Design Specification for the

Automatic Music Transcriber

Revision: 1.0

Prepared for: Dr. Andrew Rawicz – ENSC 440
Steve Whitmore – ENSC 305
School of Engineering Science
Simon Fraser University

Prepared by: Mike Tyson
Henry Huang
Patrick Wong
Shu Hui Wong

Contact Person: Shu Hui Wong
shw4@sfu.ca

Date: March 8th 2012

Executive Summary

The design specification for the Automatic Music Transcriber (AutoTab) provides a set of detailed descriptions for the design and development of our proof-of-concept model. The design specifications in this document are solely for the proof-of-concept model, therefore, we will only discuss design considerations pertaining to the functional requirements up to the proof-of-concept deliverables, as specified in the document *Functional Specification for the Automatic Music Transcriber* [1].

This document outlines the design of the AutoTab and provides justification for our design choices. Design improvements for future iterations of the AutoTab are also discussed. The user interface will consist of an array of buttons, a switch for mode adjustments, a dial for volume control, as well as an LCD and status LEDs. A line-in jack will allow for direct sampling of the audio source while the SD-card slot and USB ports will enable ease of storing and transfer of the audio files. An FPGA board will be used to read inputs, apply a conversion of audio files to MIDI algorithm, and carry out the transcription algorithm.

A detailed description of the resource requirements and selection criteria for a suitable FPGA board is provided. General software program process flow is also included and a description of test plans for the system and its subcomponents is provided at the end of the design specification.

Our initial proposed 13-week engineering cycle for this project is on schedule and the scheduled completion date for an operational prototype of our proof-of-concept model is April 2nd 2012.

Table of Contents

List of Figures

List of Tables

Glossary

BIN Value	Derived from the number of samples in an FFT divided by two metric used to describe resolution
CSA	Canadian Standards Association
DAW	Digital Audio Workstation – system designed for playback, recording, and editing of digital audio
DE2	Altera Development and Educational FPGA equipped board
MIDI	Musical Instrument Digital Interface – industry specification for the encoding, storage, transmission, and synchronization of instrument recordings and control data for MIDI compatible devices
NIOS	Embedded-processor architecture designed for the Altera family of FPGAs
Transcription	The process of writing down music that otherwise was not – such as an improvised piece
Quantization	The process of converting a continuous signal to a discrete sample set
FPGA	Field-Programmable Gate Array – an integrated circuit designed to be configured by the customer/designer after manufacturing
USB	Universal Serial Bus - an industry standard developed to define the communication protocols used in a bus for connection, communication and power supply between computers and electronic devices
SD-card	Secure Digital Card –a non-volatile memory card
LED	Light-emitting diode – a semiconductor light source
LCD	Liquid crystal display – a flat panel display that uses the light modulating properties of liquid crystals
FFT	Fast Fourier Transform – algorithm that decomposes a sequence of values into components of different frequencies
ASIC	Application-specific integrated circuit – an integrated circuit customized for a particular use, rather than intended for general-purpose use
Microcontroller	Small integrated circuit containing a processor core, memory, and programmable input/output peripherals
Ribbon Cable	A cable with many conducting wires running parallel to each other on the same flat plane

Musical Theory

Monophonic	Describes music consisting of single notes or a single melodic line
Polyphonic	Describes music with multiple melodic lines or multiple notes played simultaneously such as chords
Staff	Musical notation consisting of five horizontal lines and four spaces each representing a different musical pitch
Tempo	Musical notation describing the pace or speed of the piece
Beat	Represents the natural rhythm in a piece of music: tempo is usually described in beats-per-minute (BPM)
Signature	Musical notation describing the number of beats and length of notes in a given bar – contemporary western music is often ‘4/4’, which indicates that there are four beats in a bar (numerator) and that each beat is represented by a quarter note (denominator)
Bar	Musical notation subdividing the notes of a song
Note	Musical notation indicating a particular pitch to be held for an amount of time - a ‘whole’ note being the longest division, with shorter intervals represented by fractions of this value
Tuning	The process of adjusting the pitch of a musical instrument such that each key/string is separated by a known pitch interval
Fundamental	The lowest frequency signal present in a given musical note
Harmonic	Any whole-number multiple of the fundamental frequency present in a note

1. Introduction

The Automatic Music Transcriber (AutoTab) is a portable lightweight device that will allow for the conversion of audio recordings into written staff. Users will also be able to use it as a metronome, a tuner and also for audio manipulation. Our aim is to create an all-in-one device that not only makes music creation an easier task, it will also take away the frustration that comes with having to deal with more than one device for all their musical needs. The design specification describes the technical details for the design of the AutoTab.

1.1 Scope

This document specifies the design of the AutoTab and explains how the design meets the functional requirements as described in*Functional Specification for the Automatic Music Transcriber* [1]. The design specification includes all requirement features of the proof-of-concept system and a partial set of requirements for the encapsulated product. Since we are focusing on the proof-of-concept system, only design considerations pertaining to the function requirements up to our proof-of-concept deliverables as mentioned in *Functional Specification for the Automatic Music Transcriber* [1] will be explicitly discussed. The appendices include process flow charts to help facilitate the implementation of the AutoTab.

1.2 Intended Audience

The design specification is intended for use by all members of ScribeWare Inc. Design engineers shall refer to the specifications as overall design guidelines to ensure all requirements are met in the final product while test engineers shall use this document to implement the test plan and to confirm the correct behavior of the AutoTab.

2. System Specifications

The AutoTab will record audio through a line-in jack and convert a copy into MIDI format before transcribing it onto staff. The AutoTab will perform this task automatically after the user instructs it to stop recording through the press of a button. The user will also have the option of switching modes to go between the metronome, tuning and recording functionalities through the interface.

The total process of performing an audio recording to producing transcribed staff will be completed in a time frame within 50% the length of the recording being transcribed.

3. Overall System Design

This section provides a high-level overview of the entire design and the system can be modeled as shown below in Figure 1.

