Music Search Engine

By

Li Cao
Jason Chang
Tiffany Yeh

ECE 445, SENIOR DESIGN PROJECT

SPRING 2007

TA: Alex Spektor

May 1, 2007

Project No. 33

ABSTRACT

Music is often recognized not by its name, but rather by a familiar melody. It is currently impossible to automate a search for a particular song with only a tune. This project designs a music search engine where a user can find a song by playing the tune on a keyboard or humming a tune into a microphone.

First and foremost, we implemented an effective algorithm [1] to convert polyphonic music into an identifiable monophonic tune. Audio processing hardware was then used to extract the pitch of the input. Finally, a fast and effective search algorithm was implemented to provide the user with accurate results. Extensive testing was completed on the system to test its robustness and accuracy. Though not all tunes in a song are searchable, a short input of notes can identify a song as the top choice 83.6% of the time.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

Present day music search engines are text-based searches, and since the conception of the search engine, the main goal of developers has been to improve search time and produce more relevant results. However, there are other ways of improving a search. One way to improve a musical search is to change the type of input data. Thus, the goal of our project is different: to change a music search engine from being text-based to being musically based. The purpose is to allow users to hum or play a melody or tune and use the tune to accurately search a large song database. This is advantageous in a number of situations. Since many songs are remembered by tune and not by title, artist, or even the lyrics, it is inconvenient and often impossible to textually search for the exact song in such cases. This is a common problem encountered when everyday listeners attempt to search oldies tunes or songs performed by less popular or foreign artists. For this reason, we have developed a Music Search Engine that searches a song database with a user-inputted tune and outputs a list of matching song titles.

1.2 Specifications

There were four main specifications we defined for our project.

1) Power all circuits with the 5V supply voltage from a computer's USB terminal (i.e. no external voltage supply).
2) Contain all hardware within 1.0 ft^3 and neatly contained in project boxes.
3) Limit our search time to less than 10 sec.
4) Be successful in the search 90% of the time.

These specifications are aimed towards satisfying the consumer. Less power consumption and external supplies are obviously desired to reduce burden. A small form factor is also important because we do not want this system to take up an entire desk and deter people from purchasing it due to its largeness. Also, a short search time should be very important; if it takes longer than 10 sec to return a search, users can become impatient. And lastly, the system of course must perform well. We think that finding the song 90% of the time is performing up to high enough standards.

1.3 Subprojects

The entire search engine will be split up into four subprojects that were completely somewhat separately. Each subproject was tested individually to make sure that it was working before it was integrated with the entire system.

1.3.1 Building the Database

For the search to be performed quickly, we decided to prebuild a database of song melodies. This subproject involved extracting the pitch of the singing voice from a group of songs and storing them such that the search would be able to look at them effectively.

1.3.2 Acquiring User Input

Users of the search engine are given three ways to input a search string: virtual keyboard, device line-in, and microphone. The virtual keyboard is entirely software based, and can be used if there is no access to a music device or microphone. Our line-in input allows users to plug in any instrument with a line-out, such as a digital keyboard, and play their search string. Finally, users can hum or sing the tune into a microphone. For the hardware inputs, users must choose their input method using a toggle switch, and their input signal is sent through a prefilter and preamplifier to prepare it for DSP processing.

1.3.3 Real-time Pitch Extraction

The TI TMS320CS54x DSP board is programmed to sample the output of the previous circuit and take a 4096-point FFT of the user's input signal through a BNC connector. It will then determine the pitch the user sang or played, convert that into the closest corresponding piano note, and send that note to the DSP board's serial output upon command from the host computer. This is all done in real-time.

1.3.4 Serial-to-USB Conversion

Serial-to-USB is achieved using MAX232 and CP2102 chips. The MAX232 takes serial input from the DSP board and converts it into TTL asynchronous serial, which is converted again into USB signal for the computer.

1.3.5 Searching the Database

The music search engine software program asks for and receives data from the DSP board through USB. The processed user input is compiled into a search string, and run through an algorithm to determine which song in the database best matches this input. The final results are displayed on screen for the user.

2. DESIGN PROCEDURE

Our system can be thought of as four separate subprojects combined into one system. Figure 2.1 is a general block diagram of how our system is organized.

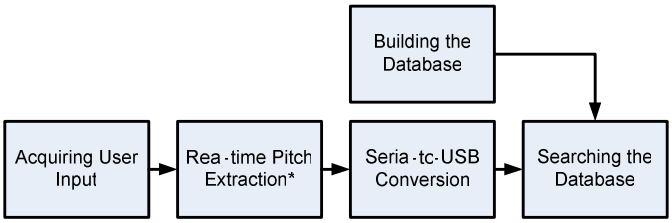

Fig. 2.1. General Block Diagram

2.1 Building the Database

The building of the search database was done completely before the user had any interaction with our product. The main concern here was to build an accurate database. After much consideration and research, it was decided that the main pitch extraction from the polyphonic audio would be too much to develop and implement in the time frame. Thus, we decided to use the algorithm proposed in [1], [2].

It is also important to note a few things about this algorithm. When considering Signal to Noise Ratio (SNR) in a song for our purposes, the signal is the voice, and the noise is all the background instruments. When the SNR is fairly high (~10dB), this algorithm does extremely well in extracting the pitch of the voice. However, when the SNR is unity or smaller (<0dB), this algorithm does not do very well. A lot of noise peaks and extra notes are inserted into the prediction, as well as missing notes altogether. In most common songs, there is some portion of the song where the voice is louder then the background (SNR>0dB). However, in most places of the song (especially the chorus), the background is louder than the voice (SNR<0dB). Thus, this algorithm does not do so well in extracting the pitch of the voice in most cases. Because of this, we must clean the data prior to sending it in, and after it finishes to eliminate as much noise as possible. This algorithm was definitely not an ideal pitch extraction algorithm. However, it was the best performing algorithm that could be found. Thus, we decided to use it. In future systems, a more accurate pitch extraction algorithm could improve results.

There is a lot of background information required to understand the pitch extraction algorithm to its fullest extent. This includes (but is not limited to) sampling theory, Fourier Transforms, correlations, time-frequency relationships, and Hidden Markov Models. However, the authors of this algorithm kindly provided source code in this algorithm, and thus, no time on this project was spent in developing or implementing it. For a completely understanding of how the algorithm works, please refer to [1], [2].

There are a few things that need to be addressed in how to integrate this algorithm with the rest of the system. Mainly, it should be noted that the algorithm takes in raw ASCII values of the sound wave that are scaled to have maximum amplitude of around 300. Also, the input into this algorithm can only be approximately 60,000 samples, and works best with a wave sampled at 16kHz. In addition to these input constraints, the output of the algorithm is actually two files, the first of which is of importance to us. This output file contains the lag (in milliseconds) to the peak in the correlation that corresponds to the voice. This lag is given for every 10ms of input data. Therefore, we must convert the lag of the peak to the actual frequency of the pitch. We use the following equation to do this (where f_i is the frequency of the pitch, F_s is the sampling frequency, and t_i is the lag of the pitch):

$$f_i = \frac{F_s}{t_i} \tag{2.1}$$

It should also be noted that almost the entirety of human singing pitches lie above 100Hz and below 3kHz. This will be important for cleaning the input data to the pitch extraction algorithm.

The last major design constraint we need to consider is storage. In a real-world application of this system, the database would contain millions of songs. Thus, an effective compression needs to be implemented. However, because of the importance to preserve the data, we would like to use a lossless compression. Therefore, we chose to use a compression coding scheme similar to the image compression technique called run-length coding. Run-length coding is described in more detail in [3]. Basically, it codes the value, and then the length of the value. In this case, we will store the frequency information along with how long the note is held for. This coding method was the most logical lossless compression we could think of.

Figure 2.2 is a general block diagram of how we build our database.

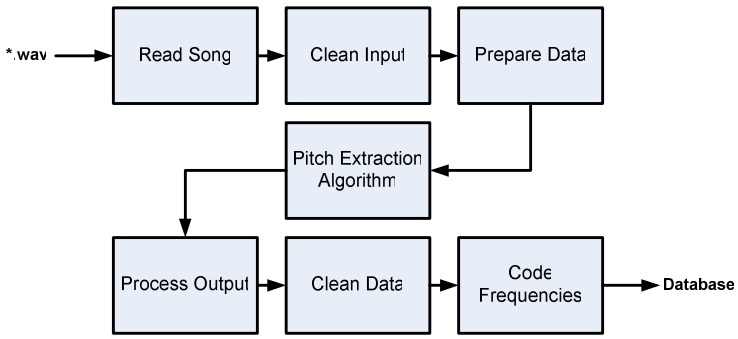

Fig. 2.2. Building the Database Block Diagram

2.2 Acquiring User Input

Aside from the direct virtual keyboard input, there are two ways in which the user can input a melody. The first is the microphone input that powers and extracts the voltage levels produced by the user's voice (typically ~ 0.5 mV). The other input is an audio line-in that is an analog signal from an instrument such as an electronic keyboard. The user chooses the input by toggling a switch. The selected input is then sent through a preamplifier circuit (Figure 2.3) that produces a gain such that the DSP will be able to recognize the signal.

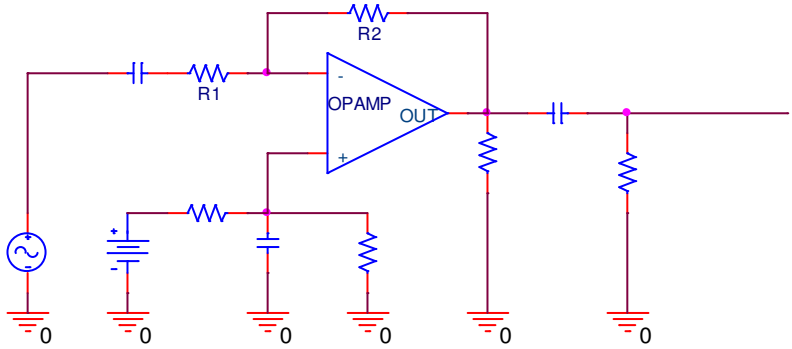

Fig. 2.3. Building the Database Block Diagram [3]

The following equation is used to calculate the actual the gain of the circuit shown in Figure 2.3.

$$A_v = \frac{A_{vout}}{A_{vin}} = 1 + \frac{R_2}{R_1} \tag{2.2}$$

The DSP needs approximately 1-2V peak to peak to recognize the signal correctly. The gain for the circuit will be chosen such that this specification is met.

2.3 Real-time Pitch Extraction

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk} \qquad k = 0, \ldots, N-1 \tag{2.3}$$

$$H_k = \sqrt{A_k^2 + B_k^2} \tag{2.4}$$

The job of the real-time pitch extraction algorithm was to pick out the dominant pitch the user was singing or playing in real time by calculating the FFT and power spectrum of an input audio signal. We had to design our FFT for expected input signals ranging from 150 – 1000 Hz (the expected frequency range for normal human voices). Also, the FFT had to be able to distinguish every half-step between notes, which is approximately 8 Hz at the lowest end of the spectrum.

$$h(k) = g(Mk) \tag{2.5}$$

$$\text{frequency resolution} = \frac{\text{sampling rate}}{\text{downsampling factor} \times \text{NFFT}} \tag{2.6}$$

To meet these requirements, we had to adjust the sampling rate by downsampling and the FFT size. By increasing the downsampling factor, M in equation 2.5, and FFT size, we could obtain our target frequency resolution. However, high frequency harmonics from singing that also have high power would produce harmful aliasing if the downsampling factor was too high, and the algorithm would run too slowly if the FFT size was too large.

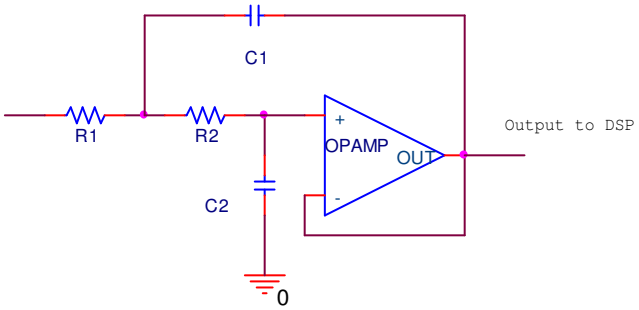

Fig. 2.4. Schematic of a standard Sallen-Key filter using an op-amp [5]

$$C1 = 2 \times C2 \tag{2.7}$$

$$R1 = R2 = \frac{1}{(2\pi \times f_0 \times C2)} \tag{2.8}$$

We decided to implement a hardware low-pass Sallen-Key [5] type prefilter to get rid of high frequency harmonics. This allowed us to downsample by a higher factor without worry of aliasing. Also, by not

including the filter in the DSP algorithm, we ensured the pitch extraction algorithm would run as fast as possible.

The pitch data for each sample cycle is sent out via the serial output buffer, which is read by the computer. However, the computer must first request data by sending a signal to the DSP. Each time the computer polls the DSP, it receives one set of pitch data. This way, we can control the speed of sampling user input to every 100 ms. We decided to implement this slow sampling rate in order to try and eliminate wavering from microphone input. When someone tries to hold a note, their voice sometimes wavers between the desired pitch and a half-step above or below. Continuous polling would record the user fluctuating rapidly between two adjacent notes, which is not desirable for our search implementation. By polling at a relatively slow rate, we dampen this fluctuation while maintaining responsiveness for fast songs.

2.4 Serial-to-USB Converter

Serial data transmitted from the DSP is converted to USB so that it can easily communicate with newer PC's that lack relatively outdated serial ports. This is achieved by sending and receiving serial data to/from a MAX232 chip that converts serial to asynchronous (also referred to as TTL level) data. Serial data is approximately 2.5 V, whereas asynchronous is inverted and approximately 5V. The asynchronous data is then sent to the CP2102 breakout board (featuring a USB controller, voltage regulator and preprogrammed internal EEPROM for device description) that converts the data to USB communication and a USB (B to A) cord connects the device to a USB port on a computer. Finally, the necessary drivers, given by [6], must be installed on the user's laptop/PC for proper device recognition.

2.5 Searching the Database

The implemented search algorithm was specifically designed such that it would work well with an imperfect database and/or minor mistakes from the user inputted search string. Also, our system specifications say that our search should be independent of the key the song is in or the user is in. This means that instead of searching for specific notes, we must search for the difference between consecutive notes. To understand this, a little music theory is needed.

All songs are in the key of something. This key defines (among other things) which notes on the scale should be sharp/flat. The key of C major is defined with no sharps or flats. The key of C# has the exact same scale, except that everything is shifted by one half-step. Therefore, for every half-step change in key, a resulting half-step change is applied to the rest of the song. In other words, if we only look at the difference between notes instead of the actual notes, we drop only the key information without losing anything else. Therefore this method gives us exactly what we want without any other losses.

Thus, our search algorithm searches for differences in notes. In addition, because we assume that the input string is much more accurate than the database, we always try to find the entire string within the song. Skipping of a note in the input is also permitted to find the string, however these results will be weighted differently. For each song, we find how many occurrences of the search string exist throughout the entire song. Then, to find the percent that the specific song is what the user is searching for, we use the following equations (where O_j is the j^{th} occurrence of the string input, n_i is the number of occurrences found in song i, N is the total number of occurrences found in all the songs, and R_i is the percentage that song i is the song the user is searching for):

$$n_i = \sum_j O_j \tag{2.9}$$

$$N = \sum_i n_i \tag{2.10}$$

$$R_i = 100 \times \frac{n_i}{N} \tag{2.11}$$

As stated previously, if the user decides that skipping notes is also allowed, then we must weight the number of occurrences accordingly. We found that a linear scheme in weighting the occurrences to the number of skipped notes worked the best. In other words, we see that the number of occurrences found in song i is now defined by the following equation (where s_j is the number of skipped notes for the j^{th} occurrence of the string input, and α is a parameter chosen by the user).

$$n_i = \sum_j \frac{O_j}{s_j \times \alpha} \tag{2.12}$$

It should be noted that α should be chosen such that it corresponds with the level of certainty that the input string is indeed part of the song the user is looking for. If the certainty is extremely high, the user would want to increase α to make sure that skipped notes are penalized more. If the certainty of the input string is rather small, then the user would want to decrease α to make sure that skipped notes are not penalized a great deal. As α decreases, a wider variety of songs will be returned, each of which will have less percentage of being correct.

3. DESIGN DETAILS

3.1 Building the Database

Referring to Figure 2.2, we see that building the database is just comprised of a few smaller steps. The logical flow chart for building the database is shown in Figure B.1. We will also briefly go through some of the steps here.

We designed the entire algorithm in Matlab for its ease of signal manipulation in the frequency domain. The actual pitch extraction algorithm was optimized for speed in C++. Matlab's built-in audio signal processing packaged includes the functions: "wavread" and "wavwrite" which make it easy to deal with *.wav files. Therefore, we first used the free program, Winamp, to convert the songs to the correct wav format.

As stated in Section 2.1, the algorithm used to extract the pitch of the singing voice is not very accurate when the SNR<0dB. Therefore, we tried to clean the input and output as much as possible to produce better results for the database. To eliminate as much noise as possible from background, we first put the input signal through a bandpass filter with a low cutoff frequency of 100Hz, and a high cutoff frequency of 3kHz.

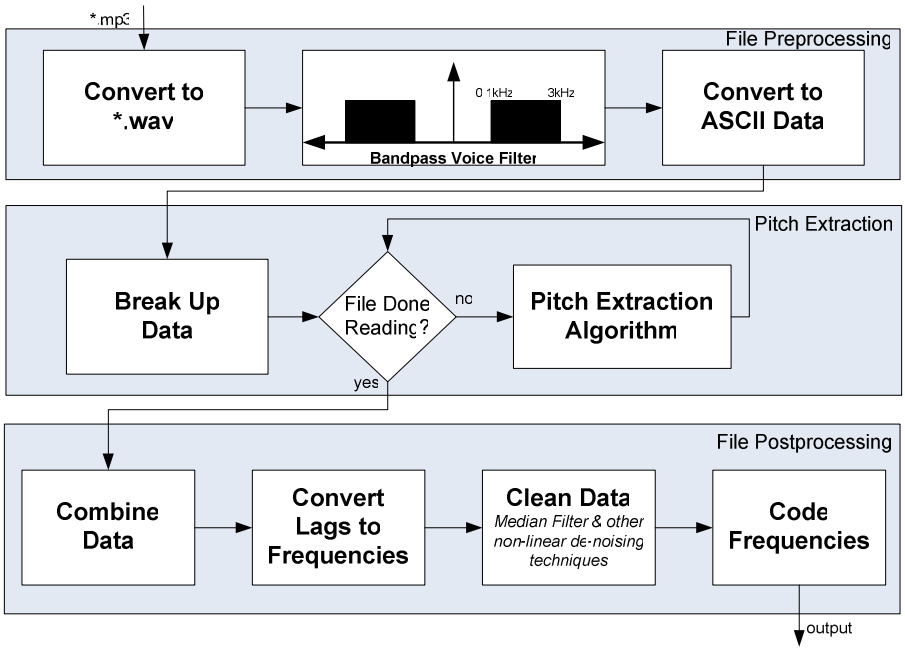

Fig. 3.1. Building the Database Detailed Block Diagram

Also, because of the algorithm constraints stated in Section 2.1, we much preprocess the sound wave to ASCII data, and break it up into smaller chunks. We feed each chunk through the algorithm, and then recombine them when they are done. Using equation 2.1, we convert the output of the algorithm to the frequency of the voice.

We then worry about cleaning the output. Many times, the algorithm picks up a very short noise spike. To clean the data, we first pass it through a 15 point median filter. This filter reduces short noise pulses without distorting the correct pitches. In addition to the median filter, we also check each consecutive note. If the next note is greater than an octave away from the current note, then we interpret it as noise. In most cases, this is valid because a singer rarely jumps more than an octave in between notes. These

two methods were able to clean the output data significantly so that a low SNR input could still produce a somewhat respectable output.

