SENIOR DESIGN PROJECT REPORT
EE/CpE 4097 – Spring 2017

Project title: Digital Music Transcriber

Team members: Patrick Bremehr (pmbb83@mst.edu, CpE/CS)
Andrew Donaldson (atdtk5@mst.edu, EE)
Stuart Miller (sm67c@mst.edu, CpE)

Customer(s): Patrick Bremehr, Andrew Donaldson, Stuart Miller

Advisor(s): Dr. Jonathan Kimball (Assoc. Prof, ECE)

Estimated cost: $ 108.50

Instructor: Dr. Daryl Beetner

Presented to the
Electrical & Computer Engineering Faculty of
Missouri University of Science and Technology

In partial fulfillment of the requirements for
SENIOR DESIGN PROJECT II (EE/CpE 4097)
May 2017 (SP 2017)

Table of Contents

1 Introduction

1.1 Executive Summary

Digital Music Transcriber is a portable, microcontroller-powered audio interpreter that provides real-time music visualization. Usage allows music educators to demonstrate passages of music while the Digital Music Transcriber performs real-time signal processing and audio analysis. A sheet music visualization is instantly available for both the instructor and student to review and use as a teaching tool.

1.2 Background and Problem Statement

The goal of this project is to make music education more accessible and interactive. Music education has traditionally been a one-on-one process consisting of just student and instructor in a closed-off room. This can be a very intimidating process for someone who has no prior experience with music. Our product will loosen up this relationship and bring some familiarity into the process. Allowing the instructor to demonstrate a passage or riff and then reviewing it on Digital Music Transcriber gives the student instant feedback and a visualization to reference during their personal practice sessions.

Digital Music Transcriber is especially valuable for beginning students. The project particularly targets students that are still learning to read music, guitar players that only know how to read tabs, and students who wish to transcribe an improvisation or "jam session".

1.2.1 Existing Works

Music educators have traditionally relied on clunky, inaccessible, software-only models to record and transcribe music. This presents a number of problems as each of these software programs is limited by both the computer already available to the learner, and the learning curve of the software itself.

Each of the following websites presents a more-or-less identical solution, making use of desktop computer applications, or low-powered web interfaces.

1. http://www.scorecloud.com/
2. https://www.smartmusic.com/
3. https://www.seventhstring.com/

1.2.2 Global and Societal Context and Motivation

Music Education has long been proven to enrich the minds of people of all ages. The process of learning music stimulates and enhances all areas of development and increases learning skills across all disciplines. Music education has been shown to greatly enhances students' understanding and achievement even in non-musical subjects. For example, a ten-year study performed by the Music Empowers Foundation, which tracked over 25,000 middle and high school students, showed that students in music classes receive higher scores on standardized tests than students with little to no musical involvement [1].

The Digital Music Transcriber is just one more way to help improve people's lives by making the process of music education more accessible. By bringing the music learning process into the realm of technology that most 21st century learners are already familiar with, we can promote this lifelong passion and enrich the lives of many.

2 Design and Methods

(Discuss the goals/tasks/requirements and how they were modified from your original proposal, why the change was needed, and how you came to the decision on the modification. Overall, you should be comparing the planned vs. actual project goals/tasks/requirements.)

2.1 Goals

2.1.1 Goal 1: Design a System to Identify Pitch from an Input

To interface with an audio signal and make a recording. The obtained signal must then be fed through a digital signal processing algorithm to obtain a frequency and be converted into a musical scale degree. This was achieved successfully and constituted the bulk of this semester's design work.

1) **Task 1.1: Implement hardware to interface with an input audio signal**
   Description: Design a two-stage gain amplifier/filter circuit using an electret microphone and pass the output to the microcontroller inputs. Assemble necessary hardware on a breadboard and connect to microcontroller inputs.
   Challenges: Amplifying and filtering the small electret microphone output while staying within the boundaries of hardware capabilities was difficult. Finding the proper gain level on the amplifiers took several testing sessions. Becoming familiar with the hardware was crucial in order to provide proper voltage/current levels and protect the components as well as the user.

2) **Task 1.2: Discretize the signal and store it into memory**
   Description: Save the signal in a data structure for later analysis.
   Challenges: Needed to determine an appropriate buffer size and implement interrupt logic to store obtained samples in the global buffer. The main challenges here were deciphering the interrupt architecture of the TI C2000 ecosystem and implementing it correctly.

3) **Task 1.3: Perform a fast Fourier Transform on the stored signal to obtain a frequency**
   Description: Perform a FFT on the recorded signal.
   Challenges: Choosing an appropriate FFT library was difficult, give our strict real-time requirement. Most of the processing was abstracted away by the library, but it still needed to be kept fast enough not to be outrun by the sampling buffer.

4) **Task 1.4: Convert frequency to a pitch (translate to a scale degree)**
   Description: Translate frequency to musical pitch.
   Challenges: This was a single simple line of code. We simply converted a frequency to a scale degree using a known logarithmic equation.

2.1.2 Goal 2: Design and Implement a Buffered Audio Input System

To store the input signal in a buffer and perform calculations on the buffered signal. The buffer should be small enough that the resultant pitch is calculated with minimal time delay from the production of the input audio signal. This was relatively easy to achieve, but took a large amount of time to properly optimize. The team was modifying timing values up until the very last week of the project.