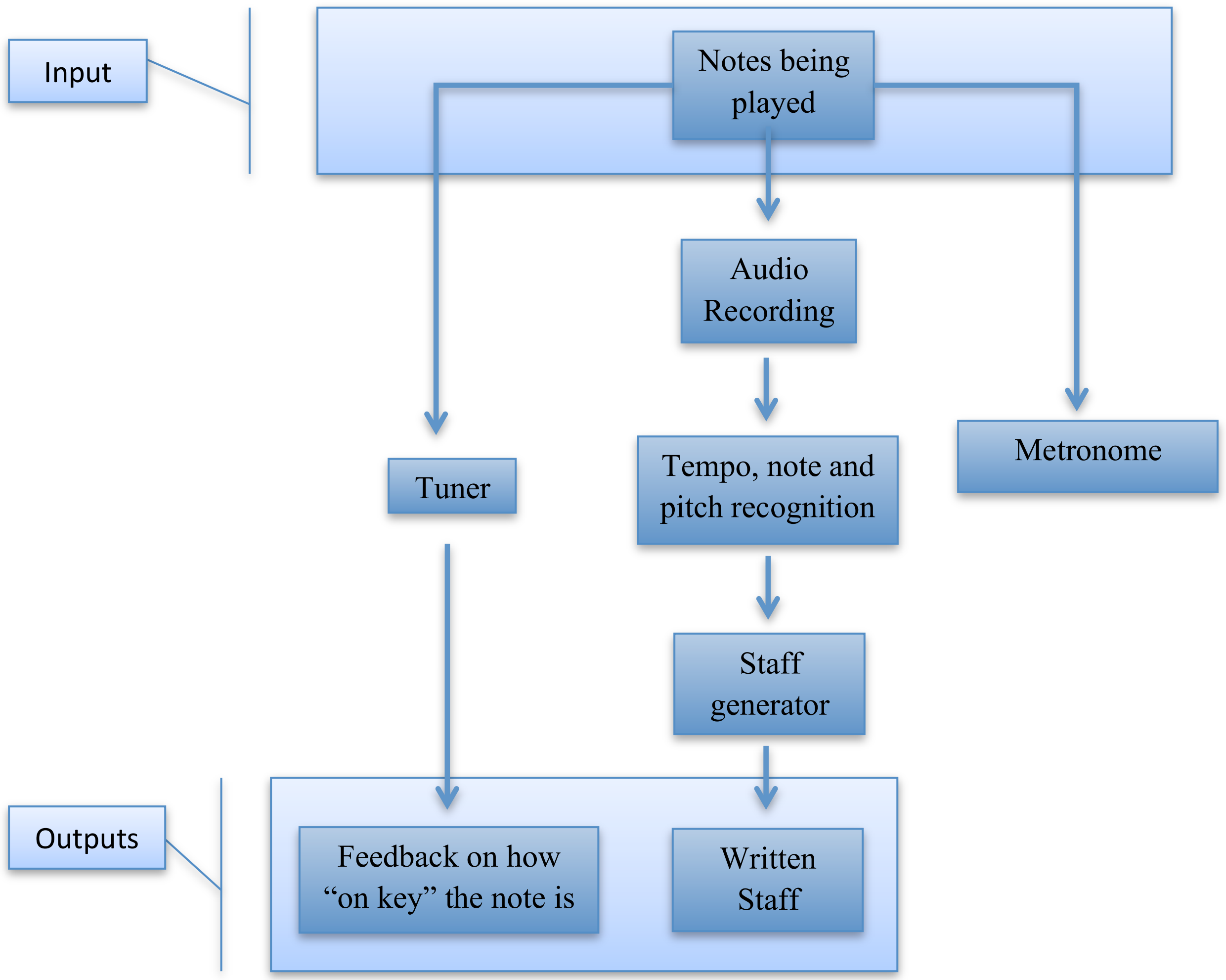

Figure 1: System Overview

Design details that are common to AutoTab as a proof-of-concept model will be discussed in the upcoming sections, while design details specific to each individual functionality will be discussed in the upcoming sections as part of the algorithm design specification.

3.1 High-level System Design

This section provides a high-level overview of the entire system. Figure 2 depicts a block diagram showing the inputs and the outputs along with the relationship between subsystems.

