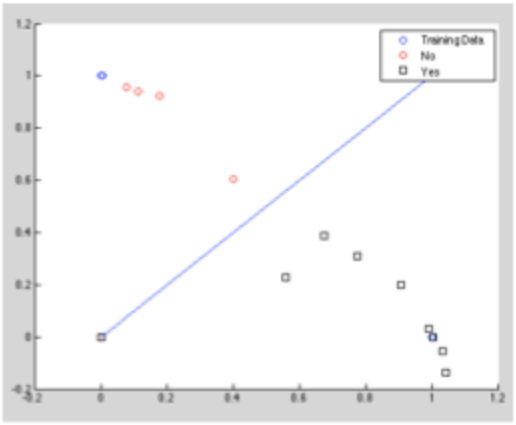
Upon testing this system, we noticed there was a strong time dependence on when a word was spoken to the accuracy of the prediction. This was to be expected since we were simply comparing spectrograms, which are by definition time dependent. We then changed our

brute force method to perform a sort of sliding dot product between the sample and the two baselines, and taking the maximum of the comparisons as values to compare for final determination. So, instead of just comparing the samples to the baseline, we multiplied each point in the sample spectrogram by the baselines separately, and then circularly slided spectrogram over by one FFT frame (128 points) and recomputed this product. The maximum value for each baseline was then compared to determine which word was spoken.

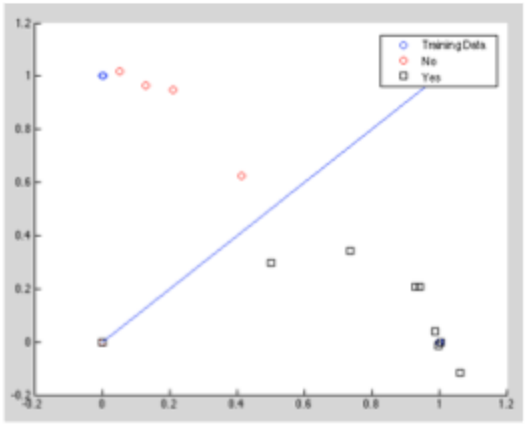
Although this gave us reasonable performance, we wanted a more accurate method, so we turned to machine learning techniques. We obtained sample spectrograms from the system and used linear regression to create a matrix for evaluation and determination of the input. This matrix was generated by taking the pseudoinverse of our training data and multiplying it by the known results. We used 10 training samples to generate this version. Upon instantiating and testing this matrix in our program, we saw the same time dependence we had seen earlier with our first brute force method. In fact, it seemed to be performing worse than that, most likely due to the small training group size. We then decided to switch to using the Mel Frequency Cepstrum Coefficients (MFCC) to try to get better features, which is what a group at Cornell did who built a similar system. However, ours would still use machine learning techniques rather than a simple and somewhat random observation for classification that the group at Cornell used.

In order to calculate the MFCC for each 128-point frame of audio, there are several steps we needed to carry out. Normally, the first step of any process involving the short-time Fourier transform is windowing. However, in our MATLAB simulations, we found that the difference in discriminability between using a Hanning window and a rectangular window was negligible:

Rectangular Window



Hanning Window



Having concluded that we did not need to compute a windowed frame, we first calculated the power spectrum by computing the squared magnitudes of each FFT component. From the power spectrum, we used a bank of triangle filters that models the human ear to calculate the energy of the signal in each of 20 overlapping frequency bands. Because humans perceived loudness on a logarithmic scale, we also computed the logarithm of these energies. Finally, to calculate the cepstrum, we needed to perform an inverse Fourier transform, but because we

know that the power spectrum is real and even (since the audio signal is real), we can use a discrete cosine transform (DCT) instead. Additionally, because the speaker-independent information about the audio signal is contained in the lower-order cepstral coefficients, we use a low-pass filter (a.k.a. “lifter”) to extract these coefficients.

We first used a MATLAB implementation of this process to perform simulations on test data from our computers (Wojcicki 2011). When we dissected the code, we found that each of the processing steps described above (with the exception of the logarithm and power spectrum) was implemented with a matrix multiplication, so we decided to compute the MFCCs on the board in this same manner. This meant we just needed to copy over the matrices from MATLAB that were being used for the triangle filter bank, DCT, and “liftering” computations. We were able to combine the DCT and liftering stages of computation into one matrix multiplication, to save processing time, but we could not combine all three matrices because we needed to take the logarithm after calculating the triangle filter bank energies.