After we clean the data, the final step is to losslessly code the frequencies such that they are easily searched. Instead of storing the actual frequency of the note, we will store the index of that frequency on a piano. Please refer to Table A.1 for the mapping sequence. Table 3.1 gives an example of how the mapping of frequencies to indices is done. As stated in Section 2.1, we then use run-length coding to store the data. For each note, we store the note frequency index and the length that the note is held. Because each lag is outputted for every 10 ms of input data, the number we are storing for the time the note is held is actually the number of 10 ms chunks. In other words, if a note is held for 1 sec, we would store 100. In addition to run-length coding, we also store the total number of notes within the entire file. This makes it easier in the search application to actually declare the size of the array before reading all the data.

The final output to the database will be in the following format (from Table 3.1):

TABLE 3.1 Database Coding Format

Line 1 (number of elements)	4
Line 2 (frequency index for note 1)	40
Line 3 (time for note 1)	10
Line 4 (frequency index for note 2)	41
Line 5 (time for note 2)	30
Line 6 (frequency index for note 3)	42
Line 7 (time for note 3)	30
Line 8 (frequency index for note 4)	43
Line 9 (time for note 4)	25

To understand the actual logical flow of how this algorithm was programmed, please refer to Figure B.1. For the source code, please refer to Appendix C.1.

3.2 Acquiring User Input

We powered all circuits using the 5V source voltage supplied by the USB terminal. For the preamplifier, we needed to establish the necessary gain. First we connected and powered the microphone using the following circuit.

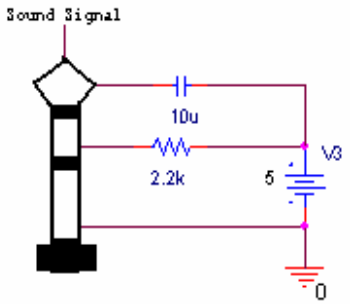

Fig. 3.2. Microphone circuit

We then measured both the microphone and audio line-in output peak-to-peak AC voltage levels (Table 3.2).

TABLE 3.2 Microphone Output Voltage

Singing level	P-P Voltage
Humming (softest)	4.0 mV
Normal singing	6.5 mV
Loud singing	8.1 mV
Line-in stereo*	8.2 mV

*Mid-range keyboard volume

Next we measured the minimum P-P voltage necessary for the DSP to respond. We found this level to be approximately 1V. We then divided the necessary voltage level by the lowest (humming) input voltage to calculate the gain of our preamp using Equation 2.2:

$$A_v = \frac{Vout}{Vin} = \frac{1.0V}{.004V} = 250\frac{V}{V}$$

After determining a gain of 250, we began by establishing resistor values that would achieve the gain. As expressed in Equation 2.2, we needed to choose resistor values R_2 and R_1 that yielded a ratio of 250. We first chose values of 1MΩ and 4kΩ (for R_2 and R_1 respectively) to establish the correct ratio. However, upon testing, we noticed that humming was not loud enough for DSP detection, and we needed to raise the gain. Realizing that our preamplifier's actual gain was less than theoretical gain, we raised our gain to 400 by replacing the 1M resistor with a 1.6M resistor. However, the larger gain produced distorted sounds at higher volumes (loud singing/line-in), so we adjusted the theoretical gain to 300 by decreasing the resistance of R_2 to 1.2MΩ so that the output signal was not distorted.

$$A_v = 1 + \frac{1.2M\Omega}{4k\Omega} = 301\frac{V}{V}$$

Our final circuit for the preamplifier is depicted in Figure 3.3.

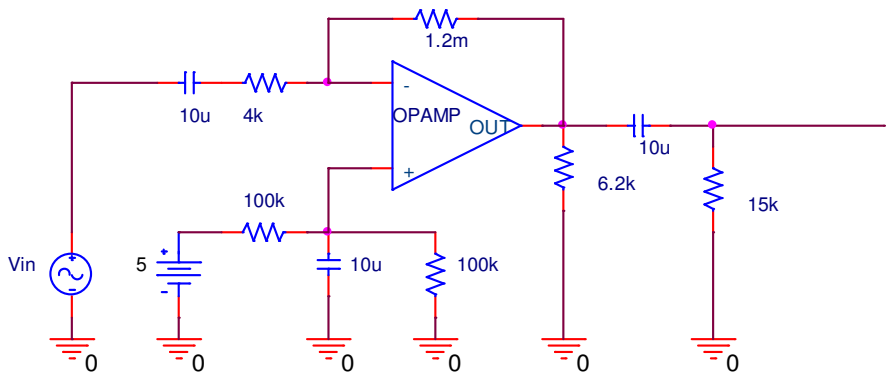

Fig. 3.3. Preamplifier circuit [4]

This yielded the actual 250 V/V gain necessary to produce a 1V P-P voltage level for the DSP, as depicted by Figure 3.3 (middle C gain).

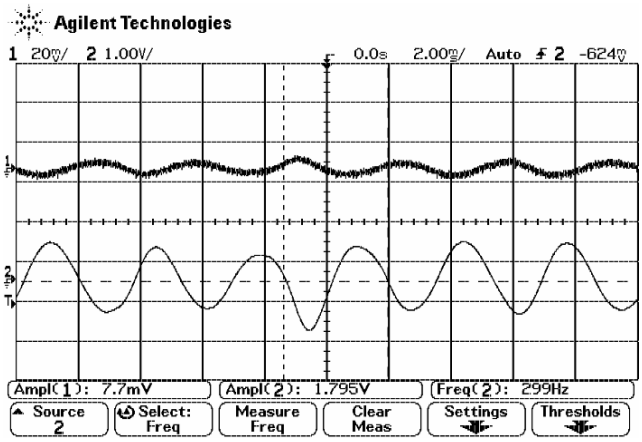

Fig. 3.3. Middle C Gain

3.3 Real-time Pitch Extraction

The two conditions of our DSP algorithm were to recognize frequencies up to 1000 Hz and to resolve inputs separated by 8 Hz. We were able to vary three parameters, including the downsampling factor, FFT size, and low-pass prefilter cutoff frequency.

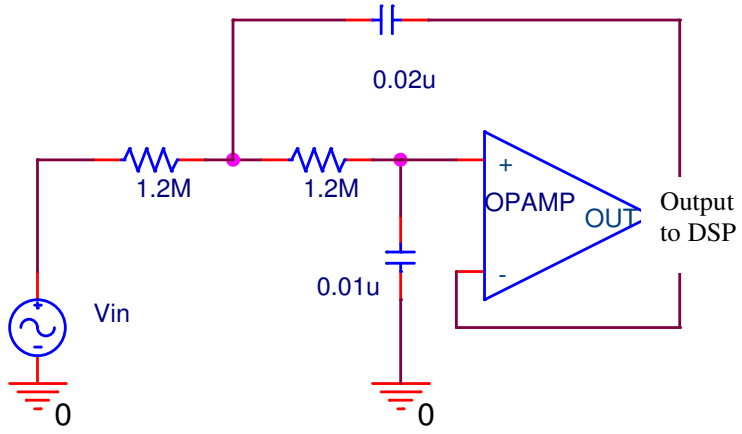

Fig. 3.4. Schematic of Sallen-Key prefilter using an op-amp with cutoff frequency 1000 Hz. [5]

The input signal first passes through a low-pass Sallen-Key filter, Figure 3.4, constructed using a LM324 op-amp. Since we do not expect any of our users to sing above 1000 Hz, we choose that frequency to be our filter cutoff, f_0. From Equation 2.8, we calculated resistors R1 and R2 to be 1.2 MΩ.

The DSP board has a native sampling rate of 44100 Hz. We downsampled this by 8 times to 5512.5 Hz, giving us a Nyquist frequency of 2756.25 Hz. Because our prefilter's cutoff frequency was 1000 Hz, the frequencies above 2756.25 Hz are greatly attenuated and contribute minimal aliasing.

Finally, we chose an FFT of size 4096. Combined with a sampling rate of 5512.5 Hz, each well of our FFT is 1.346 Hz in width. This is enough to resolve a 2.7 Hz difference between notes.

The DSP board continuously takes samples at 44100 Hz and places them in a sample buffer. Once the sample buffer fills, it interrupts our program to transfer those samples into an input buffer of length 4096. At the beginning of our program, we wait for the input buffer to be filled. Then every eighth sample in the input buffer is transferred to a FFT buffer of length 4096. The rest of the positions are simply zero-padded before performing the FFT.

Next, we had to make sure our algorithm could find the pitch of the user's input from the FFT. We weren't able to simply choose the frequency with the highest amplitude in the power spectrum, because harmonics of low frequency notes fall within our filter range, and would often be stronger than the fundamental frequency in power. Our final algorithm starts with finding the strongest frequency of the power spectrum. Then it checks if that result is actually a harmonic. For example, if the highest power peak exists at 750 Hz, we can check if it is actually a 3rd harmonic by dividing by 3 to get 250 Hz, and then looking around 250 Hz for a peak in the power spectrum. The lowest frequency with a peak in the power spectrum is assumed to be the fundamental frequency or the pitch of the user input.

Now that the fundamental frequency is known, we convert it into the closest corresponding piano note based on Table A.1, and send it as output through the DSP board's serial port, provided a couple of conditions are satisfied. The algorithm continually checks the serial input buffer for data from the computer. If the serial input buffer contains a '1', then something will be placed in the serial output buffer. If the note played is above C3 and below C6, and the amplitude of the peak at the fundamental frequency is above a set threshold, then the number of the note is placed in the buffer. Otherwise, the string "00" is placed in the buffer. The entire logical flow of the Real-time Pitch Extraction is included in Figure B.2.

3.4 Serial-to-USB Conversion

The following figure depicts our Serial-to-USB converter circuit.

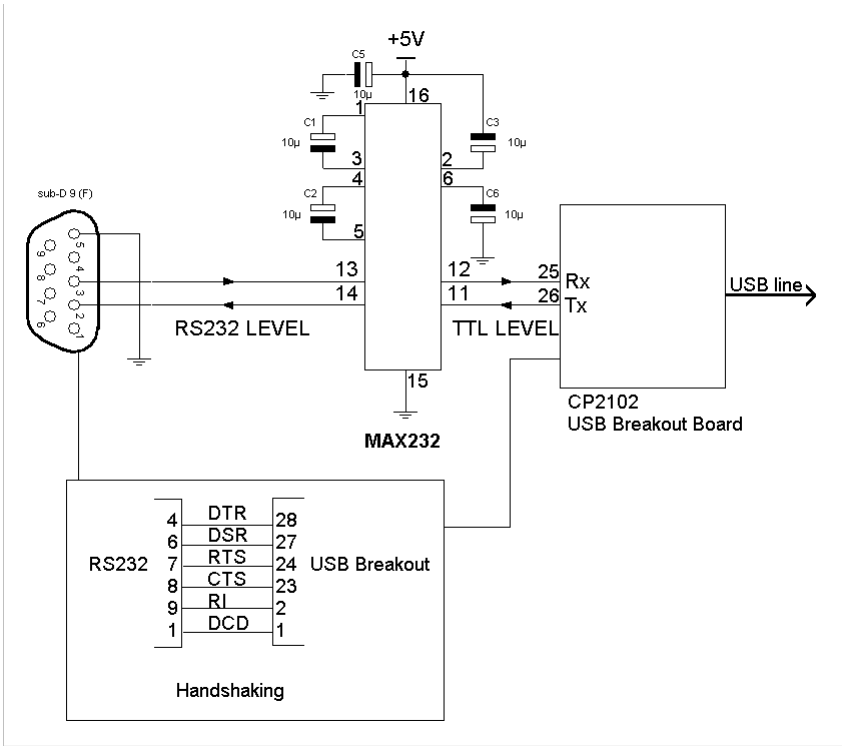

Fig. 3.5. Serial-to-USB converter

We based our serial-to-USB circuit on the datasheets from [7], [8] which provided a rough idea on how to design the components together. For the MAX232, the capacitance for all 5 capacitors used in the circuit could either be 1 uF or 10 uF. We chose 10 uF and the circuit worked well, although 1 uF would have worked just as effectively. We connected non-inverted handshaking signals directly from the RS232 DB-9 to the USB breakout board to establish appropriate communication between the board and the serial input. The RX (receive data) and TX (transmit data) terminals of the DB-9 were transmitted through the MAX232 where the data was converted to TTL then transmitted to the TX and RX terminals, respectively, of the USB breakout board. The breakout board is finally connected to the computer via a USB A-B male/male cord which allows the computer to receive serial data through a USB port.

3.5 Searching the Database

Searching the database involved quite a few steps. The following block diagram (Figure 3.6) briefly describes what was done in order to complete a successful search.

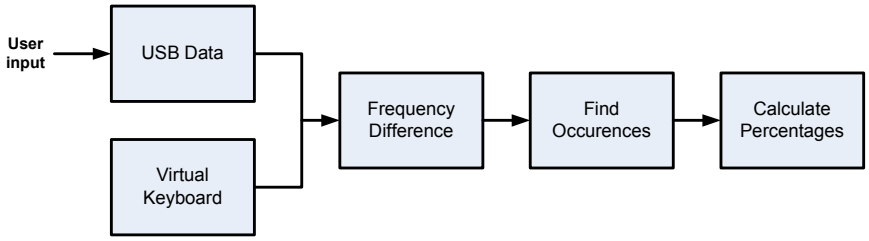

Fig. 3.6. Block Diagram of Search

The following (Figure 3.7) is a picture of the completed search program that interfaces with the user. The entire interface was writing in Visual Basic 6.0.

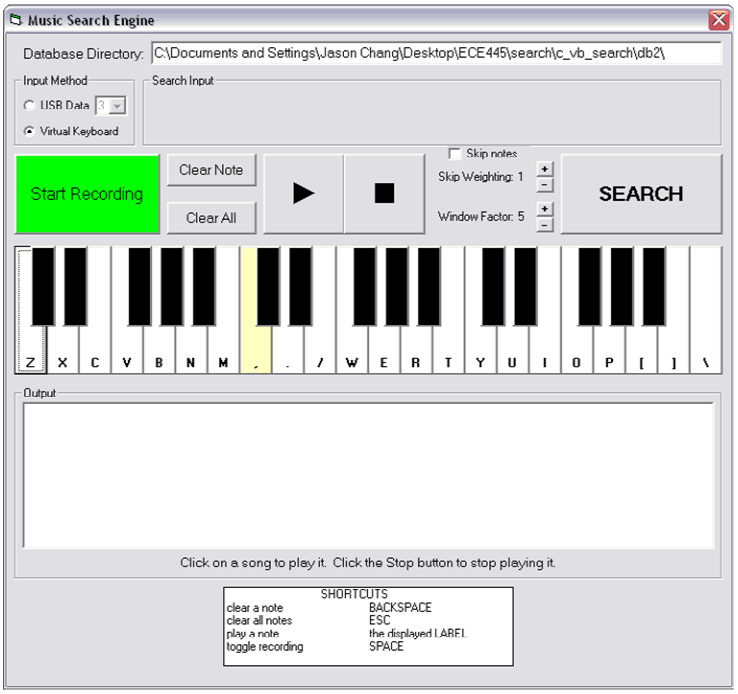

Fig. 3.7. Screenshot of the Search

As you can see, the user can select to input a search string either from the USB Data, or from the virtual keyboard. The first step that we need to do is acquire the search string.

3.5.1 Acquiring User Input from Virtual Keyboard

Getting the data from the virtual keyboard was rather straightforward. Each key on the virtual keyboard was part of an object array (indexed from 0 to 36). When a key was pressed, we subtracted its index from the previous pressed key's index to find the offset. This offset corresponds to exactly what we need to search with. In addition, each time a key is pressed, we also play a sound file that corresponds to that note. While the key is held down, we repeatedly play that note. When the key is released, we play a null sound (a value of 0) to stop the sound. All of these sound playing functions were completely easily through the Windows sndPlaySound API.

It should be noted that there was actually a considerable amount of work devoted to developing these key sounds. We created the sounds in Matlab using wavwrite. However, problems arose such that if we created a sound file of an arbitrary length and then repeated it, a high frequency popping sound would occur in between the repetitions. To understand why this occurs, we look at the following sampled sine wave in Figure 3.8.

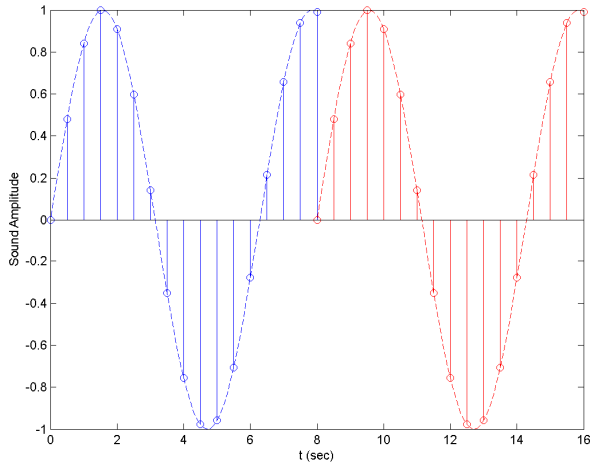

Fig. 3.8. Sampled Sine Wave (before period finishes)

As you can see, when the first repetition of the sound wave finishes, there is a quick jump from its current value to the starting zero of the next repetition. The sudden jump causes the popping sounds that were described previously. To resolve this issue, instead of making the sounds an arbitrary length, we extend it so that it finishes on the end of a period. Therefore, it will look like the following wave in Figure 3.9.

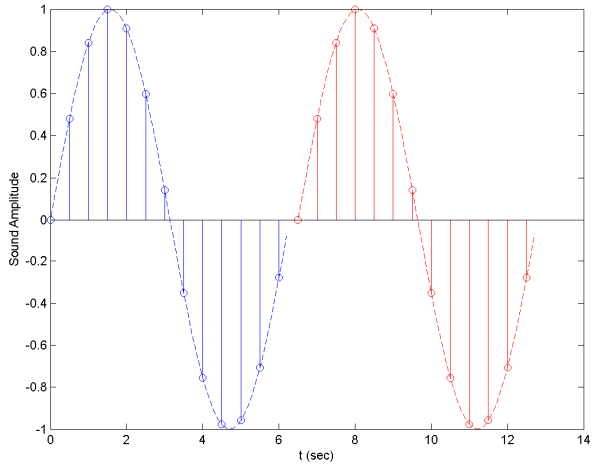

Fig. 3.9. Sampled Sine Wave (full period)

3.5.2 Acquiring User Input from USB

The other way to acquire the user inputted search string is via USB communications. The drivers for the USB device create a virtual serial port in which it communicates to and from. Therefore, using the MSComm device provided with Visual Basic 6.0, we are easily able to communicate with the DSP hardware. As stated in the previous section

When the user clicks on "Start Recording", we begin polling the DSP for data every 100ms. To poll the DSP, the search program sends the ASCII code for '1', and then waits to receive two ASCII codes corresponding to the frequency index. The reason that we poll the DSP for data is described in more detail in Section 3.3. However, it is important to note that each time the DSP is polled, the current frequency index that is being inputted is sent to the computer. When the current frequency index is different from the previous one, it indicates that the note just changed, so we store the value. Once the user decides that the entire string has been inputted, he or she can click on the "Stop Recording" button to tell the DSP to stop. Once this is done, acquiring data from the USB port is complete.

3.5.3 Search Algorithm

After the input acquisition is completed, we then begin searching the actual database for the string. Because speed was an important factor in writing the search algorithm, it was optimized in C++ instead of coding it with Visual Basic 6.0. This involved building C++ dynamic link libraries to interface with VB6. For more information on calling C++ functions from VB6, please refer to [9]. The search works as the following.

1) Read and store the entire song from the database so we can access the information quickly.