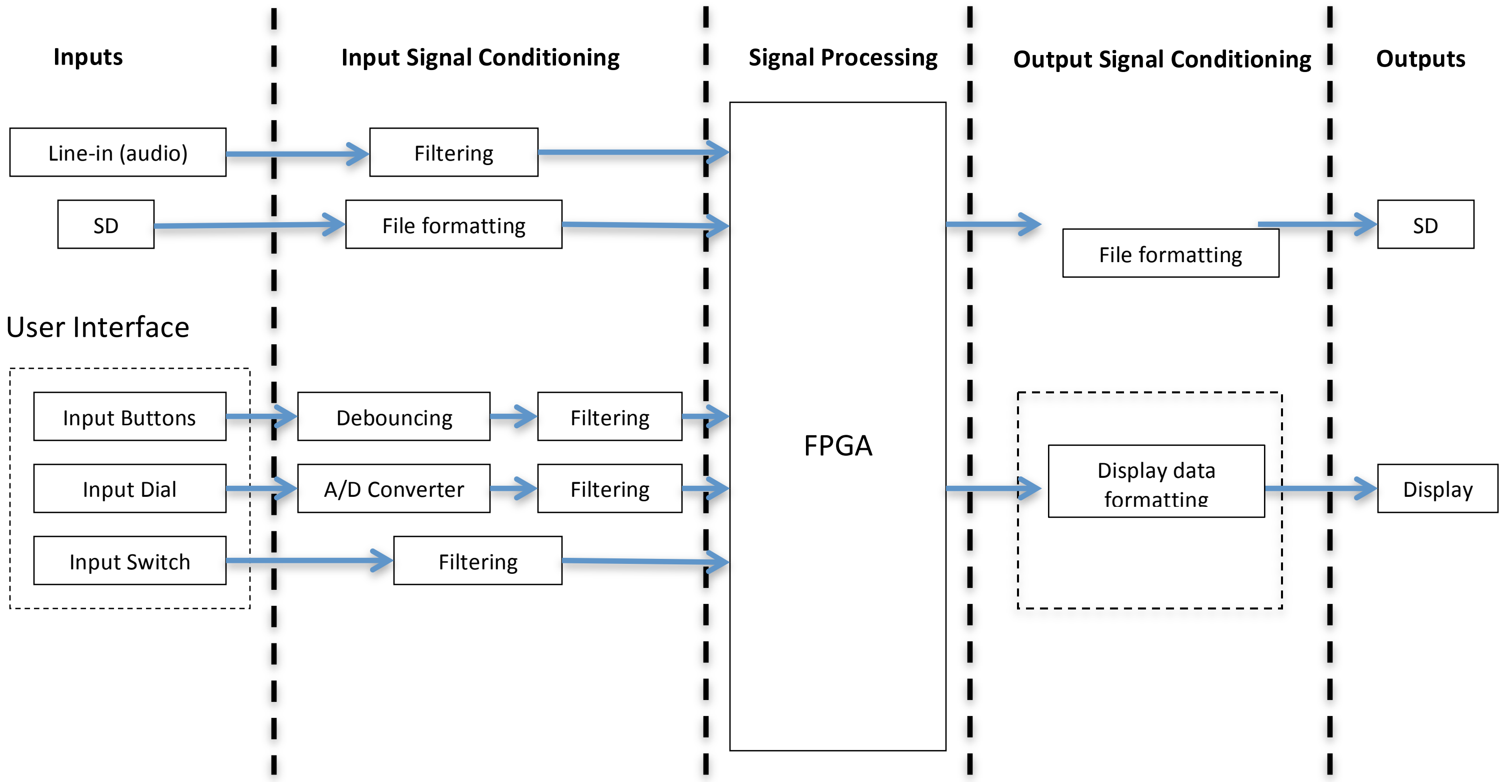

Figure 2: System Block Diagram

Systems inputs include user input buttons, audio line-ins and also ports for SD cards. These inputs are conditioned for use through signal filtering, A/D conversion and also file formatting stages.

The signal processing stage consists of the FPGA, which contains the control algorithm that provides the logic used to synchronize, monitor and activate all other system components. An FPGA was chosen to carry out our processes instead of a microcontroller because it has an in-built set of peripherals (audio codec, array of programmable buttons, SD-card slot) that brings together all the components that were needed in the making of the AutoTab into one ready-made unit.

In the output signal conditioning stage, LCD data is formatted for display transmission while the LED data is driven directly by the FPGA.

3.2 Electrical System

The AutoTab will run off the power supplied through our FPGA, which will provide a power supply voltage of 12V. Most of our electrical considerations will only affect the final product as the proof-of-concept model will be built off the FPGA.

3.2.1 Safety Considerations

Since our proof-of-concept model is based off an FPGA, our electronic components are sufficiently protected already.

However, in future iterations, to protect electronic components from damage and prevent harm to developers and users, the following precautions will be taken to enforce electrical safety in the encapsulated model of the AutoTab:

- Proper insulation of wires
- Use of current sensing circuitry
- Use of fuses in high current areas
- Use of heat-sinks to dissipate heat

Heat-sinks were chosen for heat dissipation because they are relatively inexpensive, can operate at quiet speeds and does not add complexity to the system.

3.2.2 Noise Considerations

To reduce the detrimental effects of noise in the electrical systems in our final product, the following steps will be taken:

- Utilizing sufficient signal strength
- Maintaining a high signal-to-noise ratio
- Routing wires carefully to minimize unnecessarily long wire lengths

3.3 Power Supply

The following power requirements of our proof-of-concept model of the AutoTab must be met:

- Voltage: 12V DC
- Can be run from a wall electrical outlet
- Regulated
- AC to DC conversion

In future iterations of the AutoTab, it will run using a lithium coin battery [2].

4. Algorithm Specifications

This section provides a design overview the transcriber and tuner/metronome algorithms. The transcription algorithm is based upon the papers *Automatic Music Transcription Using Genetic Algorithms and Electronic Synthesis* [3] and *Electronic Synthesis Using Genetic Algorithms for Automatic Music Transcription* [4], in which each transcription is based upon a time slice of the input audio.

4.1 Genetic Algorithm Design for Transcription

The genetic algorithm starts off by first creating a gene pool which contains a selection of the possible time slices. Each time slice is first passed into the Renderer, which transforms the musical representation into the audio representation (MIDI) before having its fitness evaluated by using a FFT to compare the power spectrums between the MIDI formats of both the original and new time slices.

Through these comparisons, the least-fit genes will be removed from the gene pool, while the more-fit ones will live on and undergo mutations to create better versions of themselves. This process results in ever-increasingly fit genetic material, which for us, means better and better transcriptions of the music.

This program flow is shown in the figure below.

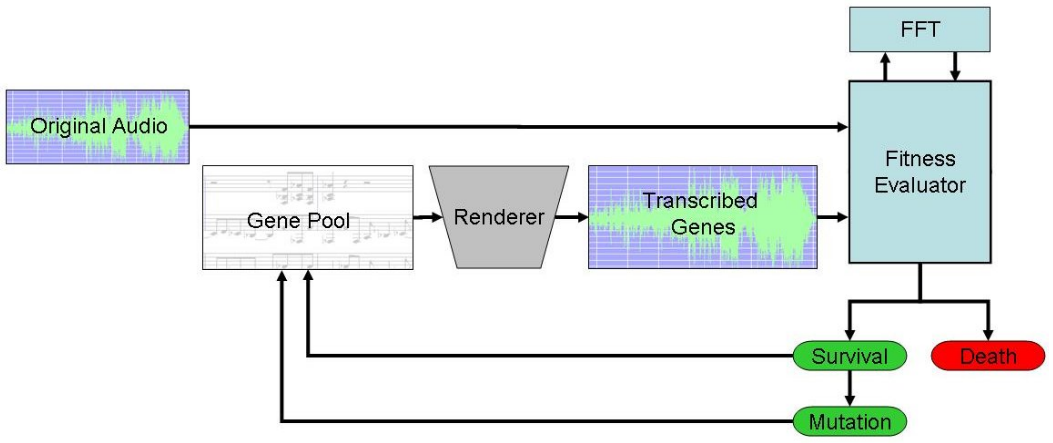

Figure 3: Program Flow of the Genetic Algorithm [3]

4.1.1 Framing and Quantization

Before the notes in a given recording can be analyzed, the audio file must first be rendered into a format that is more easily handled by the pitch detection algorithm. This is a two-stage process: first the recording is divided into equally sized ‘frames’ (segments of audio), then the actual notes are identified and clipped or extended to fully occupy a whole number of frames – this is quantization. Thus, once completed, the pitch detection algorithm can process the file frame by frame, and need only address those segments containing useful musical data.

