FPGzAm – Song Identification System

Proposal for the 6.111 final project, November 8, 2010

Yafim Landa Pranav Sood

Overview

We can represent an audible song as a time-varying signal, which can be fed to a speaker to make the signal audible. In our case, we will instead feed this signal to the FPGzAm music identification system. The goal of this system is to listen to sound sample and determine whether it is a part of a song that the system knows about. FPGzAm works as follows:

We first listen to a set of songs and generate a database of known songs. This database contains pairs of signal **fingerprints** and **track numbers**. The fingerprints are generated in the identification subsystem of FPGzAm. This step generates a database of fingerprint and track number pair, one per song.

We then listen to a short sample from the microphone and generate a fingerprint for that sample. The generated fingerprint is fed to the search subsystem of FPGzAm. The purpose of the search subsystem is to identify the fingerprint as a part of an existing fingerprint in the database.

Finally, if the search subsystem identifies the fingerprint as a part of a known fingerprint, we output the track number to the labkit LEDs. Otherwise, let the user know that this song is unknown.

We will present MATLAB simulation images of our Verilog project in this early written description. Our overall system architecture is depicted in Figure 1.

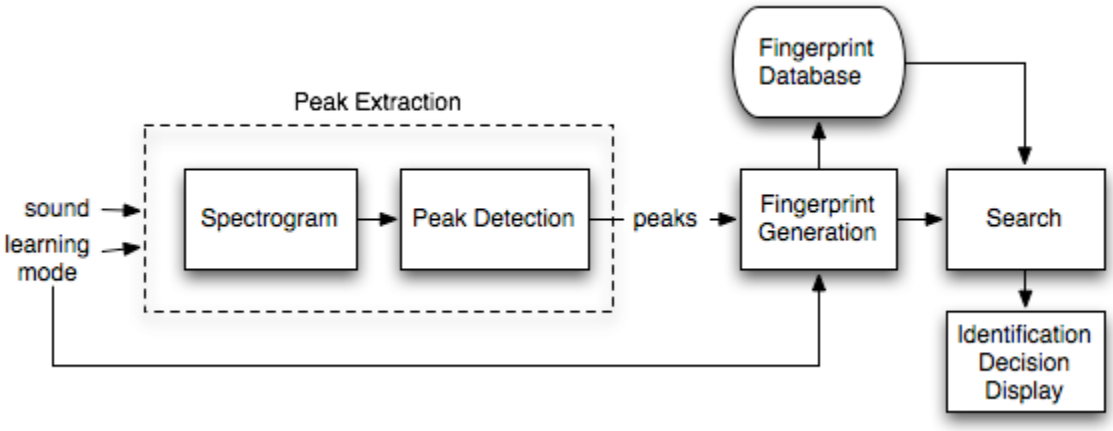

Figure 1. System-level block diagram.

1. Peak Extraction

Inputs: An audio signal.
Output: Spectrogram peaks of the input audio track.
This module consists of two sub modules:

1. Spectrogram
2. Peak Detection

When we're in learning mode, this module operates on the audio track that is to be stored in the database. Otherwise, it operates on the audio track to be recognized.

Spectrogram:

We begin with a signal that is sampled at 44.1kHz and fed to our system. This is represented in our system by a new value that is available every 22.7 microseconds. We chose a time period of 37 milliseconds over which we are able to acquire 1600 samples. Let us call these 1600 samples a **window**. A new window is started every 800 samples, so that our windows are overlapped by 50%.

On each window, we perform a Fast Fourier Transform (FFT), the result of which we call a **frame**. A frame maps each frequency in the range of 0Hz to 22,050Hz to intensity, which is indeed the output of our FFT. We therefore get around 1 / ((37 / 2) * (10^(-3))) = 54 frames every second.

Figure 2 shows the output of the spectrogram of a short guitar sample.

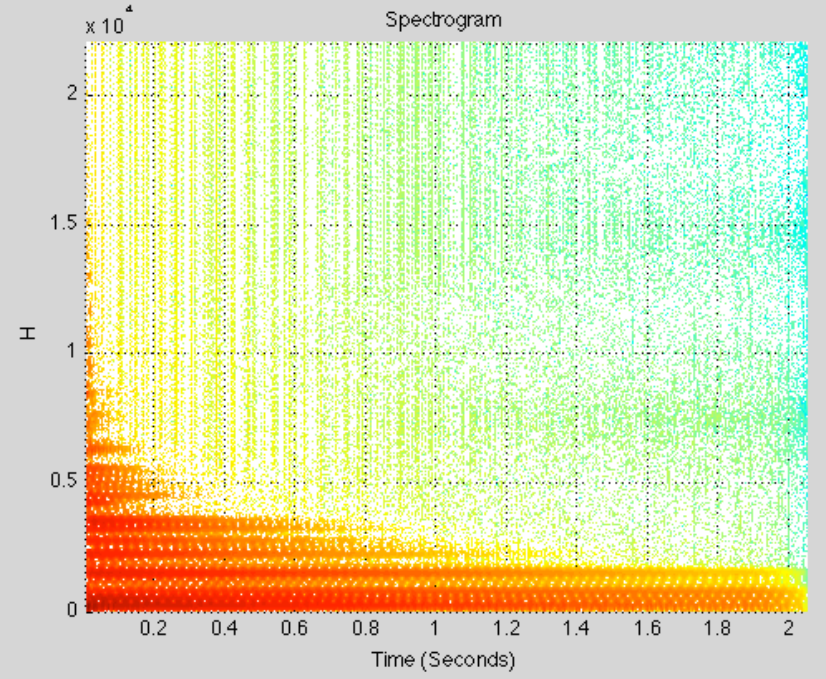

Figure 2. Spectrogram of a short guitar strum sound clip.