2) Calculate the average length of a note in the song, using this as an indication of the tempo. Do this by looking at the middle minute of the song (where m is the middle sample).
   i) Calculate the total time (t) of the middle minute (where t_i is the length of note i).
   $$t = \sum_{i=m-3000}^{m+3000} t_i \tag{3.1}$$
   ii) Calculate the total number of notes (k) within the middle minute.
   $$k = \sum_{n_{m-3000}}^{n_{m+3000}} 1 \tag{3.2}$$
   ii) Calculate the average length of a note (<t>) within the middle minute.
   $$t_{avg} = \langle t \rangle = \frac{t}{k} \tag{3.3}$$

3) Define a window to search for the note (where β is a window weighting factor, usually $\beta \approx 3$).
   $$W_n = \beta t_{avg} \tag{3.4}$$

4) Define a window to search for the entire input (where l is the length of the input).
   $$W_s = 1.5 t_{avg} l \tag{3.5}$$

5) Start at each note of a song, and search for each note. If the next note is not found within W_n of the last note, then give up on starting here, and start at the next note.

6) When the entire string is found, if the time for the string is less than W_n then increase the number of occurrences (n_i) with Equation 2.12

7) Calculate the percentage that each song is correct using Equation 2.11.

The logical flow of the search algorithm can be found in Figure B.3.

In addition to being able to perform the search (and choose the search parameters α and β), the user also has a few other options to aid in the search process. Specifically, the user can do the following things:

1) Clear the last inputted note
2) Clear the entire search string
3) Playback the search string
4) Playback songs returned by the search

These features were all straightforwardly implemented, and can be found in the source code Appendix C.3.

4. DESIGN VERIFICATION

One of the most important things in our system was to test it properly. The following tests validated the success of each subproject as well as validating our system as a whole.

4.1 Building the Database

To test that our database building algorithm was working properly, we created sound files that contained a simple tune (such as "Mary Had a Little Lamb"), and made sure that the algorithm was tracking the input perfectly. To our demise, the database did track the voice perfectly when it existed. However when the voice stopped for a split second, the algorithm would generate noise for when the voice was not there. The result was a semi-noisy signal, most of which was cleaned by our filtering techniques.

The following figure is a simulation we ran to test our database algorithm on the C Major scale.

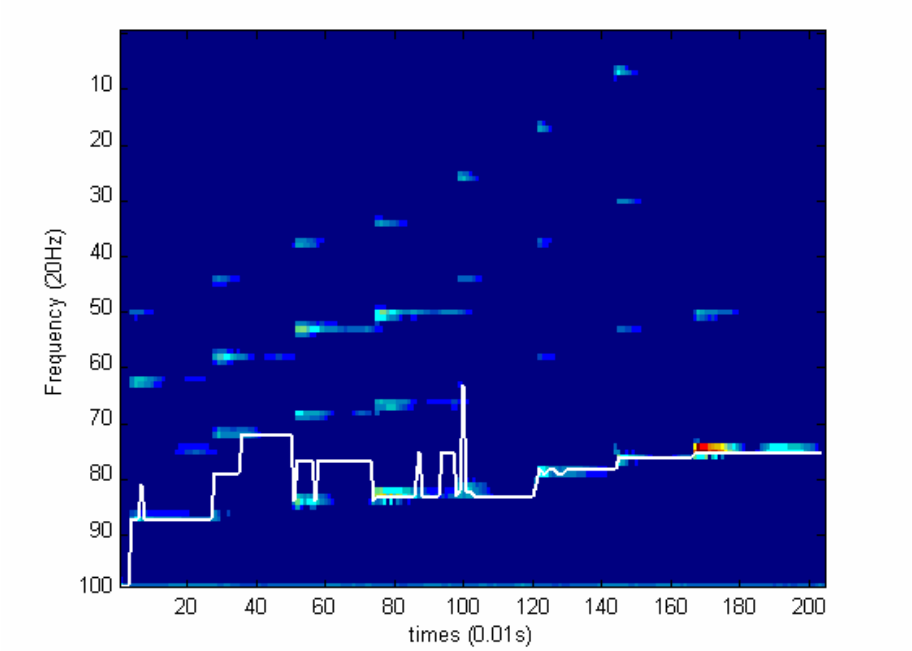

Fig. 4.1. Simulation of Building the Database on a C Major Scale

4.2 Acquiring User Input

We tested the prefilter/preamplifier circuit by measuring both the input and output voltage levels and capturing these on the oscilloscope. After verifying that the amplification was working for the frequency of middle C (262 Hz), we tested a wide range of frequencies and vocal volumes (Table 4.1) to ensure that the combination of our prefilter and preamp would produce the necessary voltage level for the DSP.

TABLE 4.1 Preamp & Prefilter I/O relationship

$V_{in,\ p\text{-}p}$	f (Hz)	$V_{out,\ p\text{-}p}$
4.0 mV	150	1.0 V
	300	1.0 V
	500	1.0 V
	1000	0.5 V
	2000	0.1 V
6.5 mV	150	1.6 V
	300	1.6 V
	500	1.6 V
	1000	0.8 V
	2000	0.2 V
8.0 mV	150	2.0 V
	300	2.0 V
	500	2.0 V
	1000	1.0 V
	2000	0.2 V

Sample scope captures are displayed in Figures 4.2-4.5.

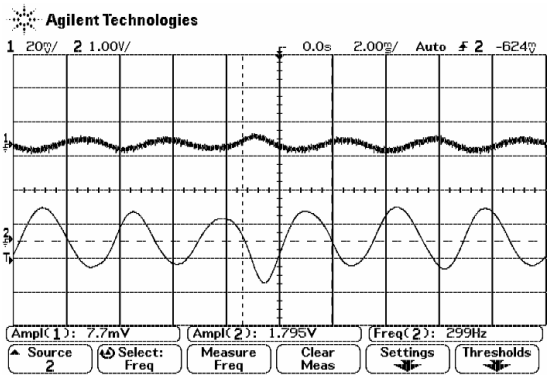

Fig. 4.2. Preamplifier & Prefilter 300Hz (Gain = 250)

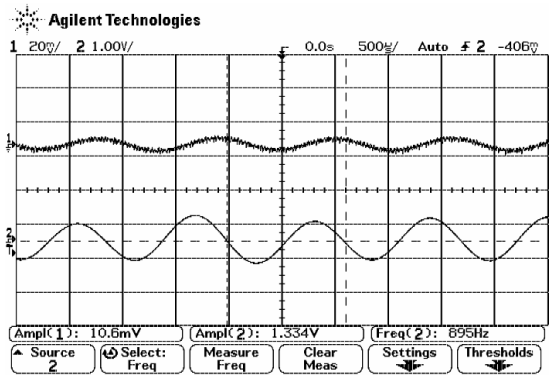

Fig. 4.3. Preamplifier & Prefilter 900Hz (Gain = 126)

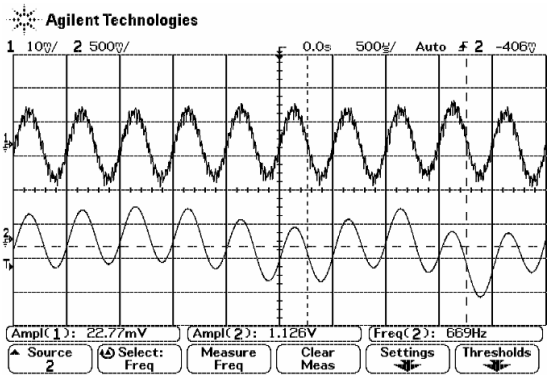

Fig. 4.4. Preamplifier & Prefilter 2kHz (Gain = 32.6)

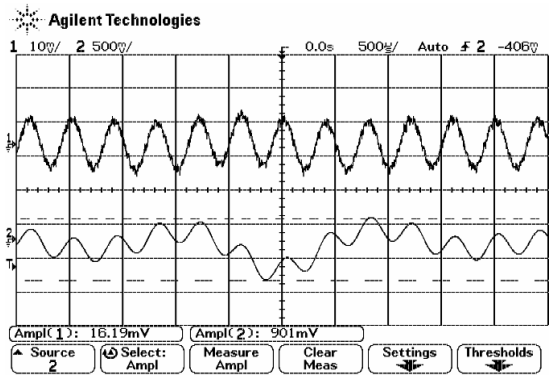

Fig. 4.5. Preamplifier & Prefilter 2.5kHz (Gain = 18.5)

4.3 Real-time Pitch Extraction

In testing this part of our design, we wanted to make sure our setup would allow us to distinguish between every note we expected to receive. We also wanted to make sure to always output the fundamental frequency and never a harmonic.

At first, a function generator was used to simulate input to the DSP board. This type of input does not contain any harmonics, but it did allow us to make sure each note, including the low notes, could be distinguished. We set the DSP to continuously output the note of each sample through the serial port, and monitored the output using hyperterminal on the computer.

The human singing voice, however, contains plenty of high frequency harmonics (above the raw sampling rate), and this was a huge issue when we tested using the microphone as input. Because of this, we had to build an anti-aliasing prefilter, redesign our FFT parameters, and construct an algorithm to find the fundamental frequency.

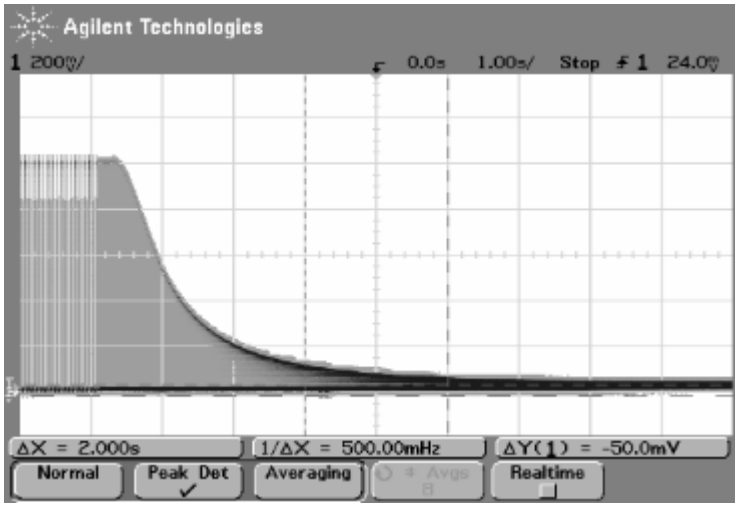

Fig. 4.6. Frequency sweep of prefilter, 0 – 10 kHz, step size 50 Hz. x-axis division 1 kHz.

We setup a frequency sweep on the frequency generator from 0 – 10 kHz, with step size 50 Hz, and connected it to the prefilter. Signals passing through the prefilter begin to be attenuated at around 1.5 kHz, as shown in Figure 4.6.

Now, when we tested the design by singing into the microphone input, we would always get the note corresponding to our fundamental frequency. However, one problem we faced was that, as untrained singers, we could not properly hold a note for a long duration. Often the output would waver between neighboring notes. This problem is not in our algorithm, but rather it is a problem of the original input. We did not have this issue when testing the line-in of a digital keyboard.

4.4 Serial to USB

We began by measuring the input and output data terminals of the MAX232. The following figure shows that the MAX232 is indeed converting the signal to TTL logic. As expected, the input signal (the lower waveform) is inverted and then amplified to 5V.

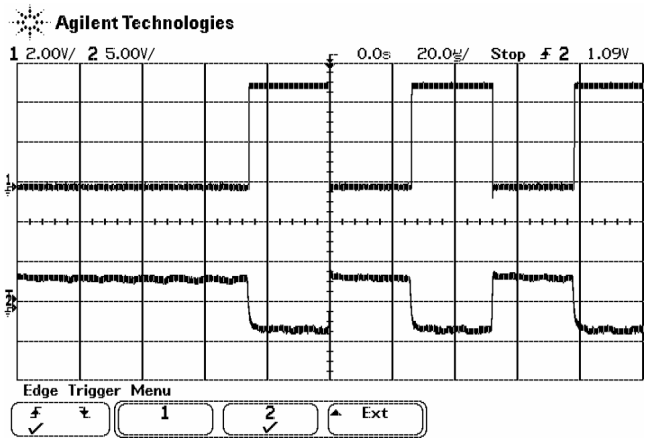

Fig. 4.7. MAX232 scope plot.

After determining that the chip was working properly, we tested the serial-to-USB conversion using Windows HyperTerminal. We entered serial data on one computer using a regular keyboard and output the data from the serial port to the serial to USB converter. We then observed if what we were typing showed up on the other PC's HyperTerminal. For example if we typed "asdf", we would expect "asdf" to be displayed on the HyperTerminal window of the connected computer. When this happened, we were certain that our serial-to-USB converter was working properly.

4.5 Searching the Database

To test the search algorithm, we hardcoded entries into our database and searched for those. Specifically, we manually created database entries for common tunes such as "Mary Had a Little Lamb", "Twinkle Twinkle Little Star", "Take Me Out to the Ballgame", etc. We tested each song by searching small portion tunes from it, and were able to identify each song as the first place in the results. A sample of the hardcoded "Mary Had a Little Lamb" database file is given here in Table 4.1.

TABLE 4.2 Mary Had a Little Lamb

Line #	Value
1	272
2	44
3	10
4	42
5	10
6	40
7	10
8	42
9	10
10	44
11	40
12	42
13	40
14	44
15	10
…	…

After creating these entries, we then proceeded to search for them. We tested all the advanced features of the search including skipping notes and windowing factors. Specific inputs were chosen to test each feature, and the search was slowly debugged and verified.

4.6 Conclusions

We tested many tunes from many of the songs in our database. The biggest goal was to retrieve the actual song. For each of the following songs, two search strings were tested and the better results are shown. This is an accurate representation of a user actually using the database, because many times the user can remember more than just one part of a song. The testing group was chosen to span a wide variety of music to not put weight on any particular type. The following table shows the results of testing our system.

TABLE 4.3 System Testing

Song Name	Search String Length	Best Search Ranking (with & without skip)	Search Time (sec)
Deep Blue Something - Breakfast at Tiffany's	8	1	0.18
Paul McCartney - Yesterday	10	2	0.17
Mamas and Papas - Puff the Magic Dragon	6	3	0.19
Norah Jones - Come Away With Me	12	2	0.20
Dido - White Flag	15	1	0.25
Paulina Rubio – Ni Una Sola Palbra	16	2	0.30
Red Hot Chili Peppers – Otherside	10	NOT FOUND	0.20
Josh Kelley – Perfect 10	9	1	0.18

Using these results, we can generalize the robustness and effectiveness of our search engine. We need to consider two categories for our design specifications, the search time and the search effectiveness.

When considering the search time, we have obviously met the specification that all searches must be completed within 10 sec. However it is also useful to consider the search time with respect to the length of the search input. It is straightforward to show that the search algorithm is directly proportional to the length of the search string. The following table shows the average search time per input note.

TABLE 4.4 Search Time per Note

Song Name	Search String Length	Search Time	Search Time / Length
Deep Blue Something - Breakfast at Tiffany's	8	0.18	0.0225
Paul McCartney - Yesterday	10	0.17	0.0170
Mamas and Papas - Puff the Magic Dragon	6	0.19	0.0317
Norah Jones - Come Away With Me	12	0.20	0.0167
Dido - White Flag	15	0.25	0.0167
Paulina Rubio – Ni Una Sola Palbra	16	0.30	0.0188
Red Hot Chili Peppers – Otherside	10	0.20	0.0200
Josh Kelley – Perfect 10	9	0.18	0.0200
		Average Search Time per Note	0.0204 sec

The other design specification we need to consider is the success rate of our search. We will rate a songs success rate as how high it is places. Because our database contained 35 songs, we will divide each songs by the inverse highest ranking by the total number of songs to find its success rate. In other words, the success rate of each song is given by the following equation.

$$Success = 100 \times \frac{35 - rank + 1}{35} \tag{4.1}$$

The following table summarizes the success rates.

TABLE 4.5 Success Rate

Song Name	**Best Search Ranking** (with & without skip)	**Success Rate** (%)
Deep Blue Something - Breakfast at Tiffany's	1	100.00
Paul McCartney - Yesterday	2	97.14
Mamas and Papas - Puff the Magic Dragon	3	94.29
Norah Jones - Come Away With Me	2	97.14
Dido - White Flag	1	100.00
Paulina Rubio – Ni Una Sola Palbra	2	97.14
Red Hot Chili Peppers – Otherside	NOT FOUND	0.00
Josh Kelley – Perfect 10	1	100.00
	Average Success Rate	85.7%

Our design specifications were to have a 90% success rate or better. Clearly we did not meet this design specification. This is due highly to the poor output of the pitch extraction algorithm in building the database. Every other component was tested separately to be almost entirely perfect. The only downfall in our entire system was this algorithm because it performed very poorly for low SNRs. Because we did not focus on the development of the actual algorithm for this class, it was hard to fix this problem.

5. COST

Since our product is marketed towards the general public, we wanted to keep our costs low. Our most expensive component was the DSP, which pushed the total cost to $133.26 (Table 5.1). This was unavoidable due to the complexity of the board. The cost of all other components combined came to approximately $70.

5.1 Cost Analysis

TABLE 5.1 Cost of Parts

Part	Cost	Number Used	Total Cost
Resistors	$0.05	20	$1.00
Capacitors	$0.10	25	$2.50
USB Breakout	$19.95	1	$19.95
MAX232	$3.83	1	$3.83
Project box	$3.00	1	$3.00
1.5” x 2.5“ Circuit board	$2.50	2	$5.00
3.5 mm Male/Male stereo cable	$5.00	1	$5.00
USB B to A Male/Male cable	$6.00	1	$6.00
Serial Male/Male cable	$6.00	1	$6.00
Microphone	$15.00	1	$15.00
Screws, bolts, stands	$0.10	30	$3.00
DSP (TI TMS320C54x)	$61.69	1	$61.69
LM324	$1.29	1	$1.29
Total			**$133.26**

Labor

$35/hr x 2.5 x 20 hrs/week x 14 weeks = $24,500 / person
$24,500 x 3 persons = **$73,500** total

6. CONCLUSIONS

6.1 Accomplishments

We accomplished three out of our four specifications. These consisted of containing all hardware within 1 ft^2, limiting search time to less than 10 seconds even with modifications like window sizing and note skipping, and also building a circuit solely powered by the 5V from the USB line. The hardware containment was achieved through strategic placement of circuitry boards and wires inside a 3x3x6 inch box, with holes drilled for easy user access to input/output terminals. A short search time was accomplished by switching from VB to C++ as well as using efficient search algorithms and techniques. Finally, because the computer supplied an ample amount of current (>100mA), we were able to achieve the 5V USB specification by using components that required exactly 5V as their voltage source [4], [7], [8].

6.2 Uncertainties

The fourth specification was difficult to measure quantitatively, and for the better songs ('better' referring to accurately encoded songs), the 90% rate was surpassed with perfect inputs. However, for certain songs the success rate fell below this threshold due to the skipping of key notes during song encoding. Fortunately, for every song there was at least one correct melody snippet that would yield a top match for that song, but in many cases such a melody snippet took many trials and familiarity with the song to figure out. Because search results for each song varied so greatly, we were unable to find a conclusive pattern in determining which part of a song or what kind of melody should have been used to yield the best results.

6.3 Ethical Considerations

There was one main ethical consideration we took into account for this project. We affirm that all music obtained for our database was obtained legally either from CDs we owned or in rarer cases, iTunes. The music industry has faced a serious dilemma with illegal music file sharing over the past decade, and we have taken all necessary steps not to promote this problem. The Music Search Engine is intended solely for the use of searching for song titles/artists and not for the promotion of file sharing or illegal downloads.

6.4 Future Work

There are many improvements that could be made to our project given more time. The most obvious improvement would be decreasing the time it took to encode each song from the current time of 4-5 hours for a 4 minute song. The algorithm for the encoding was extremely intricate and complex so additional research would be required in order to take steps towards improving speed. On a similar note, we would like to improve the accuracy of the encoding algorithm. In some cases there were important vocal notes that were not captured while being replaced with unwanted noise. Improvement in this area would also require much more research and experimentation. Finally, due to the size and expense of the DSP board, we could possibly eliminate the use of the DSP by converting this step to a software component. This would cut down on costs and leave our project with a much smaller hardware component which is obviously more appealing to the consumer. Clearly there is still much work that can be to improve the Music Search Engine, but the progress we have made in this field is extremely promising.