4.1.1.1 Framing

The framing algorithm divides the recording into subdivisions of 1/64 of a bar in length (equivalently a 64th note). This number was selected for several reasons:

- When generating a MIDI file, timing between events is defined and specified as a fraction of a beat. By similarly defining our frame size to correspond to note length (instead of grouping an arbitrary number of audio samples) the conversion to MIDI is greatly simplified, especially when these two values are made equal.
- In most DAWs the smallest resolution to which events can be manually drawn is a 64th note.
- In the majority of guitar music, with the exception of strumming it is unusual to observe notes smaller than 1/32 of a bar.

More generally, subdividing the audio into a whole number of frames simplifies computation by enabling an iterative approach.

The first step is calculating the number of samples per frame – this is based on the sample rate of the file and the tempo of the recording:

$$\frac{\text{Sample Rate }(\frac{\text{samples}}{\text{s}})}{\text{BPM }(\frac{\text{1/4 notes}}{\text{min}})} \times \frac{60\text{s}}{1/16\text{ notes}} = \frac{\text{samples}}{\text{frame}}$$

For a sample rate of 44.1kHz (lower CD quality) and at a tempo of 120 BPM, there would be:

$$\frac{44100}{120} \times \frac{60}{16} = 1378.125 \text{ samples per frame.}$$

As this is not a round number and a sample cannot be divided, the frame size is rounded down (1378 samples, in this case). In order to keep in time with the audio, the framing algorithm skips samples to make up for the small fraction being lost every frame. This is determined from the decimal part of the original frame size:

0.125 ➔ 1/8 ➔ 1 sample skipped per 8 frames

There is no guarantee, however, that a given audio clip will contain a whole number of frames, therefore the algorithm must also pad the end of the file with empty samples to round it out. In order to maintain the notational integrity of the piece being transcribed, this process is repeated anew in order to achieve a whole number of bars.

4.1.1.2 Quantization

The function of the quantization algorithm is to determine the placement of the notes in the recording and to limit or extend their positions into occupying a whole number of frames. The motivation is twofold: first, as per the framing algorithm, we wish to express all events in a song in 64th note steps. Perhaps more importantly still, in order for the FFT functions in the pitch detection algorithm to work, the audio in a given frame must be stable from beginning to end – if a note were to start or end halfway through it would corrupt the analysis.

The quantization process is as follows: upon completion of the framing process, the recording is first rectified (Figure 4) then each frame's magnitude is averaged.

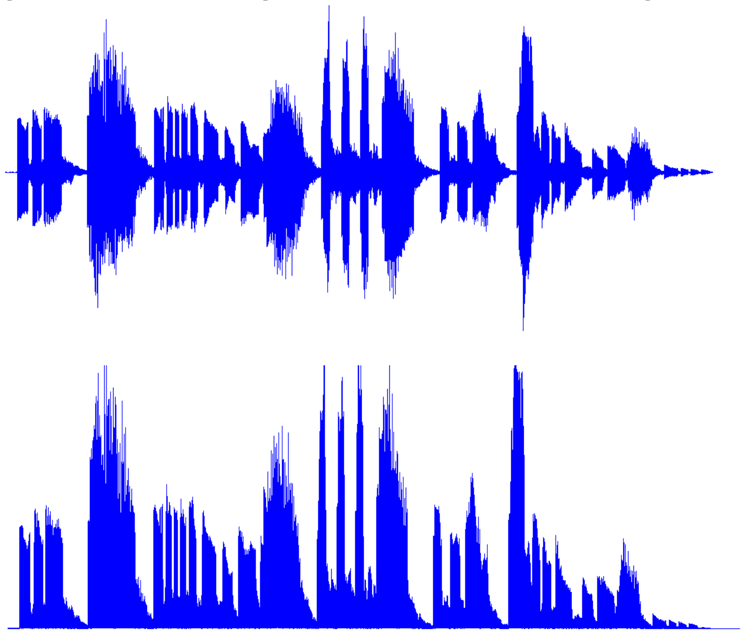

Figure 4: Bipolar recording (top), rectified clip (bottom)

The average amplitude for each bar in the recording, as well as for the entire file are calculated, which are used to generate both an attack and a release threshold. If the magnitude of a signal crosses the attack threshold (the higher of the two), a new note is detected and sustained as long as it remains above the release threshold. In Figure 5, we can see the clip from Figure 4 with the amplitudes averaged per frame (black) superimposed with the average amplitude for the clip (blue) and the detected notes (magenta). Note that in this case, the clip in question was used purely for testing and is therefore not set into any known grid. Thus, the tempo has been defaulted to 120 BPM, and the threshold set to be invariant from bar to bar, as this information is not available.