Peak Detection:

We will now calculate for each frame a frame **signature** as follows. Divide the frame into these ranges: 0 – 40Hz, 40Hz – 80Hz, 80Hz – 120Hz, 120Hz – 180Hz and 180Hz-300Hz. Within each range, we find the frequency with the maximum intensity value. Please note that the ranges get wider as the frequencies get higher. We take these five numbers and concatenate them, producing a long number that is the concatenation, and we take this number *modulus* 1024 to get one 10-bit value. The concatenation and modulus operation is a variant of a hash function. This 10-bit value represents a shuffled, truncated version of the long number that we had previously, and is our frame signature.

We are now ready to generate the **fingerprint** for a song that uniquely identifies it. As mentioned before, we get 54 frames each second. This means that we get 54 frame signatures per second, and since each frame signature is 10 bits, we are working with 54*10 = 540 bits every second.

For a 3-minute song, we need to save (3 minutes) * (60 seconds/minute) * (540 bits / second) = 95kb of data. We call this data the **fingerprint** of the song. The fingerprint generation module will perform the process of creating the fingerprints.

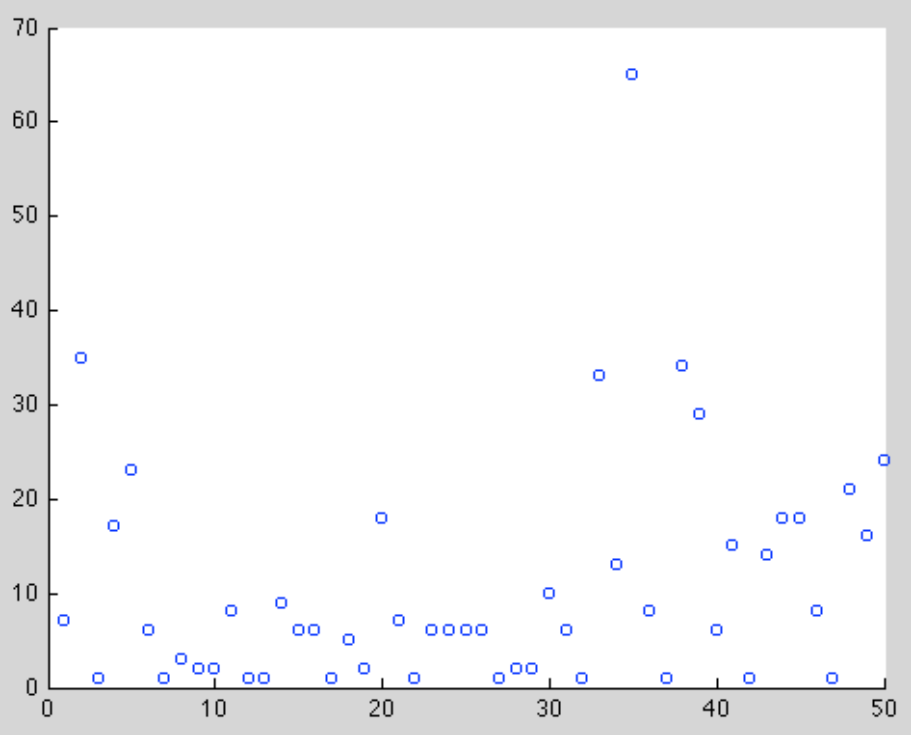

Figure 3. Fingerprint of a short guitar strum sound clip. Frames are on the horizontal axis and frequencies are on the vertical axis.

2. Search

Inputs: Fingerprints of the unknown music sample and the fingerprints stored in the database.
Outputs: Track number of the unknown music sample.

We are now given a signal, and our goal is to determine whether the fingerprint generated from this signal matches a fingerprint stored in our set of known signals. We do this as follows.
We begin with the fingerprints of the available song and the fingerprint of the sample song. We slide the sample song fingerprint through each available song fingerprint to see whether it's a match. We use the **Rabin-Karp string search algorithm** to perform this search quickly. The Rabin-Karp search algorithm works as follows:

Let the length of the fingerprint of our current full-length available song be n, and the length of the fingerprint of the sample song be m, where m << n. We will take the sum of all of the m numbers within the sample fingerprint. We now take the sum of the first m numbers of the full song. If the two sums match, then we compare the two signatures and return true if they also match. If the two sums don't match, then we shift our sum of m numbers one to the right by subtracting the number on the left and adding the

number on the right, and perform the checks again. We repeat this procedure n times, that is, until we reach the end of the string.

3. Fingerprint Database

Input/Output: Fingerprints of the known audio track.

This module is used to store the fingerprints that are extracted from the known audio track. This module may potentially include interfacing with the ZBT memory present on the FPGA, which can be used for saving the fingerprints of several songs.

Testing

Peak Detection
The peak extraction module will first be tested using a series of generated sine signals. The peak extraction module should detect peaks that correspond to these sines and output their frequencies to the labkit display. It can then be tested with actual music in a similar manner, comparing the results to our MATLAB simulation.

Search
The search module will be tested in a similar manner. First, it will attempt to detect a single sine function in a collection of generated sines. It can then be tested to comply with the MATLAB simulations by outputting the identified frequencies to the labkit displays.

External Components

1. MP3 player: It will be used to input the known audio track which will then be fingerprinted and stored into the database. . It will be interfaced to the FPGA using the AC97 codec.

2. Microphone: It will be used to input the unknown music sample from a mp3 player to the FPGA. It will be interfaced to the FPGA using the AC97 codec.

Division of Work

Pranav will build the peak extractor and the fingerprint database module.
Yafim will build the fingerprint generator, the search module, and the simulations.