APPENDIX A – Piano Key Frequencies

Table A.1 is the mapping between notes, frequencies, and frequency indices.

TABLE A.1 Piano Key Frequencies [9]

Key number	Note name	Frequency (Hz)	Key number	Note name	Frequency (Hz)
88	C8	4186.01	44	E4	329.628
87	B7	3951.07	43	D#4/Eb4	311.127
86	A#7/Bb7	3729.31	42	D4	293.665
85	A7	3520	41	C#4/Db4	277.183
84	G#7/Ab7	3322.44	40	C4	261.626
83	G7	3135.96	39	B3	246.942
82	F#7/Gb7	2959.96	38	A#3/Bb3	233.082
81	F7	2793.83	37	A3	220
80	E7	2637.02	36	G#3/Ab3	207.652
79	D#7/Eb7	2489.02	35	G3	195.998
78	D7	2349.32	34	F#3/Gb3	184.997
77	C#7/Db7	2217.46	33	F3	174.614
76	C7	2093	32	E3	164.814
75	B6	1975.53	31	D#3/Eb3	155.563
74	A#6/Bb6	1864.66	30	D3	146.832
73	A6	1760	29	C#3/Db3	138.591
72	G#6/Ab6	1661.22	28	C3	130.813
71	G6	1567.98	27	B2	123.471
70	F#6/Gb6	1479.98	26	A#2/Bb2	116.541
69	F6	1396.91	25	A2	110
68	E6	1318.51	24	G#2/Ab2	103.826
67	D#6/Eb6	1244.51	23	G2	97.9989
Key number	**Note name**	**Frequency (Hz)**	**Key number**	**Note name**	**Frequency (Hz)**
66	D6	1174.66	22	F#2/Gb2	92.4986
65	C#6/Db6	1108.73	21	F2	87.3071
64	C6	1046.5	20	E2	82.4069
63	B5	987.767	19	D#2/Eb2	77.7817
62	A#5/Bb5	932.328	18	D2	73.4162
61	A5	880	17	C#2/Db2	69.2957
60	G#5/Ab5	830.609	16	C2	65.4064
59	G5	783.991	15	B1	61.7354
58	F#5/Gb5	739.989	14	A#1/Bb1	58.2705
57	F5	698.456	13	A1	55
56	E5	659.255	12	G#1/Ab1	51.913
55	D#5/Eb5	622.254	11	G1	48.9995
54	D5	587.33	10	F#1/Gb1	46.2493
53	C#5/Db5	554.365	9	F1	43.6536
52	C5	523.251	8	E1	41.2035
51	B4	493.883	7	D#1/Eb1	38.8909
50	A#4/Bb4	466.164	6	D1	36.7081
49	A4	440	5	C#1/Db1	34.6479
48	G#4/Ab4	415.305	4	C1	32.7032
47	G4	391.995	3	B0	30.8677
46	F#4/Gb4	369.994	2	A#0/Bb0	29.1353
45	F4	349.228	1	A0	27.5

APPENDIX B – Software Flowcharts

Figure B.1 is the logical flowchart for building the database.

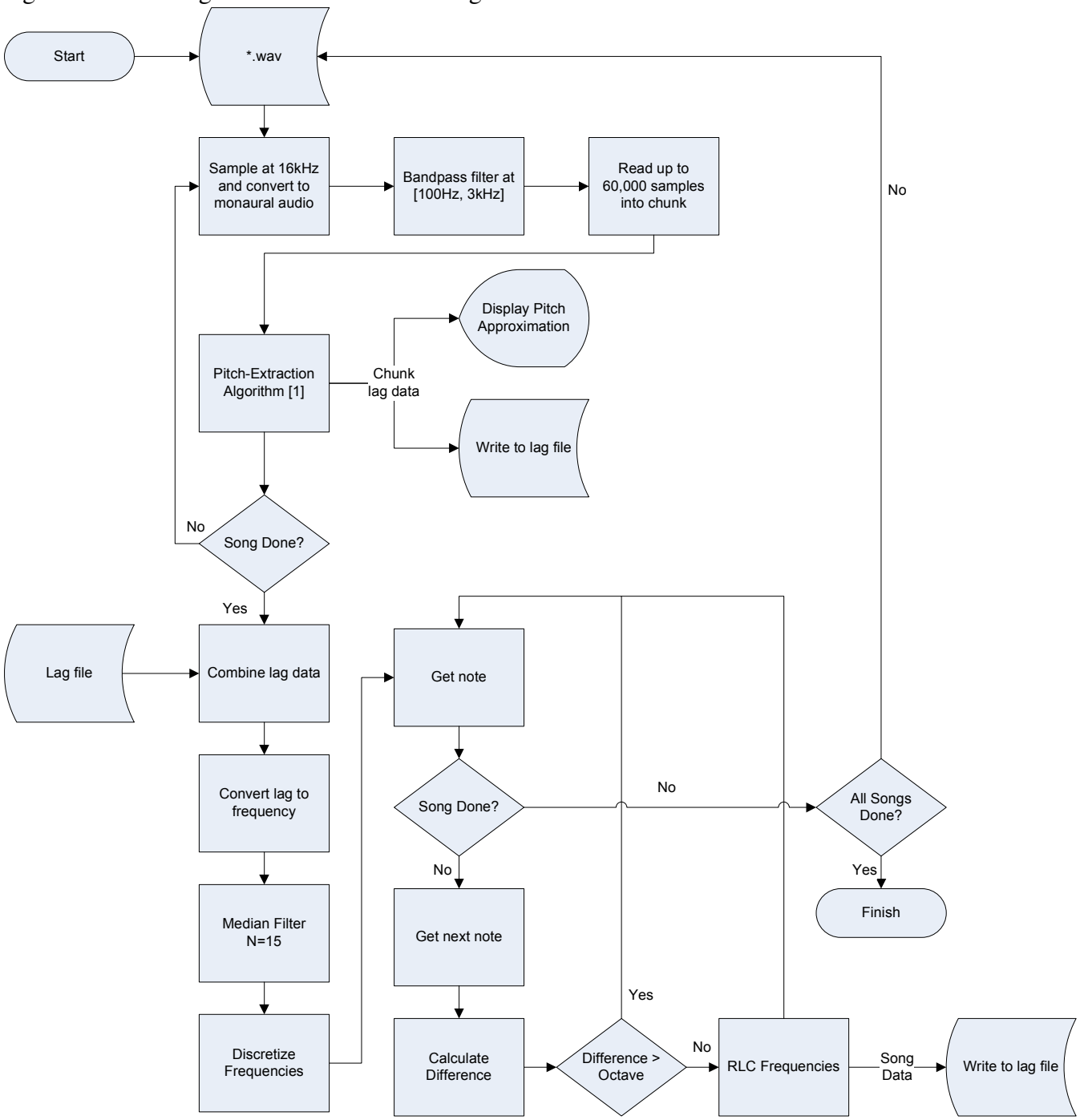

Fig. B.1. Flowchart of Building the Database

Figure B.2 is the logical flowchart for the Real-time Pitch Extraction.

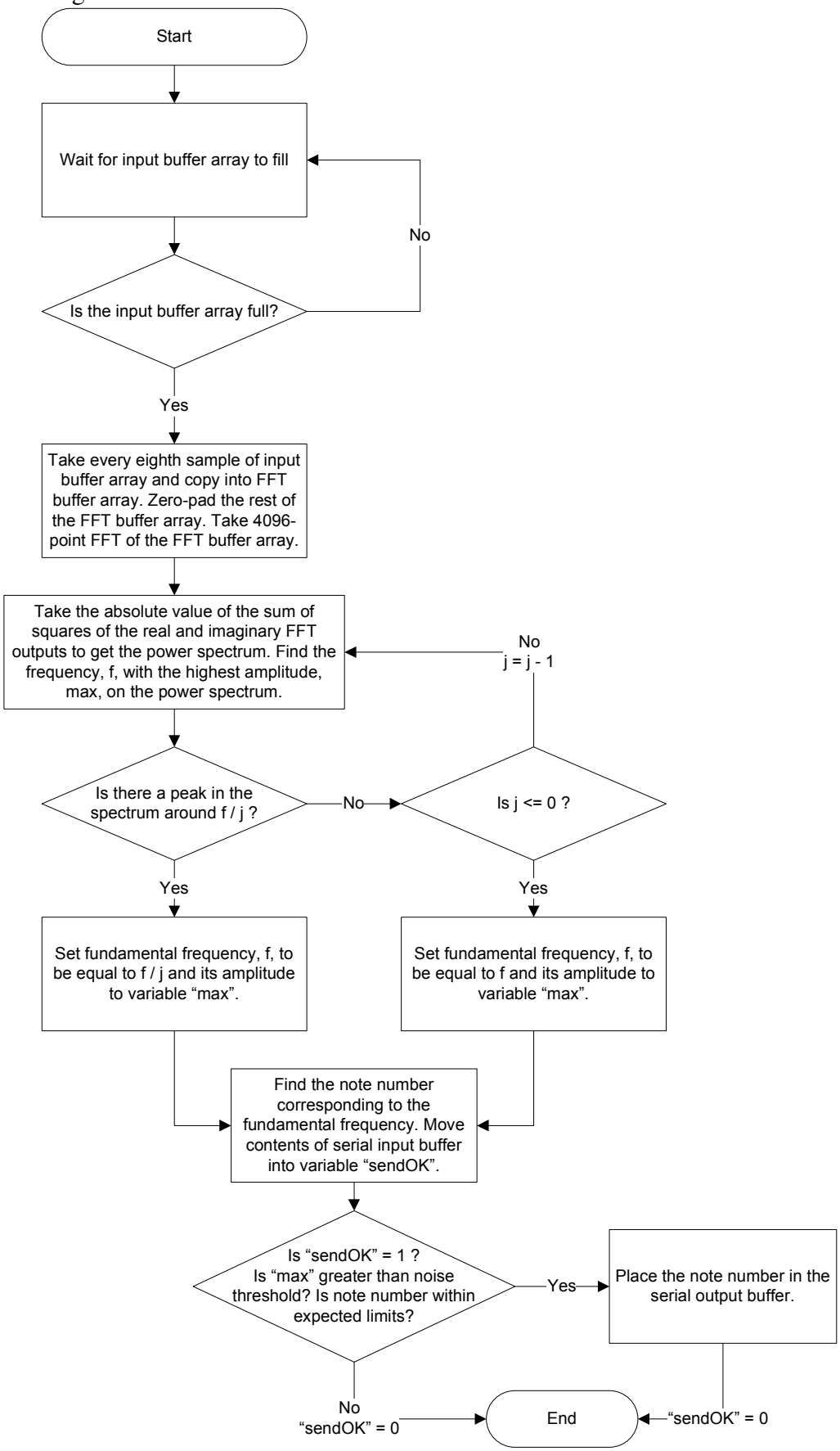

Fig. B.2. Flowchart of Real-time Pitch Extraction

Figure B.3 is the logical flowchart for the Search Algorithm.