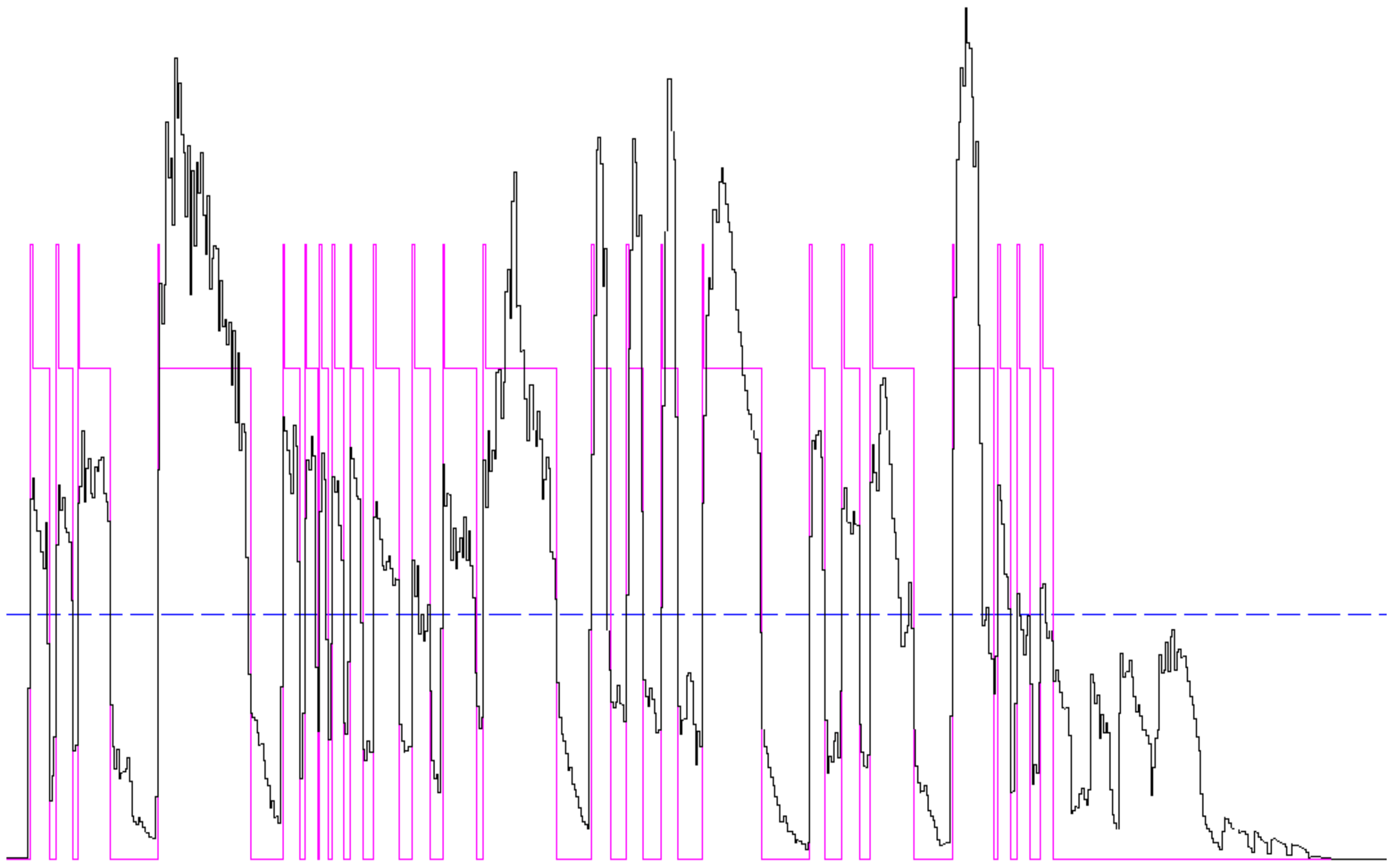

Figure 5: Quantized Clip

Once the notes have been quantized, the rest of the audio can be discarded, maintaining a whole number of bars.

4.1.2 Pitch Extractor

The pitch extractor uses a genetic algorithm that consists of a gene pool, fixed length FFT, fitness evaluator, and mutation function.

The gene pool consists of genomes organized in a hierarchy organized as follows:

Genome	The genome encapsulates the gene and serves as a basic container for the candidate
Gene	The gene represents the guitar and contains approximately twelve base pairs representing its notes
Base Pair	A base pair contains frequency information, start time, and duration of a note

The possible mutations that can be applied to a gene are as follows:

Irradiate	Change a random feature of the base pair
Nudge	Bump the frequency of a base pair by a semitone
Reclassify	Swap base pairs with another gene
Assimilate	Take a section of gene and copy it to another genome to form a chord

The steps that are taken to detect pitch are:

Step 1 – The incoming audio sample is truncated or padded to 4096 data values to which a Hann window is applied before it is then run through a fixed length FFT

Step 2 – A candidate is selected from the gene pool, a Hann window is applied to the data, and it is run through a fixed length FFT

Step 3 – The FFT of both these samples is then evaluated in the Fitness function to determine how close of a match they are

Step 4 – Steps 2 and 3 are repeated until all of the starting candidates have been evaluated for fitness. The lowest fitness value is then used to calculate the scaling factor σ in the fitness function

Step 5 – A third of the candidates with the lowest fitness value are eliminated from the gene pool

Step 6 – The top third of the candidates are selected for random mutations and reintroduced to the gene pool

Step 7 – Steps 2, 3, 5, and 6 are repeated until the fitness function returns a value of 0.99 or the fitness value hasn't improved in the last 200 iterations

Step 8 – The successful gene is then stored for the final transcription

4.1.3 Fitness Evaluator

The evaluation of the fitness of each transcription is done in the frequency domain because a straight comparison, which measures the difference between samples, would result in similar sounds being rejected.

The fitness of any particular gene is defined in Lu's paper as:

$$Fitness = \frac{1 - \sum_{t=0}^{tmax} \sum_{f=fmin}^{fmax} \big(O(t,f) - X(t,f)\big)^2}{\sigma}$$

where*O(t,f)* is the magnitude of the frequency at time *t* in the input audio, and *X(t,f)* is the same for the possible transcription. $\sigma$is a scaling factor equivalent to the first worst transcription. This scaling factor puts most fitnesses in the range [0,1], with 1 being a perfect match.

However, this was determined to be incorrect because the scaling factor would incorrectly alter the fitness value. As $\sum_{t=0}^{tmax} \sum_{f=fmin}^{fmax} \big(O(t,f) - X(t,f)\big)^2$ approaches 0 indicating convergence on the correct frequency, the fitness function becomes $\frac{1}{\sigma}$ and not 1 as expected. Our corrected fitness function is:

$$Fitness = 1 - \frac{\sum_{t=0}^{tmax} \sum_{f=f\square in}^{fmax} \big(O(t,f) - X(t,f)\big)^2}{\sigma}$$

Where σ which is initially 1 is then is calculated using the lowest returned fitness value after the first pass using the following equation:

$$\sigma = -lowest\ Fitness + 1$$

The selection process of which samples should be removed and which stays is strictly survival of the fittest. The algorithm kills off the bottom third of the population and for each sample in the top third, two mutations are created and added into the gene pool.

