# Amplitude and Frequency Visualization of Sound

Broberg, Felix
950414-7852

Duong, Julia
890922-1189

December 9, 2016

## Objective and Requirements

**This is intended as an advanced project.**
Our objective was to visualize real-time sound input from a microphone, connected to the chipkit, through graphs and visualizations displayed on the I/O-shield display. We sampled the sound through our microphone and computed and displayed an amplitude-time graph, a visual representation (pulsating oval) of the current amplitude of the audio and, with the help of a fast Fourier transform (FFT) algorithm, an amplitude-frequency graph.

Proposed project requirements:

- Graphs have to be displayed on the I/O-shield display.
- Amplitude graph must be computed.
- Frequency-Amplitude graph must be computed.
- The user should be able to switch between the two graphs using switches on the chipkit.
- The sound is inputed via the microphone connected to the chipkit.

Optional features:

- Record the input sound and save it to a file.
- Play a soundclip through the chipkit while displaying the graphs.

Of the proposed project requirements, we implemented all of the main requirements as well as a visual representation (pulsating oval) of the current amplitude of the audio. However, none of the optional features were implemented due to a lack of necessary equipment. Furthermore, we added the feature that the user can switch between sampling frequencies using buttons 3 and 4.

## Solution

The project source code was written in C with implementations of an integer square-root function[1] and an FFT[2] taken from external sources. The project was developed on the chipKit uno32 board together with the Basic I/O shield and a preconfigured microphone and amplifier. The I/O shield display is used to present the graphs and sound amplitude representation and the microphone, along with the A/D converter, is used for sound input.

The switches on the I/O shield enables us to switch between graphs. With no switches enabled, the chipKit computes and displays the amplitude-time relation, with switch 4 enabled, the chipKit

[1] http://www.codecodex.com/wiki/Calculate_an_integer_square_root which implements figure 2: http://www.embedded.com/electronics-blogs/programmer-s-toolbox/4219659/Integer-Square-Roots by Jack W. Crenshaw (Integer Square Roots)

[2] Written by: Tom Roberts 11/8/89, Made portable: Malcolm Slaney 12/15/94 malcolm@interval.com, Enhanced: Dimitrios P. Bouras 14 Jun 2006 dbouras@ieee.org, Ported to PIC18F: Simon Inns 20110104. See project source-file "fft.c" for further information.

computes and displays the amplitude-frequency relation and with switch 3 enabled (overriding other switches) the visual representation of the current amplitude will be displayed.

The buttons enables the user to switch between sampling frequencies. A sampling frequency of 10 kHz is pre-set enabling an amplitude-frequency graph with domain 5 kHz to be calculated and displayed[3]. Pressing button 4 on the Basic I/O-shield will set the sampling frequency to 20 kHz, enabling a frequency domain of 10 kHz, and pressing button 3 will set the sampling frequency to 10 kHz once again.

## Verification

The accuracy and precision of the FFT and our computations were tested by playing sounds to the microphone and comparing the displayed graphs with correct values. By playing single-frequency tones ranging from 100 Hz to half our selected sampling rate, we could investigate how accurately and precisely the sound was sampled and computed. Although the microphone's sampling range was 100 Hz - 10 kHz[4], sampling at 20 kHz still yielded results with high accuracy but slightly low precision. This could be seen by the graph having a correct peak frequency while displaying some amplitude readings for the adjacent frequencies. With a sampling rate of 10 kHz, the graph showed no tendency of spread around the peak frequency and instead showed a single amplitude column at the expected position, indicating both high accuracy and precision. Having concluded the correctness of our microphone, performing similar comparisons between the audio input and the displayed graphs proved the correctness of the amplitude-time graph and the amplitude representation as well.

## Contributions

The work was divided in the following way: Julia focused on the I/O-shield development, including the buttons and the graphical aspect, and Felix worked on the audio input and audio computations (amplitude-frequency, amplitude-time and amplitude representations). Naturally, we assisted each other in the development of our focus parts when facing difficulty as well as when combining our work into a functioning unit. The abstract was written collaboratively in order to best represent all aspects of the project.

## Reflections

Several challenges were faced during the development of both focus parts. Sampling the sound with automatic sampling at a set frequency requires both calculations of the sampling time in regards to the system clock rate as well as the handling of interrupts when sampling conversion is complete. Getting the hang of this took some time, but once completely understood, performing calculations on the result and changing sampling frequencies was relatively straightforward.

We intended to use another FFT algorithm (KissFFT), but getting it to work on the chipkit proved difficult and the performance advantage over our chosen FFT seemed minimal. As the performance of our FFT has proven to be sufficient and the simplicity of the calculations enabled us to focus on other parts of the development, we argue that this was an advantageous choice.

Showing our results on the I/O-shield display was somewhat of a challenge as it required pixel-perfect representations. However, once we understood how the pixels were partitioned on the display, we could create functions that updated the display based on columns and height. As our project is

[3] See "Nyquist frequency" for further information.

[4] https://www.sparkfun.com/products/12758

heavily dependant on the array-stored results from the audio sampling, this solution enabled us to connect our two development areas with ease.

Overall, our project has taught us a lot about hardware interaction and analog-digital data handling. Some aspects that seemed easy to implement proved themselves to be rather difficult, while other aspects that looked hard were easy. Once certain steps had been taken in the development, for instance getting the sampling to work, moving on and creating new things came natural, for instance computing the amplitude-time relation.