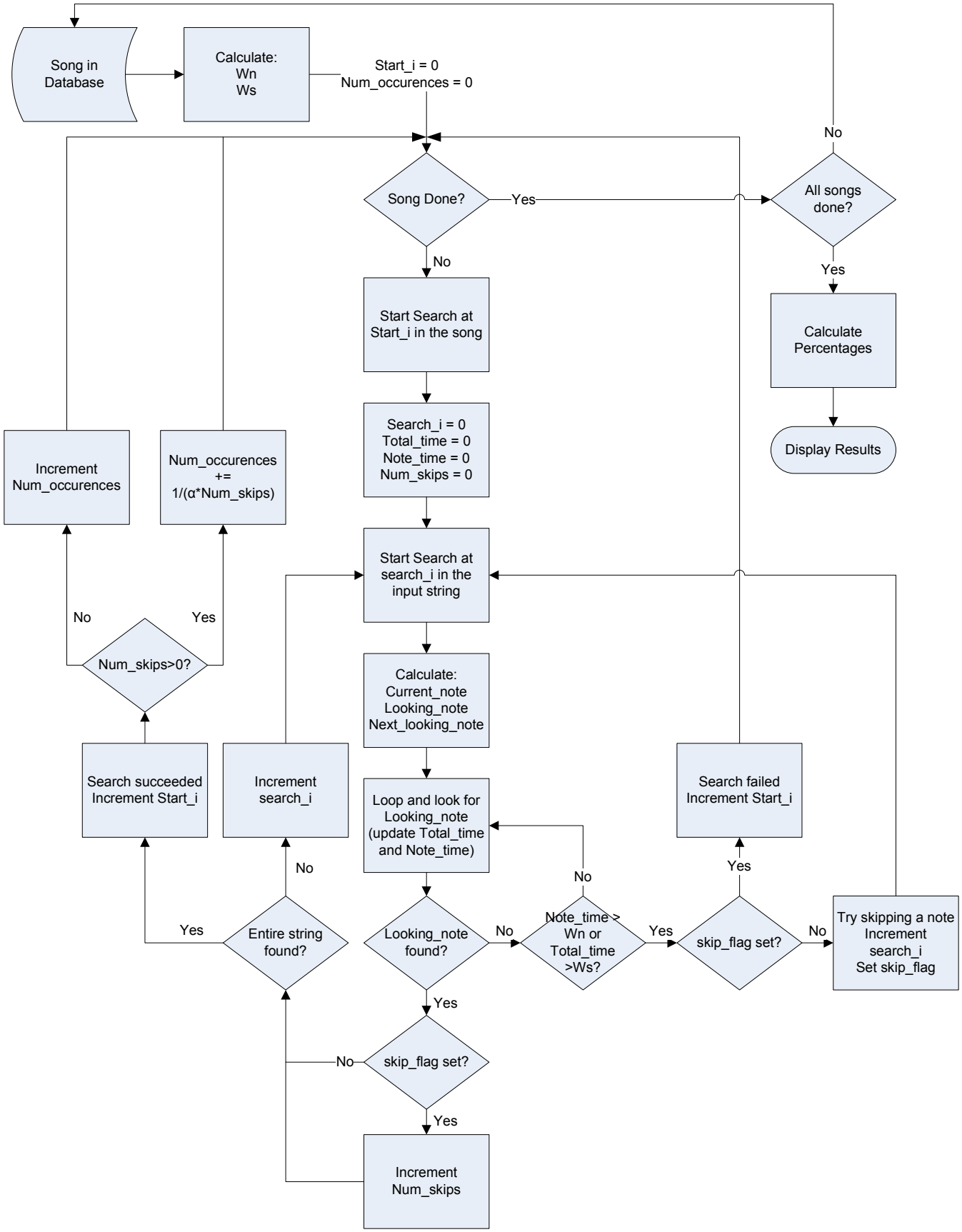

Fig. B.3. Flowchart of Search Algorithm

APPENDIX C – Code

APPENDIX C.1 – Building the Database

APPENDIX C.1.1 – main.m

```
function varargout = main(varargin)
% MAIN M-file for main.fig
%      MAIN, by itself, creates a new MAIN or raises the existing
%      singleton*.
%
%      H = MAIN returns the handle to a new MAIN or the handle to
%      the existing singleton*.
%
%      MAIN('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in MAIN.M with the given input arguments.
%
%      MAIN('Property','Value',...) creates a new MAIN or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before main_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to main_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

global start;
% Edit the above text to modify the response to help main

% Last Modified by GUIDE v2.5 29-Mar-2007 18:19:17

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @main_OpeningFcn, ...
                   'gui_OutputFcn',  @main_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before main is made visible.
function main_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to main (see VARARGIN)

% Choose default command line output for main
handles.output = hObject;
```

```
% Update handles structure
guidata(hObject, handles);

% This sets up the initial plot - only do when we are invisible
% so window can get raised using main.
if strcmp(get(hObject,'Visible'),'off')
    plot(rand(5));
end

% UIWAIT makes main wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = main_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
axes(handles.axes1);
cla;

popup_sel_index = get(handles.popupmenu1, 'Value');
switch popup_sel_index
    case 1
        plot(rand(5));
    case 2
        plot(sin(1:0.01:25));
    case 3
        comet(cos(1:.01:10));
    case 4
        bar(1:10);
    case 5
        plot(membrane);
    case 6
        surf(peaks);
end


% --------------------------------------------------------------------
function FileMenu_Callback(hObject, eventdata, handles)
% hObject    handle to FileMenu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function OpenMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to OpenMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
file = uigetfile('*.fig');
if ~isequal(file, 0)
```

```
    open(file);
end

% --------------------------------------------------------------------
function PrintMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to PrintMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
printdlg(handles.figure1)

% --------------------------------------------------------------------
function CloseMenuItem_Callback(hObject, eventdata, handles)
% hObject    handle to CloseMenuItem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
selection = questdlg(['Close ' get(handles.figure1,'Name') '?'],...
                     ['Close ' get(handles.figure1,'Name') '...'],...
                     'Yes','No','Yes');
if strcmp(selection,'No')
    return;
end

delete(handles.figure1)


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

set(hObject,        'String',        {'plot(rand(5))',        'plot(sin(1:0.01:25))',
'comet(cos(1:.01:10))', 'bar(1:10)', 'plot(membrane)', 'surf(peaks)'});

% --- Executes on selection change in popupmenu3.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu3 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu3



% --- Executes on button press in all_songs_radio.
function all_songs_radio_Callback(hObject, eventdata, handles)
% hObject    handle to all_songs_radio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if (get(hObject, 'Value') == 0)
    set(hObject, 'Value', 1);
end
```

```
% disable all other things
some_songs_radio = findobj(gcf, 'Tag', 'some_songs_radio');
set(some_songs_radio, 'Value', 0);
start_index = findobj(gcf, 'Tag', 'edit3');
stop_index = findobj(gcf, 'Tag', 'edit4');
set(start_index, 'Enable', 'off');
set(stop_index, 'Enable', 'off');
% Hint: get(hObject,'Value') returns toggle state of all_songs_radio


% --- Executes on button press in some_songs_radio.
function some_songs_radio_Callback(hObject, eventdata, handles)
% hObject    handle to some_songs_radio (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if (get(hObject, 'Value') == 0)
    set(hObject, 'Value', 1);
end

start_index = findobj(gcf, 'Tag', 'edit3');
stop_index = findobj(gcf, 'Tag', 'edit4');
set(start_index, 'Enable', 'on');
set(stop_index, 'Enable', 'on');

% disable other things
all_songs_radio = findobj(gcf, 'Tag', 'all_songs_radio');
set(all_songs_radio, 'Value', 0);
% Hint: get(hObject,'Value') returns toggle state of some_songs_radio


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a double


% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
```

```
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a double


% --- Executes on button press in stop_btn.
function stop_btn_Callback(hObject, eventdata, handles)
% hObject    handle to stop_btn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global song_choice start;
start = 0;
start_btn = findobj(gcf, 'Tag', 'start_btn');
set(start_btn, 'Enable', 'on');
set(hObject, 'Enable', 'off');

start_index = findobj(gcf, 'Tag', 'edit3');
stop_index = findobj(gcf, 'Tag', 'edit4');
some_songs_radio = findobj(gcf, 'Tag', 'some_songs_radio');
all_songs_radio = findobj(gcf, 'Tag', 'all_songs_radio');

if (max(size(song_choice)) == 0 || song_choice == 0)
    set(start_index, 'Enable', 'off');
    set(stop_index, 'Enable', 'off');
    set(some_songs_radio, 'Enable', 'off');
    set(all_songs_radio, 'Enable', 'on');
else
    set(start_index, 'Enable', 'on');
    set(stop_index, 'Enable', 'on');
    set(some_songs_radio, 'Enable', 'on');
    set(all_songs_radio, 'Enable', 'off');
end

% --- Executes on button press in start_btn.
function start_btn_Callback(hObject, eventdata, handles)
% hObject    handle to start_btn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global song_choice start;

start = 1;

% Find all the objects
stop_btn = findobj(gcf, 'Tag', 'stop_btn');
start_index = findobj(gcf, 'Tag', 'edit3');
stop_index = findobj(gcf, 'Tag', 'edit4');
some_songs_radio = findobj(gcf, 'Tag', 'some_songs_radio');
all_songs_radio = findobj(gcf, 'Tag', 'all_songs_radio');
```

```
% Get the current state
if (get(all_songs_radio, 'Value') == 1)
    song_choice = 0;
else
    song_choice = 1;
end

% Enable the stop button
set(stop_btn, 'Enable', 'on');

% Disable everything else
set(hObject, 'Enable', 'off');
set(start_index, 'Enable', 'off');
set(stop_index, 'Enable', 'off');
set(some_songs_radio, 'Enable', 'off');
set(all_songs_radio, 'Enable', 'off');

% Find all the display objects
song_name = findobj(gcf, 'Tag', 'song_name');
song_start_time = findobj(gcf, 'Tag', 'song_start_time');
song_done_time = findobj(gcf, 'Tag', 'song_done_time');
chunk_start_time = findobj(gcf, 'Tag', 'chunk_start_time');
chunk_done_time = findobj(gcf, 'Tag', 'chunk_done_time');
percent_done = findobj(gcf, 'Tag', 'percent_done');

% Populate List of songs
all_files = dir('wavs\*.wav');

if (song_choice == 0)
    song_index = [1 length(all_files)];
else
    if (get(stop_index, 'String') == 'end')
        song_index = [str2num(get(start_index, 'String')), ...
                      length(all_files)];
    else
        song_index = [str2num(get(start_index, 'String')), ...
                      str2num(get(stop_index, 'String'))];
    end
end

for i=song_index(1):song_index(2)
    if (start == 0)
        break;
    end

    % Get the name of the .wav file
    file = all_files(i).name;
    % Drop the .wav extension
    file = file(1 : length(file)-4);

    set(song_name, 'String', file);
    set(song_start_time, 'String', datestr(now));
    drawnow;

    % Define input/output files
    % original wav data
    in_file = ['wavs\' file '.wav'];

    %convert the file to replace all spaces with _
    spaces = find(file == ' ');
    file(spaces) = '_';
```

```
% original wav chunk output
out_wav_orig = ['output_wavs\' file '_orig.wav'];
% ascii file of wav data
out_ascii_file = ['asciis\' file '.txt'];
% yipeng's algorithm outputs
out_lag_file1 = ['lags\' file];
out_lag_file2 = ['lags\' file '_unused'];
% wav output with sinusoidal pitches
out_wav_freqs = ['output_wavs\' file '.wav'];
% combined output
out_wav_comb = ['output_wavs\' file '_comb.wav'];

% First find the entire song parameters
[y, f] = wavread(in_file);
max_y = max(y);
length_y = max(size(y));
length_y = floor(length_y);
wavwrite(y(1:length_y), f, out_wav_orig);

chunk_size = 60000;
last_chunk = 0;

index = 0;
approx_chunk_time = 0;

while (start == 1 && last_chunk < length_y)
    tic;
    set(chunk_start_time, 'String', datestr(now));

    if (approx_chunk_time == 0)
        set(chunk_done_time, 'String', 'Calculating');
        set(song_done_time, 'String', 'Calculating');
    else
        set(chunk_done_time, 'String', datestr(now + ...
            approx_chunk_time/86400));
        set(song_done_time, 'String', datestr(now + ...
            (length_y-last_chunk)/chunk_size*approx_chunk_time/86400));
    end
    drawnow;

    % Define the chunk to read in
    offset = last_chunk;
    wav_chunk = min(chunk_size, length_y-last_chunk);
    last_chunk = last_chunk + wav_chunk;

    % Read in the *.wav into an ascii file
    [y,f] = wav2ascii(in_file,out_ascii_file,wav_chunk,offset,max_y);

    % Run Yipeng's algorithm
    cd Predominant_Pitch_Detection;
    dos_cmd = ['Predominant_Pitch_Detection.exe', ...
            ' ../', out_ascii_file, ...
            ' ../', out_lag_file1, num2str(index), ...
            ' ../', out_lag_file2];
    dos(dos_cmd);
    cd ..;

    index = index + 1;

    hold off;
    do_output(out_lag_file1, index, out_wav_freqs, y, 11);
```

```
        approx_chunk_time = toc;
        set(percent_done, 'String', num2str(100*last_chunk/length_y));
        drawnow;
    end

    % Output the frequency data to the pitch .wav
    freqs = do_output(out_lag_file1, index, out_wav_freqs, y);

    % Output the combined .wav (left is original, right is approximated)
    y_orig = wavread(out_wav_orig);
    y_aprx = wavread(out_wav_freqs);
    comb_len = max(max(size(y_orig), size(y_aprx)));
    % zero-pad to new length
    y_orig(length(y_orig) : comb_len) = 0;
    y_aprx(length(y_orig) : comb_len) = 0;
    y_comb = [y_orig, y_aprx];
    wavwrite(y_comb, f, out_wav_comb);
end
```

APPENDIX C.1.2 – wav2ascii.m

```
function [y,f] = wav2ascii(input_file, output_file, num_samples, offset, max_y)
% ========================================================================
% | wav2ascii.m by Jason Chang 3/26/2007
% |-----------------------------------------------------------------------
% | converts a wav file to ascii.
% |-----------------------------------------------------------------------
% | input_file - specifies location of the input file (*.wav)
% | output_file - specifies location of the output file (*.txt)
% | num_samples - the number of samples to take from the wave
% | offset - the offset to start from taking samples
% | max_y - the maximum value of the entire song
% ========================================================================

% Read data
y(1:num_samples, 1:2) = 0;
[y, f] = wavread([input_file], [1 + offset, num_samples + offset]);

% scale data so that maximum of the song is around 500
y_new = y*500/max_y;

% Output formatted data
fid = fopen([output_file], 'w');
fprintf(fid, ' % 1.7e\r\n', y_new);
fclose(fid);
```

APPENDIX C.1.2 – do_output.m

```
function [freqs] = do_output(input_name, num_inputs, output_name, ...
                             original_data, plotit)
% ========================================================================
% | do_output.m by Jason Chang 3/28/2007
% |-----------------------------------------------------------------------
% | Reads in the output from Yipeng's algorithm and converts the values to
% | frequencies.  Then outputs the specgram of the original data compared
% | to the estimated frequencies.
% |-----------------------------------------------------------------------
% | input_name - the name of the input file containing the data from
% |   Yipeng's algorithm
% | output_name - the name of the output wav that we want to write to
% | original_data - the original sound clip
% | plotit (optional) - plots if 1.  default is 0
% ========================================================================

Fs = 16000;

d = [];
if (nargin == 5 && plotit == 11)
    d = dlmread([input_name, num2str(num_inputs-1)]);
else
    for i=0:num_inputs-1
        d = [d, dlmread([input_name, num2str(i)])];
    end
end
voice_indices = find(d);
d(voice_indices) = Fs ./ d(voice_indices);

d_analog = discretize_freqs(d);
%freqs = d_analog;
freqs = medfilt1(d_analog, 15);

output_wav(freqs, output_name, 0.0403/4);

s = specgram(original_data(:,1), 1024, Fs);
s = abs(s);

if (nargin == 5 && (plotit ~= 0))
    num_x_vals = size(s,2);
    max_x = size(original_data,1)/Fs;
    x_vals(1:num_x_vals) = 0 : max_x/num_x_vals : max_x - max_x/num_x_vals;

    num_y_vals = size(s,1);
    max_y = Fs/2;
    y_vals(1:num_y_vals) = 0 : max_y/num_y_vals : max_y - max_y/num_y_vals;
    pcolor(x_vals, y_vals, s);

    V = axis;
    V(4) = 2000;
    axis(V);

    hold on;
    freqs_x_values = 0:0.01:length(freqs)/100-0.01;
    plot(freqs_x_values, freqs, 'w', 'LineWidth', 2);
end
```

APPENDIX C.1.3 – discretize_freqs.m

```
function [ freqs_out ] = discretize_freqs( freqs )
% ========================================================================
% | discretize_freqs.m by Jason Chang 3/28/2007
% |-----------------------------------------------------------------------
% | Converts the discretized frequencies (freqs) from a DFT to actual
% | frequency in Hz.  We will draw on the frequencies of piano notes and
% | round to the neared piano note.
% |-----------------------------------------------------------------------
% | freqs - an array of digital frequencies that are not perfect
% | freqs_out - the discretized frequencies
% ========================================================================
% Converts the discretized frequencies (freqs) from a DFT to actual
% frequency in Hz.  We will draw on the frequencies of piano notes and
% round to the neared piano note.

freqs_out = freqs;

codebook = ...
[
0,      27.5,   29.135, 30.868, 32.703, 34.648, 36.708, 38.891, 41.203, ...
43.654, 46.249, 48.99,  51.913, 55,     58.27,  61.735, 65.406, 69.296, ...
73.416, 77.782, 82.407, 87.307, 92.499, 97.999, 103.83, 110,    116.54, ...
123.47, 130.81, 138.59, 146.83, 155.56, 164.81, 174.61, 185,    196,    ...
207.65, 220,    233.08, 246.94, 261.63, 277.18, 293.66, 311.13, 329.63, ...
349.23, 369.99, 392,    415.3,  444,    466.16, 493.88, 523.25, 554.37, ...
587.33, 622.25, 659.25, 698.46, 739.99, 783.99, 830.61, 880,    932.33, ...
987.77, 1046.5, 1108.7, 1174.7, 1244.5, 1318.5, 1396.9, 1480,   1568,   ...
1661.2, 1760,   1864.7, 1979.5, 2093,   2217.5, 2349.3, 2489,    2637,  ...
2793,   2960,   3136,   3322.4, 3520,   3729.3, 3951.1, 4186 ...
];

partition = ...
[
13.75,   28.3175, 30.0015, 31.7855, 33.6755, 35.678,  37.7995, 40.047,  ...
42.4285, 44.9515, 47.6195, 50.4515, 53.4565, 56.635,  60.0025, 63.5705, ...
67.351,  71.356,  75.599,  80.0945, 84.857,  89.903,  95.249,  100.915, ...
106.915, 113.27,  120.005, 127.14,  134.7,   142.71,  151.195, 160.185, ...
169.71,  179.805, 190.5,   201.825, 213.825, 226.54,  240.01,  254.285, ...
269.405, 285.42,  302.395, 320.38,  339.43,  359.61,  380.995, 403.65,  ...
429.65,  455.08,  480.02,  508.565, 538.81,  570.85,  604.79,  640.75,  ...
678.855, 719.225, 761.99,  807.3,   855.305, 906.165, 960.05,  1017.14, ...
1077.6,  1141.7,  1209.6,  1281.5,  1357.7,  1438.45, 1524,    1614.6,  ...
1710.6,  1812.35, 1922.1,  2036.25, 2155.25, 2283.4,  2419.15, 2563,    ...
2715,    2876.5,  3048,    3229.2,  3421.2,  3624.65, 3840.2,  4068.55  ...
];

[index, freqs_out] = quantiz(freqs_out, partition, codebook);
```

APPENDIX C.1.4 – output_wav.m

```
function [ freqs_cc ] = output_wav( freqs, output_name, duration )
% ========================================================================
% | output_wav.m by Jason Chang 3/28/2007
% |-----------------------------------------------------------------------
% | Takes in an array of frequencies and writes those frequencies to a
% | sound (*.wav) file.  The sampling frequency used for the output will be
% | 16kHz.
% |-----------------------------------------------------------------------
% | freqs - the array of analog frequencies
% | output_name - the name of the output wav that we want to write to
% | duration - the length of each frequency in seconds
% | freqs_cc - the coded frequencies described in chain_code_freqs()
% ========================================================================

freqs_cc = chain_code_freqs(freqs);

for index=1:size(freqs_cc, 2)
    sf = 16000;                                 % sample frequency (Hz)
    n = sf * duration * freqs_cc(2, index);     % number of samples
    s = (1:n) / sf;                             % sound data preparation
    s = sin(2 .* pi .* freqs_cc(1,index) .* s);   % sinusoidal modulation

    if (index == 1)
        stotal = s;
    else
        stotal = [stotal, s];
    end
end
stotal = 0.99/max(stotal)*stotal;

wavwrite(stotal, 16000, output_name);
```

APPENDIX C.1.5 – chain_code_freqs.m

```
function [ output_freqs ] = chain_code_freqs( input_freqs )
% ============================================================================
% | chain_code_freqs.m by Jason Chang 3/28/2007
% |--------------------------------------------------------------------------
% | Codes the frequencies in a particular fashion.  Illustrate by example:
% |  input_freqs = [f1 f1 f1 f2 f2 f3 f1 f3 f3];
% |  output_freqs => [frequency, occurences, offset]' =
% |    [f1         f2         f3          f1          f3         ]
% |    [3          2          1           1           2          ]
% |    [0         (f2-f1)%13  (f3-f2)%13  (f3-f1)%13  (f3-f1)%13 ]
% |--------------------------------------------------------------------------
% | input_freqs - the array of analog frequencies to code
% | output_freqs - the coded array of frequencies
% ============================================================================

freq_index = 1;
current_freq = input_freqs(1);

output_freqs(1, 1) = input_freqs(1);
output_freqs(2, 1) = 1;
output_freqs(3, 1) = 0;

for index=2:length(input_freqs)
    next_freq = input_freqs(index);

    if (next_freq == current_freq)
        output_freqs(2, freq_index) = output_freqs(2, freq_index) + 1;
    else
        freq_index = freq_index + 1;
        output_freqs(1, freq_index) = next_freq;
        output_freqs(2, freq_index) = 1;
        output_freqs(3, freq_index) = mod(next_freq - current_freq, 13);
        current_freq = next_freq;
    end
```

APPENDIX C.1.6 – postscriptNew.m

```
% ========================================================================
% | chain_code_freqs.m by Jason Chang 5/01/2007
% |-----------------------------------------------------------------------
% | The script to run after running main.m on all the song files.  This
% | will combine all the lag files into one and output the correct info
% | to the database files
% ========================================================================
folder = 'db2\Simulation Set 1\';
db_folder = 'db2\filtered\';
Fs = 16000;

all_files = dir([ folder '*_unused']);

for i=1:length(all_files)
    % Get the name of the file
    file = all_files(i).name;
    disp(['Starting ' num2str(i) ': ' file]);

    if (exist([db_folder file]) == 0)
        % Drop the .wav extension
        file = file(1 : length(file)-7);

        all_lags = dir([ folder file '*']);
        % make sure that everything is alphabetized correctly (leading 0)
        for j=1:length(all_lags)
            lag = all_lags(j).name;
            if (length(file)+1 == length(lag))
                lag_new = [lag(1:length(lag)-1) '0' lag(length(lag))];
                movefile([folder lag], [folder lag_new]);
                lag = lag_new;
            end
        end


        d = [];
        all_lags = dir([ folder file '*']);
        for j=1:length(all_lags)
            if ( length(all_lags(j).name) ~= length([file '_combined']) && ...
                    length(all_lags(j).name) ~= length([file '_unused']) )
                lag = all_lags(j).name;
                d = [d, dlmread([folder lag])];
            end
        end
        dlmwrite([folder file '_combined'], d, ' ');

        voice_indices = find(d);
        d(voice_indices) = Fs ./ d(voice_indices);
        d2 = medfilt1(d, 15);
        d3 = medfilt1(d2, 15);

        indices = discretize_freqs_index2(d3);
        indices_cc = chain_code_freqs2(indices);

        fid = fopen([db_folder file], 'w');
        fprintf(fid, '%1d\r', size(indices_cc,2));
        fprintf(fid, '%1d\r', indices_cc(:));
        fclose(fid);
    end
end
```

APPENDIX C.2 – Real-time Pitch Extraction

APPENDIX C.2.1 – musicsearch.h

```
#define N 4096       /* Number of FFT points */
#define logN 10
```