4.1.4 Mutations

The two mutations that each transcription will undergo are done in the frequency domain. After the mutations, we will end up with three samples, one being the original, one with a slightly lowered frequency and another with a slightly higher frequency. This is done to take into consideration notes that are only marginally out of tune. The amount at which the frequencies will increase/decrease by will be determined through testing.

4.1.5 Termination Condition

There are two conditions with which the transcription algorithm can be terminated. First, if the fitness of the best individual exceeds 0.99, then the program assumes that we have the correct answer and terminates. However, we also have to account for the case that the fitness does not converge to 1. For this, we consider the number of generations that have gone by since the last improvement in fitness. If that number is greater than the given limit of 200 iterations, the program gives up and terminates.

4.2 Tuner/Metronome Algorithm Design

The algorithm involved with tuning the guitar will be implemented by sampling the audio of the string being tuned, using an FFT to process the recording, and identifying the fundamental frequency of the sampled recording. When the user is tuning the guitar, at most only one string should be playing, thus a dominant fundamental frequency should be apparent in the sampled audio. Once the fundamental frequency is identified, the difference between it and the closest matching note will be calculated and feedback will be provided to the user as to whether or not the string is tuned too high, too low, or in tune.

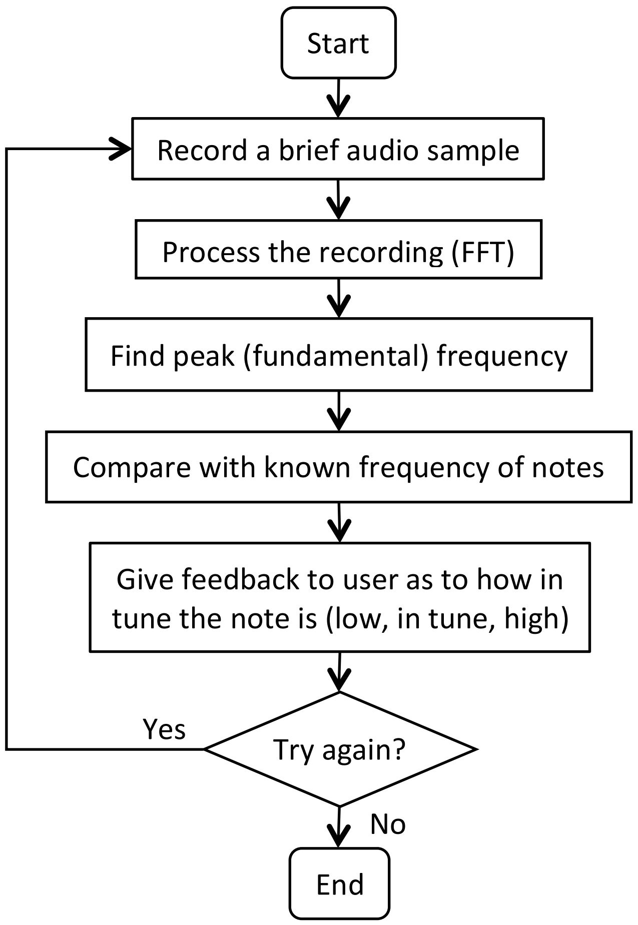
Start
Record a brief audio sample
Process the recording (FFT)
Find peak (fundamental) frequency
Compare with known frequency of notes
Give feedback to user as to how in tune the note is (low, in tune, high)
Yes
Try again?
No
End

Figure 6: Tuner System Flowchart

For the metronome, the user will need to input the tempo they will be playing at. Once this information is received, the system will play an audible click or tone to keep the user in time. A timer will be used to keep track of when the sound should be played.

5. User Interface Unit

The user interface allows the user to control what they would like to use the AutoTab for. For the final product, in addition to the switch that determines which mode we are in and a knob for volume control, there are also 4 buttons for navigation in between menus, to start/stop recording and to cancel/confirm a selection. The user interface will also feature an LCD and LEDs for message displays to the user for debugging purposes.

However, in our proof-of-concept model, volume control and mode determination will be accomplished using buttons.

5.1 User Interface Hardware

The hardware will consist of the LCD, LEDs, buttons, dials and switches.

5.1.1 LCD and LEDs

The LCD will display current mode, status and error messages. LEDs will be used to signal pitch position in the tuning mode and as a visual status for metronome usage. The data from the FPGA will be formatted for display on the LCD and both the LCD and the LEDs will be used for debugging purposes.

5.1.2 Buttons

The functionalities related with the AutoTab will be controlled using buttons. Table 1 lists all the user input functions and the number of buttons associated with it.

Table 1: Button Specifications

Function	Number of Buttons
Start/Stop Recording	1
Cancel/Confirm selection and Options menu	1
Menu navigation	2
Volume control	1

5.1.3 Switch (final product feature)

A sliding switch will control in future iterations of the AutoTab, the functional mode that the AutoTab is in. The sliding switch will determine whether or not the AutoTab is in recording mode or in tuning mode. Figure 4 below shows the power switch for our final product.

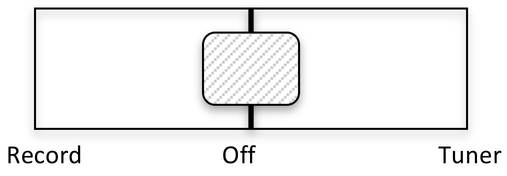

Figure 7: Power Switch

5.1.4 Knob (final product feature)

In future iterations, the volume control of the metronome clicks and audio playback will be adjusted using a knob. A volume knob will replace the buttons used in our proof-of-concept model because it proves to be more intuitive for users.