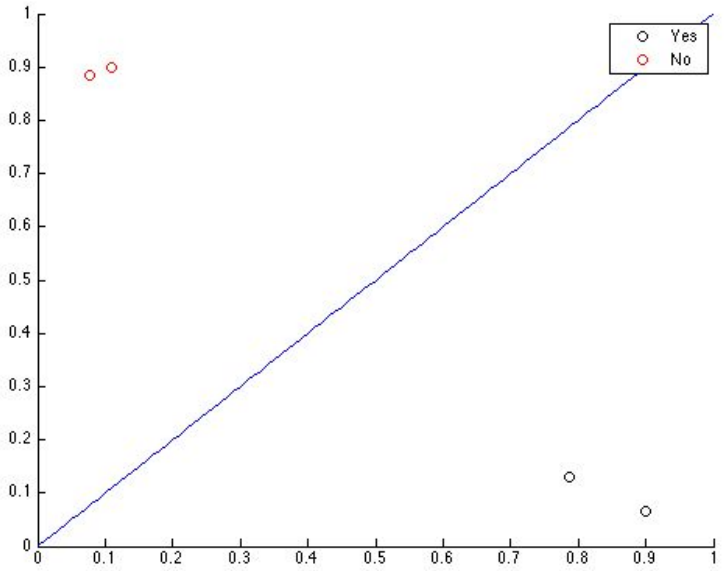
Because this process reduced the length of each FFT vector from 128 to 20 energy values, and from 20 energy values to 13 coefficients (after the low-pass liftering), the feature vectors used for linear regression became much smaller than those used in the brute force approach. After figuring out how to transfer data from the FPGA to our own computers, we collected 15-20 audio samples of “yes” and “no,” and performed a linear regression using the pseudo-inverse approach. We copied over this matrix from MATLAB onto the FPGA, and our C code simply multiplies this matrix with the MFCC vector of the speech signal to determine whether a “yes” or a “no” was spoken.

Operation

Before operation, plug a microphone into the pink microphone in line and a speaker is plugged into the green line out. To operate the system, push BTN1 on the DE2-115 board and speak into the microphone, as mentioned on the LCD character display. The red LEDs should turn on (not to full brightness). After this, there is a one second timeframe over which the word “yes” or “no” can be spoken. After, the green LEDs should turn on and the character LCD display should show the words, “Processing.” After about 8 seconds (for the machine learning version), the result should be displayed in the console. A pair of numbers, followed by a print out of “Yes” or “No” will appear. The first number is the output of the processing assuming “yes” was spoken, and the second is that for “no.” The printed result is the greater of the two numbers.

Evaluation

Our final system has >90% accuracy. The system, however, is trained to one voice, and experiences much worse accuracy when another user attempts to use it. On the few trial runs we had, the system got most of them wrong, and the magnitudes printed out were about even, meaning that system was essentially guessing. Below is a graph created in MATLAB that shows the accuracy expected using four trial data points collected from the FPGA:



The x-axis is the magnitude for “yes” and the y-axis is the magnitude for “no.” The distance from the blue line indicates the confidence. All four trials produced the correct output (they are on the correct side of the blue line).

Problems Encountered

We had a lot of trouble getting started and framing our system as a whole. We looked into different modules to calculate FFTs and into different methods of speech recognition. The first issue was solved by basing our system off of code from last year, which had an FFT module. After researching Hidden Markov Models and other forms of machine learning, we decided on linear regression due to its ease of calculation.

The first major problem we encountered was getting data off the board to use for offline training. When we attempted to print a large number of data points, the console in the Altera Monitoring Program would print out large sections of blank spaces between values, which could not be copied past. To solve this issue, we attempted to use ethernet packets and read them from our computers. We could not get this to work, as the interface for raw ethernet packets is difficult to interact with programmatically. We then attempted to use RS232 to get the data. One issue with this was that both the output of the cable and the input of the board had metal tabs on the side that prevented the cable from plugging in. We spent a long time trying to take these off the cable. Once we could actually plug the cable in, we could finally read data through PuTTY. The method used to do this is present in the current version of our code.

Another issue we had was that we could not initialize the large matrix needed for the machine learning version of our system that was based on the FFTs. The compiler could create space for the large array, but would throw an error when we attempted to initialize it. To fix this, we declared an empty array and fed the values in through PuTTY. This method is also still present in the final solution.

The calculation speed was another problem we had. Our final system initially took ~15 seconds to calculate results. We were able to drop this to ~8 seconds by using a lookup table

to find the non-zero values of the triangle filter matrix. Also, since we could not find the compiler flags, we manually unrolled some loops.

Future Work

Future work would include speeding up our system. It currently takes about 8 seconds to generate a result, which is far too long. One method would be to create a floating or fixed point matrix multiply module on the FPGA to accelerate these calculations. In addition, turning off audio interrupts during calculation would help, since this is firing often during this timeframe. This was left on simply so that the user could still hear themselves in the speaker while audio was being processed, but this is unnecessary.

In the future, it would be nice to have better speaker independence. The current system is trained to one of our voices, so it would be good to explore either training both of our voices, better forms of regression, or some other way of determination using the MFCC as a feature. We would like to experiment with logistic regression, which seems to be more often used in classification problems. Lastly, in the future we could also expand the vocabulary of our system to include more words.

GitHub Link: https://github.com/pavmeh/fpga-speech-recognition

References

Aakash, J., Paliwal N., Panchagnula K. T., (2010).“Real Time Speech Recognition Engine.” https://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2010/np276_ksp55_aj355/np276_ksp55_aj355/

Geis, T. & Sly, A. (2015). ottobonn/fpga-music-visualizer. GitHub repository. https://github.com/ottobonn/fpga-music-visualizer

Lutter, Michael. (2014). “Mel-Frequency Cepstral Coefficients”. http://recognize-speech.com/feature-extraction/mfcc

Storm, F. (2013). fstorm. “bel_fft:FFT co-processor in Verilog based on the KISS FFT”.https://sourceforge.net/projects/belfft/

Wojcicki, Kamil. (2011). HTK MFCC MATLAB. http://www.mathworks.com/matlabcentral/fileexchange/32849-htk-mfcc-matlab