APPENDIX C.2.2 – musicsearch.c

```
#include "core.h"
#include "window.h"
#include "musicsearch.h" /* Number of C FFT points defined here */

/* function defined in lab4fft.c */
void fft(void);

/* FFT data buffers */
int real[N]; /* Real part of data      */
int imag[N]; /* Imaginary part of data */

/* Our input/output buffers */
int inputs[N];
int outputs[N];

volatile int input_full = 0;/* volatile means interrupt changes it */
int count = 0;

/* Standard frequency bounds for all 88 keys of the piano */

int codebook[] = {13.75, 28.3175, 30.0015, 31.7855, 33.6755, 35.678, 37.7995,
40.047, 42.4285, 44.9515, 47.6195, 50.4515,
    53.4565, 56.635, 60.0025, 63.5705, 67.351, 71.356, 75.599, 80.0945, 84.857,
89.903, 95.249, 100.9145,
    106.915, 113.27, 120.005, 127.14, 134.7, 142.71, 151.195, 160.185, 169.71,
179.805, 190.5, 201.825,
    213.825, 226.54, 240.01, 254.285, 269.405, 285.42, 302.395, 320.38, 339.43,
359.61, 380.995, 403.65,
    429.65, 455.08, 480.02, 508.565, 538.81, 570.85, 604.79, 640.75, 678.855,
719.225, 761.99, 807.3,
    855.305, 906.165, 960.05, 1017.135, 1077.6, 1141.7, 1209.6, 1281.5, 1357.7,
1438.45, 1524, 1614.6,
    1710.6, 1812.35, 1922.1, 2036.25, 2155.25, 2283.4, 2419.15, 2563, 2715, 2876.5,
3048, 3229.2,
                  3421.2, 3624.65, 3840.2, 4068.55};

/* Translates frequency to note number or name */

char                                    notebook[]                                  =
"0102030405060708091011121314151617181920212223242526272829303132333435363738394041
4243444546474849505152535455565758596061626364656667686970717273747576778798081828
38485868788";

int notes = 89;     /* length of the codebook array */

/* Flag to indicate when to send serial data */
int sendOK = 0;

interrupt void irq(void)
{
  int *Xmitptr,*Rcvptr;              /* pointers to Xmit & Rcv Bufs   */
  int i,j;

  static int in_irq = 0;                 /* Flag to prevent reentrance */

  /* Make sure we're not in the interrupt (should never happen) */
  if( in_irq )
    return;
```

```
  /* Mark we're processing, and enable interrupts */
  in_irq = 1;
  enable_irq();

  /* The following waitaudio call is guaranteed not to
     actually wait; it will simply return the pointers. */
  WaitAudio(&Rcvptr,&Xmitptr);

  /* input_full should never be true... */
  if( !input_full )
  {
    for (i=0; i<BlockLen; i++)
    {
      /* Save input, and echo to channel 1 *
       * Echo only for debug purposes      */
                                  //inputs[count] = Xmitptr[6*i] = Rcvptr[4*i];
       inputs[count] = Rcvptr[4*i];

      /* Send FFT output to channel 2 *
       * Used only for debug purposes */
                                    //Xmitptr[6*i+1] = outputs[count];

                                                                  count++;
    }
  }

  /* Have we collected enough data yet? */
  if( count >= N )
    input_full = 1;

  /* We're not in the interrupt anymore... */
  disable_irq();
  in_irq = 0;
}


main()
{
  int i, max, freq, max_index, max_note, k;
  int new_max;
  int j = 0;
  int l = 5;
  int threshold = 10;
  int t = 0.01;
  int D = 8;
  int history[5];
  int last_note = 0;

  for (i=0; i<l; i++) history[i] = 0;
  /* Initialize IRQ stuff */
  count = 0;
  input_full = 0;
  SetAudioInterrupt(irq);       /* Set up interrupts */

  while (1)
  {
    while( !input_full );           /* Wait for a data buffer to collect */

    /* From here until we clear input_full can only take *
     * BlockLen sample times, so don't do too much here. */

    /* First, transfer inputs and outputs    *
```

```
     *  and decimate by factor D             */
                    for (i=0, j; i<N; i += D, j++)
                                        {
                                                        real[j] = inputs[i];
                                                                imag[j] = 0;
                                        }

    /* Zero-pad the rest of the FFT buffer   */
                             for (j; j<N; j++)
                                        {
                                                                real[j] = 0;
                                                                imag[j] = 0;
                                        }

    /* Done with that... ready for new data collection */
    count = 0;      /* Need to reset the count                */
    input_full = 0; /* Mark we're ready to collect more data  */

    /*********************************************************/
    /* Now that we've gotten the data moved, we can do the    */
    /* more lengthy processing.                               */

    /* Multiply the input signal by the Hamming window.       */
                           //for (i=0; i<N; i++)
                                    //real[i] = _smpy(real[i], window[i]);

    /* Bit-reverse and compute FFT in C                       */
    fft();

    /* Now, take absolute value squared of FFT                */
    max = 0;
                            for (i=0; i<N; i++)
                                        {
                                        outputs[i]  =  25*_sadd(_smpy(real[i],  real[i]),
_smpy(imag[i], imag[i]));

                                        /* Determine the position/absolute frequency of
the peak of the spectrum */
                                              if (max < outputs[i] && i<N/2)
                                                                        {
                                                                               max  =
outputs[i];

                                 max_index = i;
                                                                        }
                                        }

    /* Check if the max is a harmonic of some      *
     * fundamental frequency. Divide by an integer *
     * and check around there for a peak above the *
     * noise threshold 'threshold'.                */
                                new_max = 0;
                           for (i=-4; i<=4; i++)
                                     if (outputs[max_index/4+i] > max*t)
                                                                        {

                     new_max = outputs[max_index/4+i];

                          max_index = max_index/4;
                                                                        }
    new_max = 0;
                            for (i=-4; i<=4; i++)
```

```
                                    if (outputs[max_index/3+i] > max*t)
                                                                    {

                    new_max = outputs[max_index/3+i];

                        max_index = max_index/3;
                                                                    }
                             new_max = 0;
                          for (i=-4; i<=4; i++)
                                    if (outputs[max_index/2+i] > max*t)
                                                                    {

                    new_max = outputs[max_index/2+i];

                        max_index = max_index/2;
                                                                    }

    /* Translate FFT index into absolute frequency */
                      freq = max_index * 44100/N/D;

    /* Update history of past frequencies */
                         for (i=0; i<l-1; i++)
                                              history[i] = history[i+1];
                          history[l-1] = freq;

    /* Find the median of the past five greatest frequency   *
     *  values. Worst case n^2 where n is length of history. *
     *  Since history is small, negligible performance hit.  */

                           for (i=0; i<l; i++)
                                    {
                                                          k = 0;
                                                 for (j=0; j<l; j++)
                                                                    {
                                                                        if
(history[i] > history[j])

                                     k++;
                                                                    }
                                                       if (k = l/2)
                                                                    {
                                                                        freq
= history[i];

                           //history[l-1] = freq;
                                                                    }
                                    }

  /* Then translate that frequency to its corresponding key number */
                          for (i=0; i<notes; i++)
                                       if (codebook[i] - freq > 0) break;
                               max_note = --i;

    /* Check for input from computer. Needs to be initialized *
     *  before DSP board begins outputting. This can be done  *
     *  anywhere.                                             */
                             freq = SerialRX();
                               if (freq > 0)
                                                    sendOK = freq;

                     /* Apply some simple conditions
```

```
                                    //Check if the note heard is same as previous.
We don't want to duplicate a held note  *
                                    //Check if the note is within what is commonly
accepted as humanly possible            *
                                    //                                          i.e.
approximately 1 octave below to 2 octaves above middle C                    *
                                    //Check if the power spectrum of the signal is
above some noise floor                  *
                                    //Finally, check if we are ready to receive data
*/
    if (max_note >=25 && max_note <=63 && max > threshold && sendOK == 49)
                                     {
                                      if ((max_note < last_note+12) || last_note ==
0)
                                                                              {

                      SerialTX(notebook[max_note*2]);

                     SerialTX(notebook[max_note*2+1]);

                             //SerialTX(';');

                           last_note = max_note;
                                                                             }
                                     }
                          else if (sendOK == 49)
                                     {
                                                          SerialTX('0');
                                                          SerialTX('0');
                                     }

    /* Reset send condition */
                               sendOK = 48;

    /* Last, set the DC coefficient to -1 for a trigger pulse */
                  /* Used only for debugging purposes */
                          //outputs[0] = -32768;

    /* done, wait for next time around!                      */

  }
}
```

APPENDIX C.2.3 – lab4fft.c

```
/*****************************************************************/
/* lab4fft.c                                                     */
/* Douglas L. Jones                                              */
/* University of Illinois at Urbana-Champaign                    */
/* January 19, 1992                                              */
/* Changed for use w/ short integers and lookup table for ECE420 */
/* Matt Kleffner                                                 */
/* February 10, 2004                                             */
/*                                                               */
/*   fft: in-place radix-2 DIT DFT of a complex input            */
/*                                                               */
/*   Permission to copy and use this program is granted          */
/*   as long as this header is included.                         */
/*                                                               */
/* WARNING:                                                      */
/*   This file is intended for educational use only, since most  */
/*   manufacturers provide hand-tuned libraries which typically  */
/*   include the fastest fft routine for their DSP/processor     */
/*   architectures. High-quality, open-source fft routines       */
/*   written in C (and included in MATLAB) can be found at       */
/*   http://www.fftw.org                                         */
/*                                                               */
/*   #defines expected in lab4.h                                 */
/*         N:   length of FFT: must be a power of two            */
/*      logN:   N = 2**logN                                      */
/*                                                               */
/*   16-bit-limited input/output (must be defined elsewhere)     */
/*   real:   integer array of length N with real part of data    */
/*   imag:   integer array of length N with imag part of data    */
/*                                                               */
/*   sinetables.h must                                           */
/*   1) #define Nt to an equal or greater power of two than N    */
/*   2) contain the following integer arrays with                */
/*      element magnitudes bounded by M = 2**15-1:               */
/*         costable:   M*cos(-2*pi*n/Nt), n=0,1,...,Nt/2-1       */
/*         sintable:   M*sin(-2*pi*n/Nt), n=0,1,...,Nt/2-1       */
/*                                                               */
/*****************************************************************/

#include "musicsearch.h"
#include "sinetables.h"

extern int real[N];
extern int imag[N];

void fft(void)
{
   int   i,j,k,n1,n2,n3;
   int   c,s,a,t,Wr,Wi;

   j = 0;            /* bit-reverse */
   n2 = N >> 1;
   for (i=1; i < N - 1; i++)
   {
      n1 = n2;
      while ( j >= n1 )
      {
         j = j - n1;
         n1 = n1 >> 1;
```

```
   }
   j = j + n1;

   if (i < j)
   {
      t = real[i];
      real[i] = real[j];
      real[j] = t;
      t = imag[i];
      imag[i] = imag[j];
      imag[j] = t;
   }
}

/* FFT */
n2 = 1; n3 = Nt;

for (i=0; i < logN; i++)
{
   n1 = n2;      /* n1 = 2**i     */
   n2 = n2 + n2; /* n2 = 2**(i+1) */
   n3 = n3 >> 1; /* cos/sin arg of -6.283185307179586/n2 */
   a = 0;

   for (j=0; j < n1; j++)
   {
      c = costable[a];
      s = sintable[a];
      a = a + n3;

      for (k=j; k < N; k=k+n2)
      {
         /* Code for standard 32-bit hardware, */
         /* with real,imag limited to 16 bits  */
         /*
         Wr = (c*real[k+n1] - s*imag[k+n1]) >> 15;
         Wi = (s*real[k+n1] + c*imag[k+n1]) >> 15;
         real[k+n1] = (real[k] - Wr) >> 1;
         imag[k+n1] = (imag[k] - Wi) >> 1;
         real[k] = (real[k] + Wr) >> 1;
         imag[k] = (imag[k] + Wi) >> 1;
         */
         /* End standard 32-bit code */

         /* Code for TI TMS320C54X series */

         Wr = ((long int)(c*real[k+n1]) - (long int)(s*imag[k+n1])) >> 15;
         Wi = ((long int)(s*real[k+n1]) + (long int)(c*imag[k+n1])) >> 15;
         real[k+n1] = ((long int)real[k] - (long int)Wr) >> 1;
         imag[k+n1] = ((long int)imag[k] - (long int)Wi) >> 1;
         real[k] = ((long int)real[k] + (long int)Wr) >> 1;
         imag[k] = ((long int)imag[k] + (long int)Wi) >> 1;

         /* End code for TMS320C54X series */

         /* Intrinsic code for TMS320C54X series */
         /*
         Wr = _ssub(_smpy(c, real[k+n1]), _smpy(s, imag[k+n1]));
         Wi = _sadd(_smpy(s, real[k+n1]), _smpy(c, imag[k+n1]));
         real[k+n1] = _sshl(_ssub(real[k], Wr),-1);
         imag[k+n1] = _sshl(_ssub(imag[k], Wi),-1);
         real[k] = _sshl(_sadd(real[k], Wr),-1);
```

```
            imag[k] = _sshl(_sadd(imag[k], Wi),-1);
            */
            /* End intrinsic code for TMS320C54X series */
         }
      }
   }
   return;
}
```

APPENDIX C.2.4 – core.h

```
#ifndef __CORE_H
#define __CORE_H

extern int * _K_FRAME_SIZE;

/* Block size #defines - change if necessary */
#define K_FRAME_SIZE  ((int)(&_K_FRAME_SIZE))
#define BlockLen      (K_FRAME_SIZE/2)

/* CPU control #defines */
#define disable_irq()      asm(" ssbx intm");   /* disable interrupts */
#define enable_irq()       asm(" rsbx intm");    /* enable interrupts */


/* Audio functions */
void WaitAudio(int **Rcv, int **Xmit);
void SetInterrupt(void (*intr)(void));

/* Serial functions */
int  SerialRXBufCheck(void);                                          /*
Returns number of chars in RX buffer */
int  SerialTXBufCheck(void);                                          /*
Returns number of chars in TX buffer */

int  SerialRX(void);                          /* Returns incoming character or -1
*/
void SerialTX(int character);                 /* Sends a character */

int  SerialRXm(int count, int *buffer);                               /*
Read up to count chars into buffer */
int  SerialTXm(int count, int *buffer);                               /*
Write count chars from buffer */

/* Extended Program RAM functions */
void ExtRead(long source, int *dest, int count);
void ExtWrite(long dest, int *source, int count);

#endif
```

APPENDIX C.2.5 – sinetables.h

```
#define Nt 1024

int costable[]={ \
 32767,  32767,  32766,  32762,  32758,  32753,  32746,  32738, \
 32729,  32718,  32706,  32693,  32679,  32664,  32647,  32629, \
 32610,  32590,  32568,  32546,  32522,  32496,  32470,  32442, \
 32413,  32383,  32352,  32319,  32286,  32251,  32214,  32177, \
 32138,  32099,  32058,  32015,  31972,  31927,  31881,  31834, \
 31786,  31737,  31686,  31634,  31581,  31527,  31471,  31415, \
 31357,  31298,  31238,  31177,  31114,  31050,  30986,  30920, \
 30853,  30784,  30715,  30644,  30572,  30499,  30425,  30350, \
 30274,  30196,  30118,  30038,  29957,  29875,  29792,  29707, \
 29622,  29535,  29448,  29359,  29269,  29178,  29086,  28993, \
 28899,  28803,  28707,  28610,  28511,  28411,  28311,  28209, \
 28106,  28002,  27897,  27791,  27684,  27576,  27467,  27357, \
 27246,  27133,  27020,  26906,  26791,  26674,  26557,  26439, \
 26320,  26199,  26078,  25956,  25833,  25708,  25583,  25457, \
 25330,  25202,  25073,  24943,  24812,  24680,  24548,  24414, \
 24279,  24144,  24008,  23870,  23732,  23593,  23453,  23312, \
 23170,  23028,  22884,  22740,  22595,  22449,  22302,  22154, \
 22006,  21856,  21706,  21555,  21403,  21251,  21097,  20943, \
 20788,  20632,  20475,  20318,  20160,  20001,  19841,  19681, \
 19520,  19358,  19195,  19032,  18868,  18703,  18538,  18372, \
 18205,  18037,  17869,  17700,  17531,  17361,  17190,  17018, \
 16846,  16673,  16500,  16326,  16151,  15976,  15800,  15624, \
 15447,  15269,  15091,  14912,  14733,  14553,  14373,  14192, \
 14010,  13828,  13646,  13463,  13279,  13095,  12910,  12725, \
 12540,  12354,  12167,  11980,  11793,  11605,  11417,  11228, \
 11039,  10850,  10660,  10469,  10279,  10088,   9896,   9704, \
  9512,   9319,   9127,   8933,   8740,   8546,   8351,   8157, \
  7962,   7767,   7571,   7376,   7180,   6983,   6787,   6590, \
  6393,   6195,   5998,   5800,   5602,   5404,   5205,   5007, \
  4808,   4609,   4410,   4211,   4011,   3812,   3612,   3412, \
  3212,   3012,   2811,   2611,   2411,   2210,   2009,   1809, \
  1608,   1407,   1206,   1005,    804,    603,    402,    201, \
     0,   -201,   -402,   -603,   -804,  -1005,  -1206,  -1407, \
 -1608,  -1809,  -2009,  -2210,  -2411,  -2611,  -2811,  -3012, \
 -3212,  -3412,  -3612,  -3812,  -4011,  -4211,  -4410,  -4609, \
 -4808,  -5007,  -5205,  -5404,  -5602,  -5800,  -5998,  -6195, \
 -6393,  -6590,  -6787,  -6983,  -7180,  -7376,  -7571,  -7767, \
 -7962,  -8157,  -8351,  -8546,  -8740,  -8933,  -9127,  -9319, \
 -9512,  -9704,  -9896, -10088, -10279, -10469, -10660, -10850, \
-11039, -11228, -11417, -11605, -11793, -11980, -12167, -12354, \
-12540, -12725, -12910, -13095, -13279, -13463, -13646, -13828, \
-14010, -14192, -14373, -14553, -14733, -14912, -15091, -15269, \
-15447, -15624, -15800, -15976, -16151, -16326, -16500, -16673, \
-16846, -17018, -17190, -17361, -17531, -17700, -17869, -18037, \
-18205, -18372, -18538, -18703, -18868, -19032, -19195, -19358, \
-19520, -19681, -19841, -20001, -20160, -20318, -20475, -20632, \
-20788, -20943, -21097, -21251, -21403, -21555, -21706, -21856, \
-22006, -22154, -22302, -22449, -22595, -22740, -22884, -23028, \
-23170, -23312, -23453, -23593, -23732, -23870, -24008, -24144, \
-24279, -24414, -24548, -24680, -24812, -24943, -25073, -25202, \
-25330, -25457, -25583, -25708, -25833, -25956, -26078, -26199, \
-26320, -26439, -26557, -26674, -26791, -26906, -27020, -27133, \
-27246, -27357, -27467, -27576, -27684, -27791, -27897, -28002, \
-28106, -28209, -28311, -28411, -28511, -28610, -28707, -28803, \
-28899, -28993, -29086, -29178, -29269, -29359, -29448, -29535, \
-29622, -29707, -29792, -29875, -29957, -30038, -30118, -30196, \
```

```
-30274, -30350, -30425, -30499, -30572, -30644, -30715, -30784, \
-30853, -30920, -30986, -31050, -31114, -31177, -31238, -31298, \
-31357, -31415, -31471, -31527, -31581, -31634, -31686, -31737, \
-31786, -31834, -31881, -31927, -31972, -32015, -32058, -32099, \
-32138, -32177, -32214, -32251, -32286, -32319, -32352, -32383, \
-32413, -32442, -32470, -32496, -32522, -32546, -32568, -32590, \
-32610, -32629, -32647, -32664, -32679, -32693, -32706, -32718, \
-32729, -32738, -32746, -32753, -32758, -32762, -32766, -32767  \
};

int sintable[]={ \
     0,   -201,   -402,   -603,   -804,  -1005,  -1206,  -1407, \
 -1608,  -1809,  -2009,  -2210,  -2411,  -2611,  -2811,  -3012, \
 -3212,  -3412,  -3612,  -3812,  -4011,  -4211,  -4410,  -4609, \
 -4808,  -5007,  -5205,  -5404,  -5602,  -5800,  -5998,  -6195, \
 -6393,  -6590,  -6787,  -6983,  -7180,  -7376,  -7571,  -7767, \
 -7962,  -8157,  -8351,  -8546,  -8740,  -8933,  -9127,  -9319, \
 -9512,  -9704,  -9896, -10088, -10279, -10469, -10660, -10850, \
-11039, -11228, -11417, -11605, -11793, -11980, -12167, -12354, \
-12540, -12725, -12910, -13095, -13279, -13463, -13646, -13828, \
-14010, -14192, -14373, -14553, -14733, -14912, -15091, -15269, \
-15447, -15624, -15800, -15976, -16151, -16326, -16500, -16673, \
-16846, -17018, -17190, -17361, -17531, -17700, -17869, -18037, \
-18205, -18372, -18538, -18703, -18868, -19032, -19195, -19358, \
-19520, -19681, -19841, -20001, -20160, -20318, -20475, -20632, \
-20788, -20943, -21097, -21251, -21403, -21555, -21706, -21856, \
-22006, -22154, -22302, -22449, -22595, -22740, -22884, -23028, \
-23170, -23312, -23453, -23593, -23732, -23870, -24008, -24144, \
-24279, -24414, -24548, -24680, -24812, -24943, -25073, -25202, \
-25330, -25457, -25583, -25708, -25833, -25956, -26078, -26199, \
-26320, -26439, -26557, -26674, -26791, -26906, -27020, -27133, \
-27246, -27357, -27467, -27576, -27684, -27791, -27897, -28002, \
-28106, -28209, -28311, -28411, -28511, -28610, -28707, -28803, \
-28899, -28993, -29086, -29178, -29269, -29359, -29448, -29535, \
-29622, -29707, -29792, -29875, -29957, -30038, -30118, -30196, \
-30274, -30350, -30425, -30499, -30572, -30644, -30715, -30784, \
-30853, -30920, -30986, -31050, -31114, -31177, -31238, -31298, \
-31357, -31415, -31471, -31527, -31581, -31634, -31686, -31737, \
-31786, -31834, -31881, -31927, -31972, -32015, -32058, -32099, \
-32138, -32177, -32214, -32251, -32286, -32319, -32352, -32383, \
-32413, -32442, -32470, -32496, -32522, -32546, -32568, -32590, \
-32610, -32629, -32647, -32664, -32679, -32693, -32706, -32718, \
-32729, -32738, -32746, -32753, -32758, -32762, -32766, -32767, \
-32767, -32767, -32766, -32762, -32758, -32753, -32746, -32738, \
-32729, -32718, -32706, -32693, -32679, -32664, -32647, -32629, \
-32610, -32590, -32568, -32546, -32522, -32496, -32470, -32442, \
-32413, -32383, -32352, -32319, -32286, -32251, -32214, -32177, \
-32138, -32099, -32058, -32015, -31972, -31927, -31881, -31834, \
-31786, -31737, -31686, -31634, -31581, -31527, -31471, -31415, \
-31357, -31298, -31238, -31177, -31114, -31050, -30986, -30920, \
-30853, -30784, -30715, -30644, -30572, -30499, -30425, -30350, \
-30274, -30196, -30118, -30038, -29957, -29875, -29792, -29707, \
-29622, -29535, -29448, -29359, -29269, -29178, -29086, -28993, \
-28899, -28803, -28707, -28610, -28511, -28411, -28311, -28209, \
-28106, -28002, -27897, -27791, -27684, -27576, -27467, -27357, \
-27246, -27133, -27020, -26906, -26791, -26674, -26557, -26439, \
-26320, -26199, -26078, -25956, -25833, -25708, -25583, -25457, \
-25330, -25202, -25073, -24943, -24812, -24680, -24548, -24414, \
-24279, -24144, -24008, -23870, -23732, -23593, -23453, -23312, \
-23170, -23028, -22884, -22740, -22595, -22449, -22302, -22154, \
-22006, -21856, -21706, -21555, -21403, -21251, -21097, -20943, \
-20788, -20632, -20475, -20318, -20160, -20001, -19841, -19681, \
```

```
-19520, -19358, -19195, -19032, -18868, -18703, -18538, -18372, \
-18205, -18037, -17869, -17700, -17531, -17361, -17190, -17018, \
-16846, -16673, -16500, -16326, -16151, -15976, -15800, -15624, \
-15447, -15269, -15091, -14912, -14733, -14553, -14373, -14192, \
-14010, -13828, -13646, -13463, -13279, -13095, -12910, -12725, \
-12540, -12354, -12167, -11980, -11793, -11605, -11417, -11228, \
-11039, -10850, -10660, -10469, -10279, -10088,  -9896,  -9704, \
 -9512,  -9319,  -9127,  -8933,  -8740,  -8546,  -8351,  -8157, \
 -7962,  -7767,  -7571,  -7376,  -7180,  -6983,  -6787,  -6590, \
 -6393,  -6195,  -5998,  -5800,  -5602,  -5404,  -5205,  -5007, \
 -4808,  -4609,  -4410,  -4211,  -4011,  -3812,  -3612,  -3412, \
 -3212,  -3012,  -2811,  -2611,  -2411,  -2210,  -2009,  -1809, \
 -1608,  -1407,  -1206,  -1005,   -804,   -603,   -402,   -201  \
};
```