5.2 User Interface Software

The software will take the inputs from the buttons, switch and knob and process them. The action that corresponds to the pressed button or dial will be completed within 500ms.

5.2.1 Button Identification

The buttons will be mapped to interrupts so that when they are pressed, an interrupt service routine will execute to determine which button was pressed. In our proof-of-concept model, the buttons that are on the board will be used, and thus, we would not have to worry about debouncing the button inputs. However, in future iterations, debouncing has to be done in order to properly read a button input. Introducing a small delay before sampling the button input can do this. The required delay time will be determined through testing.

5.2.2 Switch Identification (final product feature)

The switch will also be mapped to interrupts, like the buttons. However, unlike the buttons, no debouncing is required for the switch.

5.2.3 Knob Identification (final product feature)

As with the switch, the knob will be mapped to interrupts and require no debouncing.

5.2.4 Display

The LCD will display information to indicate which mode the AutoTab is currently on. It will also indicate memory status, the time elapsed since a recording has started, and also provide feedback for tuning and metronome statuses. If any errors occur, an error message will be displayed asking the user to reset the device.

5.3 User Interface Verification

To test the user interface, we must perform the following tests:

1. Pushing the buttons to cause an interrupt to occur
2. Display error messages on the LCD
3. Light the LEDs for visualization of metronome beats and tuning status
4. When the Start/Stop Record buttons are pressed, the corresponding LED is lit and the LCD displays a message

6. System Test Plan

The proof-of-concept model of the AutoTab requires intensive testing on our transcription algorithm and also that of our tuner/metronome algorithm. After individual section testing on our algorithms, the ideal operation of the AutoTab is examined under normal and extreme conditions.

6.1 Unit Testing

To verify that each of our three functionalities is working properly, we plan to test them each separately. To confirm that each function is fully functional, the following tests must be performed.

1. Verify that audio recording, tuning and the metronome works separately
2. Monitor the LCD and LEDs for verify that they act accordingly to the requirements mentioned in *Functional Specifications for the AutoTab*[1].
3. Observe the mean time of failure for all the hardware of our device and verify that it agrees with the document, *Functional Specifications for the AutoTab* [1].
4. Observe the time needed for the tuner to show a result and ensure that it is within 2 minutes.

5. Verify that when the metronome and audio recording is done concurrently, that the metronome clicks are not recorded as well.
6. Verify that our transcription algorithm is able to properly identify single notes and chords.
7. Verify that the device will not malfunction if dropped from a short distance.

6.2 Normal Case 1: Audio Recording with sufficient memory

User Input: The user presses the "Start Record" button and starts playing.

Conditions: The guitar is plugged into the audio line-in, the device is in "Record" mode, and there is sufficient internal memory or SD card space for the recording.

Expected Observations: An LED lights up and the LCD displays a message that the device is currently recording. Recording is successful until user stops the recording and that session is saved.

6.3 Normal Case 2: Tuner – Single Notes Played

User Input: The user plays a note.

Conditions: The guitar is plugged into the audio line-in and the device is in "Tuner" mode.

Expected Observations: An LED lights up as the note is being processed and the LCD displays a message that the device is currently tuning. Within 2 minutes, the LED will turn off and the LCD will display a message stating that the result has been achieved.

6.4 Normal Case 3: Tuner – Concurrent Notes Played

User Input: The user plays a note, followed by another one.

Conditions: The guitar is plugged into the audio line-in and the device is in "Tuner" mode.

Expected Observations: An LED lights up as the first note is being processed and the LCD displays a message that the device is currently tuning. Within 2 minutes, the LED will turn off and the LCD will display a message stating that the result has been achieved. The second note played will be ignored.

6.5 Extreme Case 1: Audio Recording with low memory

User Input: The user presses the “Start Record” button and starts playing.

Conditions: The guitar is plugged into the audio line-in, the device is in “Record” mode, and there is insufficient internal memory or SD card space to save/process the recording.

Expected Observations: An LED lights up and the LCD displays a message that the device is currently recording but that there is insufficient memory available. The recording is halted and the current session is saved/processed unfinished.

6.6 Extreme Case 2: Tuner – Chord Played

User Input: The user plays a chord.

Conditions: The guitar is plugged into the audio line-in and the device is in “Tuner” mode.

Expected Observations: An LED lights up as the note is being processed and the LCD displays a message that the device is currently tuning. Within 2 minutes, the LED will turn off and the LCD will display a message stating that the note has not been recognized.

6.7 Extreme Case 3: Metronome On While Recording

User Input: The user presses the “Start Record” button, starts playing, then presses the “Stop Record” button.

Conditions: The guitar is plugged into the audio line-in, the device is in “Record” mode, and the metronome is currently on.

Expected Observations: An LED lights up and the LCD displays a message that the device is currently recording. Recording is successful until user stops the recording and that session is then saved. Metronome clicks in saved session is not audible.

7. Future Iterations

In future iterations of the AutoTab, features deemed as “final product feature” in this document will be implemented. In addition:

1. To increase the accuracy of the transcription results while accounting for varied styles of guitar playing, a commercial version of the product could host a training mode, either in software or on the device. This would calibrate the device by having the user play

specific notes on command, allowing the algorithms to set appropriate thresholds for note detection, account for the varying harmonic signatures of different guitars, and to correct for any potential detuning. Moreover, this would create the potential for the user to save custom gene pools, thus allowing the device to be attuned to different instruments.

2. A microphone input will be added, as well as gene pools and detection rules for additional instruments, thus expanding the possible uses and markets for the device.

3. In order to increase efficiency, price, and power usage, the end product would be built off an ASIC instead of an FPGA.

The figures below show an example of what the encapsulated model might look like in future iterations of the AutoTab. Prior to building the AutoTab off an ASIC, a handheld version of the device will be implemented through the usage of a ribbon cable running from the FPGA to the encapsulated model.