APPENDIX C.3 – Search

APPENDIX C.3.1 – main.def

```
LIBRARY main
DESCRIPTION     'A C++ dll that can be called from VB'

EXPORTS
  find_occurences_c_dll @1
```

APPENDIX C.3.1 – main.h

```
#include <iostream>
#include <windows.h>
#include <oleauto.h>
#include "math.h"

long __declspec (dllexport) __stdcall find_occurences_c_dll(
           long *input_string_arr, long input_string_arr_len,
    long *song_notes_arr, long *song_lengths_arr, long song_arr_len,
         long search_window, long allow_skip, long skip_weight);
```

APPENDIX C.3.1 – main.cpp

```
#include "main.h"

// ****************************************************************************
// **  find_occurences
// ****************************************************************************

long __declspec (dllexport) __stdcall find_occurences_c_dll(
  long *input_string_arr, long input_string_arr_len,
  long *song_notes_arr, long *song_lengths_arr, long song_arr_len,
  long search_window, long allow_skip, long skip_weight)
{
  // indices for the loops
  long start_i;
  long input_i;
  long search_i;
  long skip_i;


  // the total_time of all the notes within the current search window
  long total_time;
  long skip_total_time;
  // the time since you have found the last note
  long note_time;

  // the current_note and the note you are looking for
  long current_note;
  long looking_note;
  long next_looking_note;

  // the number of times the input ahs been found
  double num_found = 0;

  // the longest you can go before you give up on finding the next note (in10ms)
  long max_note_time = 300;

  // boolean values indicating search progress
  int note_found = 0;
  int input_found;

  // the number of notes skipped within the song
  int num_skips = 0;
  int cur_num_skips;

  // loop over the entire song
  for (start_i=0; start_i<song_arr_len; start_i++)
  {
    input_found = 0;
    search_i = start_i;
    total_time = song_lengths_arr[start_i];
    cur_num_skips = 0;

    // preserve total_time for skip purposes in future
    skip_total_time = total_time;

    // loop over all the notes in the input
    for (input_i=0; input_i<input_string_arr_len; input_i++)
    {
      // define the current note and note we are looking for
      current_note = song_notes_arr[search_i];
```

```
looking_note = current_note + input_string_arr[input_i];
if (input_i+1 < input_string_arr_len)
  next_looking_note = looking_note + input_string_arr[input_i+1];

note_found = 0;
note_time = 0;

// preserve the starting index in case skips need to be addressed
skip_i = search_i+1;

// begin the search starting at search_i
for (search_i=search_i+1; search_i < song_arr_len; search_i++)
{
  total_time += song_lengths_arr[search_i];

  // check to see if we have not found the note in the time frame
  if (note_time > max_note_time)
    // finished the note
    break;

  note_time += song_lengths_arr[search_i];

  // check to see if we found the note
  if (song_notes_arr[search_i] == looking_note)
  {
    // we found the note
    note_found = 1;

    // check to see if we found all of them
    if (input_i >= input_string_arr_len - 1)
      input_found = 1;

    // finished the note
    break;
  }
}

// finished one note see if we are done with starting at start_i
if (input_found)
  // found the string... break out
  break;
else if (!note_found)
{
  // didn't find a note within the time frame... see if user allows skips
  if (allow_skip == 1)
  {
    // check to see if we are just skipping the last note
    if (input_i >= input_string_arr_len-2)
    {
      // we are skipping the last note and we found the string!
      cur_num_skips++;
      input_found = 1;
      break;
    }

    note_time = 0;
    // not on second to last note...
    for (skip_i; skip_i < song_arr_len; skip_i++)
    {
      skip_total_time += song_lengths_arr[skip_i];

      // check to see if we have not found the note in the time frame
```

```
            if (note_time > max_note_time)
              // finished the note
              break;

            note_time += song_lengths_arr[skip_i];

            // check to see if we found the note
            if (song_notes_arr[skip_i] == next_looking_note)
            {
              // we found the note
              note_found = 1;
              cur_num_skips++;
              input_i++;

              // get the new index
              search_i = skip_i;

              // check to see if we found all of them
              if (input_i >= input_string_arr_len - 1)
                input_found = 1;

              // finished the note
              break;
            }
          }
        }

        if (!note_found || input_found)
          // did not find a note or found the string... break out
          break;
      }

      // if reached here, we found a note, and there are more notes left
    }

    // we finished searching for an entire input
    if (input_found && (total_time < (input_string_arr_len)*search_window))
    {
      num_skips += cur_num_skips;
      if (cur_num_skips > 0)
        num_found = num_found + 1/(cur_num_skips*skip_weight);
      else
        num_found++;
    }
  }

  return((long)num_found);
}
```

APPENDIX C.3.1 – search.frm

```
VERSION 5.00
Object = "{648A5603-2C6E-101B-82B6-000000000014}#1.1#0"; "MSCOMM32.OCX"
Begin VB.Form Form1
   BorderStyle     =   1  'Fixed Single
   Caption         =   "Music Search Engine"
   ClientHeight    =   9780
   ClientLeft      =   45
   ClientTop       =   435
   ClientWidth     =   10830
   LinkTopic       =   "Form1"
   MaxButton       =   0   'False
   MinButton       =   0   'False
   ScaleHeight     =   9780
   ScaleWidth      =   10830
   StartUpPosition =   2  'CenterScreen
   Begin VB.Timer Timer3
      Left            =   8760
      Top             =   9240
   End
   Begin VB.Timer Timer2
      Interval        =   100
      Left            =   9480
      Top             =   9240
   End
   Begin VB.Timer timer1
      Enabled         =   0   'False
      Interval        =   10
      Left            =   10200
      Top             =   9240
   End
   Begin VB.CommandButton sw_plus
      Caption         =   "+"
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   9.75
         Charset         =   0
         Weight          =   700
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   225
      Left            =   7920
      TabIndex        =   65
      Top             =   1920
      Width           =   255
   End
   Begin VB.CommandButton sw_minus
      Caption         =   "-"
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   9.75
         Charset         =   0
         Weight          =   700
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   225
```

```
   Left            =   7920
   TabIndex        =   64
   Top             =   2160
   Width           =   255
End
Begin VB.CommandButton wf_minus
   Caption         =   "-"
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   9.75
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   225
   Left            =   7920
   TabIndex        =   63
   Top             =   2760
   Width           =   255
End
Begin VB.CommandButton wf_plus
   Caption         =   "+"
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   9.75
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   225
   Left            =   7920
   TabIndex        =   62
   Top             =   2520
   Width           =   255
End
Begin VB.CheckBox allow_skip
   Caption         =   "Skip notes"
   Height          =   255
   Left            =   6600
   TabIndex        =   59
   Top             =   1680
   Width           =   1215
End
Begin VB.Frame searching
   BackColor       =   &H000000FF&
   BorderStyle     =   0  'None
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   18
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   2655
   Left            =   3000
   TabIndex        =   56
```

```
   Top             =   3600
   Visible         =   0   'False
   Width           =   5175
   Begin VB.Label Label2
      Alignment       =   2  'Center
      BackColor       =   &H000000FF&
      Caption         =   "Searching... Please Wait"
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   18
         Charset         =   0
         Weight          =   400
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      ForeColor       =   &H00FFFFFF&
      Height          =   495
      Left            =   0
      TabIndex        =   57
      Top             =   1080
      Width           =   5175
   End
End
Begin MSCommLib.MSComm MSComm1
   Left            =   1080
   Top             =   8400
   _ExtentX        =   1005
   _ExtentY        =   1005
   _Version        =   393216
   DTREnable       =   -1  'True
   BaudRate        =   38400
End
Begin VB.ListBox instructions
   Appearance      =   0  'Flat
   BackColor       =   &H80000004&
   ForeColor       =   &H00000000&
   Height          =   1200
   Left            =   3240
   TabIndex        =   55
   Top             =   8280
   Width           =   4335
End
Begin VB.CommandButton stop_btn
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   13.5
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Left            =   5040
   Picture         =   "search.frx":0000
   Style           =   1  'Graphical
   TabIndex        =   54
   Top             =   1800
   Width           =   1215
End
Begin VB.TextBox db_dir
```

```
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   9.75
      Charset         =   0
      Weight          =   400
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   360
   Left            =   2160
   TabIndex        =   49
   Top             =   120
   Width           =   8535
End
Begin VB.CommandButton clear_note
   Caption         =   "Clear Note"
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   9.75
      Charset         =   0
      Weight          =   400
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   495
   Left            =   2400
   TabIndex        =   47
   Top             =   1800
   Width           =   1335
End
Begin VB.FileListBox file_list
   Height          =   870
   Left            =   7800
   TabIndex        =   46
   Top             =   6720
   Visible         =   0   'False
   Width           =   2655
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00000000&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Index           =   10
   Left            =   2760
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   19
   Top             =   3180
   Width           =   375
End
Begin VB.CommandButton key
```

```
   Appearance      =   0  'Flat
   BackColor       =   &H00000000&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Index           =   8
   Left            =   2280
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   17
   Top             =   3180
   Width           =   375
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00000000&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Index           =   6
   Left            =   1800
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   15
   Top             =   3180
   Width           =   375
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00000000&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Index           =   3
   Left            =   840
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   12
   Top             =   3180
   Width           =   375
End
```

```
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00000000&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Index           =   1
   Left            =   360
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   10
   Top             =   3180
   Width           =   375
End
Begin VB.CommandButton start_stop
   BackColor       =   &H0000FF00&
   Caption         =   "Start Recording"
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   12
      Charset         =   0
      Weight          =   400
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Left            =   120
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   7
   Top             =   1800
   Width           =   2175
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00000000&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Index           =   34
   Left            =   9480
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   43
   Top             =   3180
   Width           =   375
End
```

```
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00000000&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Index           =   32
   Left            =   9000
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   41
   Top             =   3180
   Width           =   375
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00000000&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Index           =   30
   Left            =   8520
   MaskColor       =   &H00000000&
   Style           =   1  'Graphical
   TabIndex        =   39
   Top             =   3180
   Width           =   375
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00000000&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Index           =   27
   Left            =   7560
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   36
   Top             =   3180
   Width           =   375
```

```
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00000000&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Index           =   25
   Left            =   7080
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   34
   Top             =   3180
   Width           =   375
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00000000&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Index           =   22
   Left            =   6120
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   31
   Top             =   3180
   Width           =   375
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00000000&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Index           =   20
   Left            =   5640
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   29
   Top             =   3180
```

```
   Width           =   375
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00000000&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Index           =   18
   Left            =   5160
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   27
   Top             =   3180
   Width           =   375
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00000000&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Index           =   13
   Left            =   3720
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   22
   Top             =   3180
   Width           =   375
End
Begin VB.CommandButton search
   Caption         =   "SEARCH"
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   13.5
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Left            =   8280
   TabIndex        =   9
   Top             =   1800
   Width           =   2415
End
Begin VB.CommandButton clear
```

```
      Caption         =   "Clear All"
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   9.75
         Charset         =   0
         Weight          =   400
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   495
      Left            =   2400
      TabIndex        =   8
      Top             =   2520
      Width           =   1335
   End
   Begin VB.Frame Frame3
      Caption         =   "Output"
      Height          =   2895
      Left            =   120
      TabIndex        =   5
      Top             =   5280
      Width           =   10575
      Begin VB.ListBox output_name
         Height          =   2205
         Left            =   120
         TabIndex        =   51
         Top             =   240
         Width           =   10335
      End
      Begin VB.Label Label7
         Alignment       =   2  'Center
         Caption         =   "Click on a song to play it.  Click the Stop button to
stop playing it."
         BeginProperty Font
            Name            =   "MS Sans Serif"
            Size            =   9.75
            Charset         =   0
            Weight          =   400
            Underline       =   0   'False
            Italic          =   0   'False
            Strikethrough   =   0   'False
         EndProperty
         Height          =   255
         Left            =   120
         TabIndex        =   53
         Top             =   2520
         Width           =   10335
      End
   End
   Begin VB.Frame Frame2
      Caption         =   "Search Input"
      Height          =   1095
      Left            =   2040
      TabIndex        =   4
      Top             =   600
      Width           =   8655
      Begin VB.Label search_input
         Alignment       =   2  'Center
         BeginProperty Font
            Name            =   "MS Sans Serif"
            Size            =   9.75
```

```
         Charset         =   0
         Weight          =   400
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   495
      Left            =   120
      TabIndex        =   6
      Top             =   240
      Width           =   8415
   End
End
Begin VB.Frame input_frame
   Caption         =   "Input Method"
   Height          =   1095
   Left            =   120
   TabIndex        =   1
   Top             =   600
   Width           =   1815
   Begin VB.ComboBox cboComm
      Height          =   315
      Left            =   1200
      TabIndex        =   58
      Top             =   330
      Width           =   495
   End
   Begin VB.OptionButton input1
      Caption         =   "Virtual Keyboard"
      Height          =   255
      Index           =   1
      Left            =   120
      TabIndex        =   3
      Top             =   750
      Value           =   -1  'True
      Width           =   1455
   End
   Begin VB.OptionButton input1
      Caption         =   "USB Data"
      Height          =   255
      Index           =   0
      Left            =   120
      TabIndex        =   2
      Top             =   360
      Width           =   1455
   End
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00000000&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Index           =   15
   Left            =   4200
```

```
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   24
   Top             =   3180
   Width           =   375
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   0
   Left            =   120
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   0
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   2
   Left            =   600
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   11
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   4
```

```
   Left            =   1080
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   13
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   5
   Left            =   1560
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   14
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   7
   Left            =   2040
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   16
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
```

```
   Index           =   9
   Left            =   2520
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   18
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   11
   Left            =   3000
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   20
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00C0FFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   12
   Left            =   3480
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   21
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
```

```
   Height          =   1935
   Index           =   14
   Left            =   3960
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   23
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   16
   Left            =   4440
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   25
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   17
   Left            =   4920
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   26
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
```

```
   EndProperty
   Height          =   1935
   Index           =   19
   Left            =   5400
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   28
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   21
   Left            =   5880
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   30
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   23
   Left            =   6360
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   32
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
```

```
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   24
   Left            =   6840
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   33
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   26
   Left            =   7320
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   35
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   28
   Left            =   7800
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   37
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
```

```
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   29
   Left            =   8280
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   38
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   31
   Left            =   8760
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   40
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   33
   Left            =   9240
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   42
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
```

```
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   35
   Left            =   9720
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   44
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton key
   Appearance      =   0  'Flat
   BackColor       =   &H00FFFFFF&
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   8.25
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1935
   Index           =   36
   Left            =   10200
   MaskColor       =   &H00FFFFFF&
   Style           =   1  'Graphical
   TabIndex        =   45
   Top             =   3180
   Width           =   495
End
Begin VB.CommandButton play
   BeginProperty Font
      Name            =   "MS Sans Serif"
      Size            =   13.5
      Charset         =   0
      Weight          =   700
      Underline       =   0   'False
      Italic          =   0   'False
      Strikethrough   =   0   'False
   EndProperty
   Height          =   1215
   Left            =   3840
   Picture         =   "search.frx":006F
   Style           =   1  'Graphical
   TabIndex        =   52
   Top             =   1800
   Width           =   1215
End
Begin VB.Label Label4
   Alignment       =   1  'Right Justify
   Caption         =   "Skip Weighting:"
   Height          =   375
   Left            =   6360
   TabIndex        =   67
   Top             =   2040
   Width           =   1215
End
Begin VB.Label skip_weight
```

```
      Alignment       =   2  'Center
      Caption         =   "1"
      Height          =   375
      Left            =   7560
      TabIndex        =   66
      Top             =   2040
      Width           =   255
   End
   Begin VB.Label window_factor
      Alignment       =   2  'Center
      Caption         =   "5"
      Height          =   375
      Left            =   7560
      TabIndex        =   61
      Top             =   2640
      Width           =   255
   End
   Begin VB.Label Label1
      Alignment       =   1  'Right Justify
      Caption         =   "Window Factor:"
      Height          =   375
      Left            =   6360
      TabIndex        =   60
      Top             =   2640
      Width           =   1215
   End
   Begin VB.Label Label6
      Caption         =   "Label6"
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   8.25
         Charset         =   0
         Weight          =   700
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   495
      Left            =   4800
      TabIndex        =   50
      Top             =   3600
      Width           =   1215
   End
   Begin VB.Label Label5
      Alignment       =   1  'Right Justify
      Caption         =   "Database Directory:"
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   9.75
         Charset         =   0
         Weight          =   400
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   375
      Left            =   120
      TabIndex        =   48
      Top             =   180
      Width           =   1935
   End
End
```

```
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Private Declare Function sndPlaySound Lib "winmm.dll" Alias "sndPlaySoundA" _
    (ByVal lpszSoundName As String, ByVal uFlags As Long) As Long

'Private Declare Function find_occurences_c_dll _
'    Lib "W:\c_dll\c_dll.dll" _
'   (input_string_arr As Long, ByVal input_string_arr_len As Long, _
'   song_notes_arr As Long, song_lengths_arr As Long, ByVal song_arr_len As Long, _
'    ByVal search_window As Long, ByVal allow_skip As Long, ByVal skip_weight As
Long) As Long

Private Declare Function find_occurences_c_dll _
    Lib "C:\c_dll\c_dll.dll" _
   (input_string_arr As Long, ByVal input_string_arr_len As Long, _
   song_notes_arr As Long, song_lengths_arr As Long, ByVal song_arr_len As Long, _
   ByVal search_window As Long, ByVal allow_skip As Long, ByVal skip_weight As
Long) As Long

'Private Declare Function find_occurences_c_dll _
'                              Lib       "C:\Documents       and       Settings\Jason
Chang\Desktop\c_vb_search\c_dll\Debug\c_dll.dll" _
'   (input_string_arr As Long, ByVal input_string_arr_len As Long, _
'   song_notes_arr As Long, song_lengths_arr As Long, ByVal song_arr_len As Long, _
'    ByVal search_window As Long, ByVal allow_skip As Long, ByVal skip_weight As
Long) As Long

Const SND_ASYNC = &H1
Const SND_LOOP = &H8
Const SND_NODEFAULT = &H2
Const SND_SYNC = &H0
Const SND_NOSTOP = &H10
Const SND_MEMORY = &H4

Dim search_time As Long
Dim dead As Boolean
Dim input_string(0 To 49) As Integer
Dim input_string_str As String
Dim input_index As Integer
Dim last_note As Integer
Dim first_note As Integer
Dim start As Boolean
Dim db_folder As String

Dim song_names() As String
Dim song_occurences() As Double
Dim num_songs As Integer

Private Sub Form_Load()
    input_index = 0
    timer1.Enabled = False
    Dim i As Integer
    i = 0
    For i = 0 To 49
        input_string(i) = 100
    Next i
    start = True
```

```
    db_folder = CurDir & "\db2\"
    'db_folder = "C:\Documents and Settings\Jason Chang\Desktop\search2\db2\"
    db_dir.Text = db_folder

    Dim note_kbd() As Variant
    note_kbd = Array( _
        "Z", "S", "X", "D", "C", _
        "V", "G", "B", "H", "N", "J", "M", _
        ",", "l", ".", ";", "/", _
        "W", "3", "E", "4", "R", "5", "T", _
        "Y", "7", "U", "8", "I", _
        "O", "0", "P", "-", "[", "=", "]", _
        "\")
    For i = 0 To 36
        key(i).Caption = vbCrLf & vbCrLf & vbCrLf & vbCrLf & _
            vbCrLf & vbCrLf & vbCrLf & vbCrLf & note_kbd(i)
    Next i
    instructions.AddItem vbTab & vbTab & "SHORTCUTS"
    instructions.AddItem "clear a note" & vbTab & vbTab & "BACKSPACE"
    instructions.AddItem "clear all notes" & vbTab & vbTab & "ESC"
    instructions.AddItem "play a note" & vbTab & vbTab & "the displayed LABEL"
    instructions.AddItem "toggle recording" & vbTab & vbTab & "SPACE"

    ListComPorts
    cboComm.Enabled = input1(0).value
End Sub

' ================================================================================
' ===============    KEY PRESSES    ============================================
' ================================================================================
Private Sub clear_KeyDown(KeyCode As Integer, Shift As Integer)
    eval_key (KeyCode)
End Sub
Private Sub Form_KeyDown(KeyCode As Integer, Shift As Integer)
    eval_key (KeyCode)
End Sub
Private Sub clear_note_KeyDown(KeyCode As Integer, Shift As Integer)
    eval_key (KeyCode)
End Sub
Private Sub key_KeyDown(index As Integer, KeyCode As Integer, Shift As Integer)
    eval_key (KeyCode)
End Sub

Private Sub play_KeyDown(KeyCode As Integer, Shift As Integer)
    eval_key (KeyCode)
End Sub
Private Sub search_KeyDown(KeyCode As Integer, Shift As Integer)
    eval_key (KeyCode)
End Sub
Private Sub start_stop_KeyDown(KeyCode As Integer, Shift As Integer)
    eval_key (KeyCode)
End Sub
Private Sub stop_btn_KeyDown(KeyCode As Integer, Shift As Integer)
    eval_key (KeyCode)
End Sub

Public Function eval_key(KeyCode As Integer) As Boolean
    eval_key = True
    Select Case KeyCode
        Case 8
            clear_last_note
```

```
Case 27
    clear_all_notes

Case 90
    pressed_key 0
Case 83
    pressed_key 1
Case 88
    pressed_key 2
Case 68
    pressed_key 3
Case 67
    pressed_key 4

Case 86
    pressed_key 5
Case 71
    pressed_key 6
Case 66
    pressed_key 7
Case 72
    pressed_key 8
Case 78
    pressed_key 9
Case 74
    pressed_key 10
Case 77
    pressed_key 11

Case 188
    pressed_key 12
Case 76
    pressed_key 13
Case 190
    pressed_key 14
Case 186
    pressed_key 15
Case 191
    pressed_key 16

Case 87
    pressed_key 17
Case 51
    pressed_key 18
Case 69
    pressed_key 19
Case 52
    pressed_key 20
Case 82
    pressed_key 21
Case 53
    pressed_key 22
Case 84
    pressed_key 23
Case 54
    pressed_key 23
Case 89
    pressed_key 24

Case 55
    pressed_key 25
Case 85
```

```
            pressed_key 26
        Case 56
            pressed_key 27
        Case 73
            pressed_key 28
        Case 79
            pressed_key 29

        Case 48
            pressed_key 30
        Case 80
            pressed_key 31
        Case 189
            pressed_key 32
        Case 219
            pressed_key 33
        Case 187
            pressed_key 34
        Case 221
            pressed_key 35
        Case 220
            pressed_key 36
        Case 32
            If start Then
                start = False
                start_stop.Caption = "Stop Recording"
                start_stop.BackColor = &HFF&
            Else
                start = True
                start_stop.Caption = "Start Recording"
                start_stop.BackColor = &HFF00&
            End If

        Case Else
            eval_key = False
    End Select
End Function

Private Sub key_KeyUp(index As Integer, KeyCode As Integer, Shift As Integer)
    sndPlaySound App.Path & "\wavs\stop.wav", SND_ASYNC
End Sub

Private Sub key_MouseDown(index As Integer, Button As Integer, Shift As Integer, X
As Single, Y As Single)
    pressed_key (index)
End Sub

Private Sub key_MouseUp(index As Integer, Button As Integer, Shift As Integer, X As
Single, Y As Single)
    sndPlaySound App.Path & "\wavs\stop.wav", SND_ASYNC
End Sub

Private Sub stop_btn_Click()
    sndPlaySound App.Path & "\wavs\stop.wav", SND_ASYNC
End Sub


' ================================================================================
' ===============    CLEAR FUNCTIONS    ========================================
' ================================================================================
Private Sub clear_Click()
    clear_all_notes
```

```
End Sub
Private Sub clear_note_Click()
    clear_last_note
End Sub

Public Sub clear_all_notes()
    input_index = 0
    Dim i As Integer
    i = 0
    For i = 0 To 49
        input_string(i) = 100
    Next i
    search_input.Caption = ""

    For i = 0 To 36
        Select Case i
            Case 12
                key(i).BackColor = &HC0FFFF
            Case 0, 2, 4, 5, 7, 9, 11, 14, 16, 17, 19, 21, 23, 24, 26, 28, 29, 31,
33, 35, 36
                key(i).BackColor = &HFFFFFF
            Case Else
                key(i).BackColor = &H0&
        End Select
    Next i
End Sub

Public Sub clear_last_note()
    Dim i As Integer
    input_string_str = ""
    If (input_index <= 1) Then
        clear_all_notes
    ElseIf (input_index > 0) Then
        Select Case last_note
            Case 12
                key(last_note).BackColor = &HC0FFFF
            Case 0, 2, 4, 5, 7, 9, 11, 14, 16, 17, 19, 21, 23, 24, 26, 28, 29, 31,
33, 35, 36
                key(last_note).BackColor = &HFFFFFF
            Case Else
                key(last_note).BackColor = &H0&
        End Select

        last_note = last_note - input_string(input_index - 1)
        key(last_note).BackColor = &HFFFF&
        input_string(input_index - 1) = 100
        input_index = input_index - 1
        i = 1

        input_string_str = Format(input_string(0))
        While (i < 49 And Not input_string(i) = 100)
            If (input_string(i) >= 0) Then
                input_string_str     =     input_string_str     &     "     +"     &
Format(input_string(i))
            Else
                input_string_str = input_string_str & " " & Format(input_string(i))
            End If
            i = i + 1
        Wend
        search_input.Caption = input_string_str
        DoEvents
    ElseIf (input_index = 0) Then
```

```
        search_input.Caption = ""
    End If
End Sub


' ================================================================================
' ==============    USER INPUT FUNCTIONS    ================================
' ================================================================================
Private Sub db_dir_Change()
    db_folder = db_dir.Text
End Sub

Private Sub db_dir_Click()
    db_dir.SelStart = 0
    db_dir.SelLength = Len(db_dir.Text)
End Sub

Private Sub input1_Click(index As Integer)
    Dim i As Integer
    i = 0

    cboComm.Enabled = input1(0).value

    For i = 0 To 36
        key(i).Enabled = input1(1).value
    Next i
End Sub

Private Sub output_name_Click()
    sndPlaySound App.Path & "\song_wavs\" & _
        song_names(Int(output_name.ListIndex)) & _
        "_orig.wav", SND_ASYNC
End Sub

Private Sub play_Click()
    Dim i As Integer
    Dim note As Integer
    note = first_note
    For i = 0 To input_index - 1
        note = note + input_string(i)
        sndPlaySound App.Path & "\wavs\" & Format(note) & ".wav", SND_SYNC
    Next i
End Sub

Private Sub start_stop_Click()
    Dim i As Integer
    If start Then
        start = False
        start_stop.Caption = "Stop Recording"
        start_stop.BackColor = &HFF&
        If (input1.Item(0)) Then
            input1(0).Enabled = False
            input1(1).Enabled = False
            For i = 0 To 36
                key(i).Enabled = False
            Next i
            input_usb_data
        End If
    Else
        timer1.Enabled = False
        start = True
        start_stop.Caption = "Start Recording"
```

```
        start_stop.BackColor = &HFF00&
        If (input1.Item(0)) Then
            input1(0).Enabled = True
            input1(1).Enabled = True
            For i = 0 To 36
                key(i).Enabled = True
            Next i
        End If
    End If
End Sub

' ================================================================================
' ==============    GETTING SEARCH STRING    ================================
' ================================================================================
Public Sub input_usb_data()
    timer1.Interval = 100
    timer1.Enabled = True
End Sub

Private Sub timer1_Timer()
    ' tell the DSP that I am ready to receive data
    timer1.Enabled = False
    MSComm1.CommPort = Int(cboComm.Text)
    MSComm1.PortOpen = True
    MSComm1.Output = "1"

    Timer2.Enabled = True
    dead = False
    Do
        DoEvents
    Loop Until (MSComm1.InBufferCount >= 2 Or start = True Or dead)

    Dim temp As String
    temp = MSComm1.Input
    Dim note As Integer
    note = Val(temp) - 28

    If Not (note = last_note) Then
        If (note >= 0 And note <= 36) Then
            If (last_note >= 0 And last_note <= 36) Then
                Select Case (last_note)
                    Case 12
                        key(last_note).BackColor = &HC0FFFF
                    Case 0, 2, 4, 5, 7, 9, 11, 14, 16, 17, 19, 21, 23, 24, 26, 28,
29, 31, 33, 35, 36
                        key(last_note).BackColor = &HFFFFFF
                    Case Else
                        key(last_note).BackColor = &H0&
                End Select
            End If

            If (input_index = 0) Then
                last_note = note
                input_string(input_index) = 0
                input_index = input_index + 1
                If (last_note >= 0 And last_note <= 36) Then
                    key(last_note).BackColor = &HFFFF&
                End If
            ElseIf (input_index < 50) Then
                input_string(input_index) = note - last_note
                last_note = note
                input_index = input_index + 1
```

```
                If (last_note >= 0 And last_note <= 36) Then
                    key(last_note).BackColor = &HFFFF&
                End If
            End If

            If (input_index < 50) Then
                If (input_string(input_index - 1) > 0) Then
                    search_input.Caption = search_input.Caption + "+"
                End If
                search_input.Caption        =        search_input.Caption        +
Format(input_string(input_index - 1)) + " "
            End If
        End If
    End If


    ' tell the DSP that I am done
    MSComm1.Output = "E"
    MSComm1.PortOpen = False
    timer1.Enabled = True
End Sub


Public Sub pressed_key(index As Integer)
    sndPlaySound  App.Path  &  "\wavs\"  &  Format(index)  &  ".wav",  SND_ASYNC  Or
SND_LOOP

    Select Case last_note
        Case 12
            key(last_note).BackColor = &HC0FFFF
        Case 0, 2, 4, 5, 7, 9, 11, 14, 16, 17, 19, 21, 23, 24, 26, 28, 29, 31, 33,
35, 36
            key(last_note).BackColor = &HFFFFFF
        Case Else
            key(last_note).BackColor = &H0&
    End Select
    key(index).BackColor = &HFFFF&

    If Not start Then
        If (input_index = 0) Then
            last_note = index
            first_note = index
            input_string(input_index) = 0
            input_index = input_index + 1
        ElseIf (input_index < 50) Then
            If (Not (index = last_note)) Then
                input_string(input_index) = index - last_note
                last_note = index
                input_index = input_index + 1
            End If
        End If

        input_string_str = Format(input_string(0))
        Dim i As Integer
        i = 1
        While (i < 49 And Not input_string(i) = 100)
            If (input_string(i) >= 0) Then
                input_string_str     =     input_string_str     &     "     +"     &
Format(input_string(i))
            Else
                input_string_str = input_string_str & " " & Format(input_string(i))
```

```
            End If
            i = i + 1
        Wend
        search_input.Caption = input_string_str
    End If
    key(index).SetFocus
End Sub

' ================================================================================
' ==============    SEARCH FUNCTIONS    ========================================
' ================================================================================
Private Sub search_Click()
    Timer3.Interval = 10
    search_time = 0
    Timer3.Enabled = True
    DoEvents
    searching.Visible = True
    Form1.Enabled = False
    DoEvents

    file_list.Path = db_folder

    Dim i As Integer
    Dim num_found As Long

    ReDim song_names(file_list.ListCount - 1)
    ReDim song_occurences(file_list.ListCount - 1)

    For i = 0 To file_list.ListCount - 1
        If Not (file_list.List(i) = "") Then
            DoEvents
            song_names(i) = file_list.List(i)
            song_occurences(i) = search_song(db_folder & file_list.List(i))
        End If
    Next i

    BubbleSort song_occurences, song_names, file_list.ListCount - 1, True

    Dim total As Double
    Dim percent As Integer
    total = 0
    For i = 0 To file_list.ListCount - 1
        If Not (file_list.List(i) = "") Then
            total = total + song_occurences(i)
        End If
    Next i

    output_name.clear
    For i = 0 To file_list.ListCount - 1
        If Not (file_list.List(i) = "") And Not (song_occurences(i) = 0) Then
            percent = song_occurences(i) / total * 100
            output_name.AddItem    Format(percent)    &    "%"    &    vbTab    &
Format(song_occurences(i)) & vbTab & song_names(i)
        End If
    Next i

    DoEvents
    searching.Visible = False
    Form1.Enabled = True
    DoEvents
    Timer3.Enabled = False
    MsgBox (search_time)
```

```
End Sub

Public Function search_song(theSong As String) As Integer
    Dim song_notes() As Long
    Dim song_lengths() As Long

    Dim temp As String
    Dim i As Integer
    Dim j As Integer
    Dim k As Integer
    Dim m As Integer

    Dim size As Integer
    Dim length As Long

    Open theSong For Input As #1
        ' set the size of the arrays
        Line Input #1, temp
        size = Int(temp)
        ReDim song_notes(size)
        ReDim song_lengths(size)
        i = 0
        length = 0

        ' read in all the song data
        While Not EOF(1)
            DoEvents
            Line Input #1, temp
            song_notes(i) = Int(temp)
            Line Input #1, temp
            song_lengths(i) = Int(temp)
            length = length + Int(temp)
            i = i + 1
        Wend
    Close #1

    Dim search_window_size As Double
    Dim search_window_length As Double
    Dim search_window As Long

    search_window_length = 0
    search_window_size = size / 2
    For i = size / 4 To 3 * size / 4
        search_window_length = search_window_length + song_lengths(i)
    Next i
    search_window   =   Int(search_window_length   *   Int(window_factor.Caption)   /
search_window_size)

    Dim search_string(50) As Long
    For i = 0 To UBound(input_string) - 1
        search_string(i) = input_string(i)
    Next i

    Dim allow_skip_long As Long
    If allow_skip Then
        allow_skip_long = 1
    Else
        allow_skip_long = 0
    End If

    Dim skip_weight_long As Long
    skip_weight_long = Int(skip_weight.Caption)
```

```
    search_song = find_occurences_c_dll(search_string(1), _
        input_index - 1, song_notes(0), song_lengths(0), _
        UBound(song_notes) - 1, search_window, allow_skip_long, skip_weight_long)
End Function

Sub BubbleSort(arr1 As Variant, arr2 As Variant, _
    Optional numEls As Variant, Optional descending As Boolean)

    Dim value As Variant
    Dim value2 As Variant
    Dim index As Long
    Dim firstItem As Long
    Dim indexLimit As Long, lastSwap As Long

    ' account for optional arguments
    If IsMissing(numEls) Then numEls = UBound(arr1)
    firstItem = LBound(arr1)
    lastSwap = numEls

    Do
        indexLimit = lastSwap - 1
        lastSwap = 0
        For index = firstItem To indexLimit
            value = arr1(index)
            value2 = arr2(index)
            If (value > arr1(index + 1)) Xor descending Then
                ' if the items are not in order, swap them
                arr1(index) = arr1(index + 1)
                arr2(index) = arr2(index + 1)
                arr1(index + 1) = value
                arr2(index + 1) = value2
                lastSwap = index
            End If
        Next
    Loop While lastSwap
End Sub

' CODE FROM: http://www.developerfusion.co.uk/show/21/2/
Public Sub ListComPorts()
    Dim i As Integer

    cboComm.clear
    For i = 1 To 16
        If COMAvailable(i) Then
            cboComm.AddItem i
        End If
    Next
    cboComm.ListIndex = 0
End Sub

Private Sub sw_minus_Click()
    If (Int(skip_weight.Caption) > 0) Then
        skip_weight.Caption = Int(skip_weight.Caption) - 1
    End If
End Sub

Private Sub sw_plus_Click()
    skip_weight.Caption = Int(skip_weight.Caption) + 1
End Sub

Private Sub Timer2_Timer()
```

```
    dead = True
End Sub

Private Sub Timer3_Timer()
    search_time = search_time + 10
End Sub

Private Sub wf_minus_Click()
    If (Int(window_factor.Caption) > 0) Then
        window_factor.Caption = Int(window_factor.Caption) - 1
    End If
End Sub

Private Sub wf_plus_Click()
    window_factor.Caption = Int(window_factor.Caption) + 1
End Sub
```

REFERENCES

[1] Y. Li and D. L. Wang, “Detecting Pitch of Singing Voice in Polyphonic Audio.” *Proceedings of ICASSP-05*, pp. III.17-20, 2005.

[2] Y. Li and D. L. Wang, “Separation of Singing Voice from Music Accompaniment for Monaural Recordings.” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, pp. 1475-1487, 2007.

[3] P. Moulin, “ECE 418 Lecture Notes: Introduction to Image and Video Processing,” class notes for ECE 418, Department Of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, 1998.

[4] National Semiconductor, “LM124/LM224/LM324/LM2902, Low Power Quad Operational Amplifiers,” May 1999, http://pubpages.unh.edu/~aperkins/pdf/LM-devices/LM324.pdf.

[5] eCircuit Center, “Sallen-Key Low-Pass Filter,” January 2002, http://www.ecircuitcenter.com/Circuits/opsalkey1/opsalkey1.htm.

[6] Sparkfun Electronics, “SFE USB Drivers,” April 27, 2005, http://www.sparkfun.com/datasheets/SFE_USB_Drivers-v011.zip.

[7] Sparkfun Electronics, "USB Breakout CP2102 Datasheet," April 27, 2005, http://www.sparkfun.com/datasheets/PCB/CP2102%20Breakout-v01.pdf.

[8] Maxim – Dallas Semiconductor, "+5V-Powered, Multichannel RS-232 Drivers/Receivers," February 2003, http://www.ortodoxism.ro/datasheets/maxim/MAX220-MAX249.pdf.

[9] J. Fruits, “Creating a C++ DLL for use with VB6 – Step by Step,” January 2005, http://www.programmers-corner.com/tutorial/4.

[10] Wikipedia, “Piano Key Frequencies,” April 2007, http://en.wikipedia.org/wiki/Piano_key_frequencies.