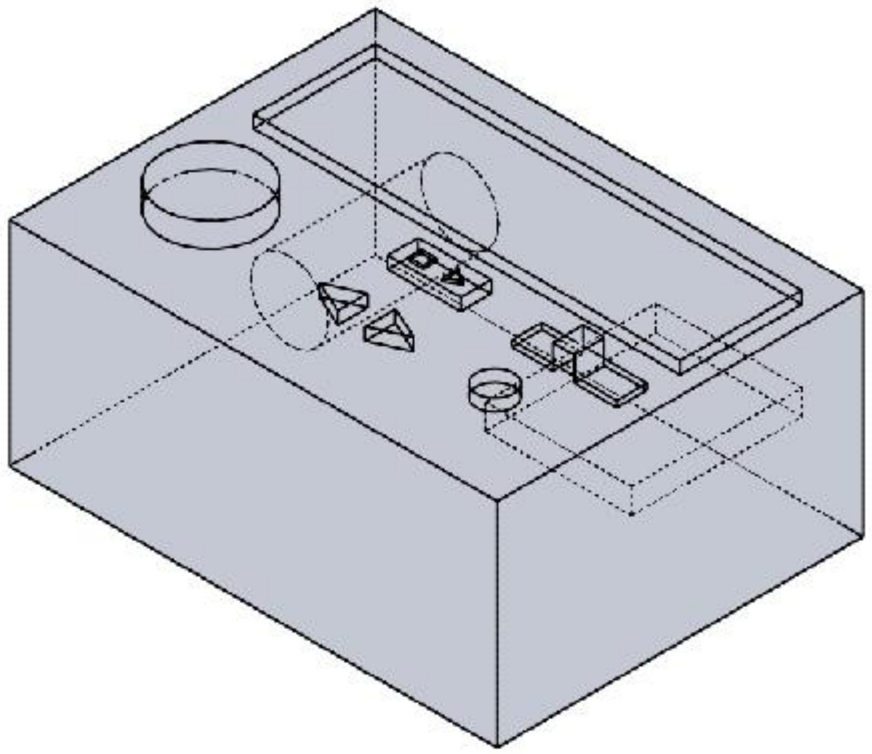

Figure 8: Isometric view of potential encapsulated model

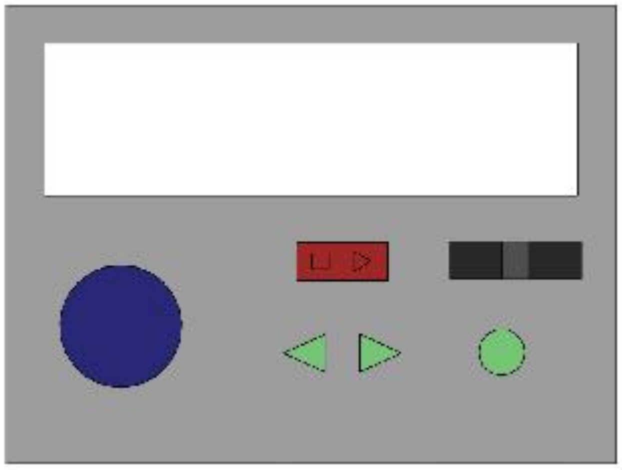

Figure 9: Front view of potential encapsulated model

8. Bill of Materials

Our bill of materials shows the best estimate of the materials we will be using in building an encapsulated product while using the FPGA board as the main processor. These components were found through the electronics supplier, DigiKey [10].

Table 2: Bill of Materials

Component	Model	Quantity	Price
FPGA Development Board	Terasic: Altera Cyclone IV	1	$260
Battery	Energizer CR-1620	1	$5
Casing	-	-	$30
Power Switch	Micro-mini slide: MMS1208	1	$1.50
Volume Dial	Stratocaster control knob: 0465-B	1	$4
Push Buttons	C&K Components: D6R10 F1 LFS	4	$5
Flat Head Machine Screws	McFeely's: FMZ-1430-M	4	$1.50
Resistors	Panasonic-ECG: ERD-S2TJ5R1V	1000	$5
Jumper Wires	3M: 923345-03-C	200	$18
Ribbon Cable	3M: 3365/06 300SF	1	$2
Insulation-displacement connector	TE Connectivity: 552301-1	2	$6
SD Card Slot	TE Connectivity: 2041021-4	1	$3

Stereo Audio Jack	CUI Inc: SJ-3523-SMT	1	$1
Copper Clad Perforated Prototype Board	Injectorall Electronics: B3427D	1	$22
Total Cost			$390

9. Conclusion

The proposed design solutions to meet the functional specification of the AutoTab have been discussed in this document and provide clear goals for the development of our proof-of-concept model. During the actual development, these design specifications will be adhered to as much as possible to meet the functional specification. Through the test plans included in the design specifications, we can ensure that all of the required functionality of the AutoTab is present.

References

1. ScribeWare Inc. (2012). *Functional Specification for the Automatic Music Transcriber.* Simon Fraser University, Canada.

2. The Joy Of Guitar. (1999). Retrieved February 27, 2012, from http://www.the-joy-of-guitar.com/guitartuner.html

3. Lu, D. (2006). *Automatic Music Transcription Using Genetic Algorithms and Electronic Synthesis.* University of Rochester, New York.

4. dos Reis, G. M. *Electronic Synthesis Using Genetic Algorithms for Automatic Music Transcription.* School of Technology and Management Polytechnic Institure of Leiria, Portugal.

5. Suits, B. (1998). *Frequencies of Musical Notes*. Retrieved March 2, 2012, from http://www.phy.mtu.edu/~suits/notefreqs.html

6. Gerhard, D. *Pitch Extraction and Fundamental Frequency - history and current techniques.* University of Regina, Department of Computer Science, Canada.

7. Kianpour, A., & Manuel, D. (2006). *Signal Processing Methods for Music Transcription.* New York, USA: Springer.

8. J. Sieger, N., & H. Tewfik, A. (1998). *Audio Coding for Representation in MIDI via Pitch Detection.* University of Minnesota, Department of Electrical and Computer Engineering, USA.

9. Bryd, D., & Crawford, T. (2002). *Problems of music information retrieval in the real world.* University of Massachusetts; King's College, Department of Computer Science; Music Department, USA; UK.

10. DigiKey Corporation. (n.d.). Retrieved March 6, 2012, from http://www.digikey.ca/?curr=CAD