1) **Task 2.1: Implement input buffer**
   Description: Store the discretized signal in a temporary buffer for processing.
   Challenges: Two buffers were implemented and swapped as each filled. The only challenges here was finding appropriate space in RAM to fit both buffers and not overrun existing data structures.

2) **Task 2.2: Perform FFT/calculation on buffered input**
   Description: Implement the calculations for FFT/pitch on the input buffer.
   Challenges: Choosing the correct FFT configuration proved difficult. Eventually, it was discovered that the program was using a phase-aligned FFT, which skewed the results. Changing this to a general unaligned FFT fixed this.

3) **Task 2.3: Reduce buffer size to allow for near realtime calculations**
   Description: Optimize buffer/calculation pipeline until the time delay is no longer noticeable.
   Challenges: The main challenge here was choosing an appropriate sampling rate and buffer size to that the sampling does not outrun the calculation. This was achieved mainly by trial-and-error.

2.1.3 Goal 3: Design and Build a System to Display Pitches

To show the resultant scale degrees in a format that is familiar to musicians. This may take the form of a transcription on a grand staff, or merely an itemized list of scale degrees as they were played.

1) **Task 3.1: Display resulting pitch on an LCD screen**
   Description: Interface the microcontroller with a LCD to output the pitches that are detected.
   Challenges: Getting the LCD screen up and running took longer than expected and a lot of man hours were required before it worked correctly. In addition, each pixel had to be drawn more or less individually, making the code to lengthy to draw simple objects.

2) **Task 3.2: Employ musical notation to display the pitch on a staff on the LCD screen**
   Description: Display a staff and employ standard musical notation to display the notes.
   Challenges: Related to challenges above, drawing something as complicated as a treble clef required tedious pixel art conversion. In addition, there was no easy way to convert a note number to the actual note and sharp/not sharp on the staff due to the complex nature of sharps and flats within an octave.

2.2 Deliverable

Digital Music Transcriber provides a portable, microcontroller-powered transcription platform that provides real-time music visualization.Trained musicians are able to utilize Digital Music Transcriber as a teaching and learning tool.

2.3 Specifications and Requirements

System functionality is verified by three main requirements: the system must be able to identify a pitch, to do so in real-time, and to convey that information to the user in an appropriate manner. As shown in Table 1 below, each requirement was met successfully with the final product.

Table 1: Requirements Matrix

ID	Title	Description	Metrics	Verification	Results
Requirement 1:	Pitch Identification	Identify pitch within an error margin	Within 2% Hz of designated pitch	PASS/FAIL	PASS
Requirement 2:	Real-time Implementation	Display identified pitch within a small time delay	Delay <= 25 ms	PASS/FAIL	PASS
Requirement 3:	User Interface	User interface is easily understood by average user	95% of user acceptance testing pass without assistance	PASS/FAIL	PASS

3 Technical Approach and Results

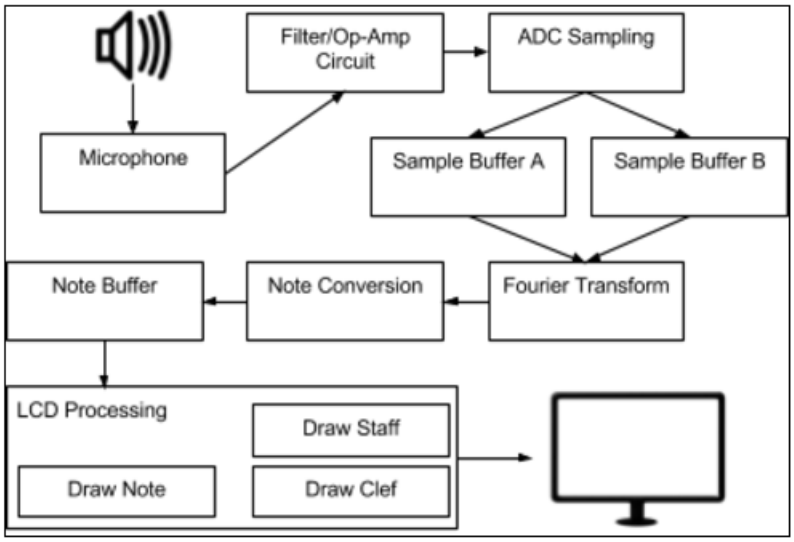
3.1 Final Design

3.1.1 System Block Diagram

The following block diagram (Figure 1) outlines the system architecture from a microphone input to an LCD display output. In brief, the system starts by taking input from a microphone and pipes is through a filter/operational amplifier circuit. The microcontroller samples the input through its onboard analog/digital converter and stores it in a buffer. The resulting buffer is passed into a fast Fourier transform algorithm, which calculates a frequency. That frequency is converted to a note and then passed to display processing to output on the LCD screen.

Further details on various stages of this diagram can be found in the following sections.

Figure 1: Project Block Diagram



3.1.2 Microcontroller Platform

For the signal processing requirements in the Digital Music Transcriber, it was necessary to choose a powerful, DSP-optimized chip. The chip must have floating point hardware, and be fast enough to be able to calculate an FFT algorithm without falling behind the sampling rate. Additionally, the microcontroller must have onboard ADC support, or be able to easily interface with an external ADC chip. The Texas Instruments launchpad series of microcontroller boards seemed the logical choice for the project's needs. It was decided to use the TMS320F28377S launchpad, featuring a Texas instruments 32-bit C28x CPU, which includes a floating point arithmetic unit. Additionally, this board is part of Texas Instruments' launchpad series, meaning it can easily accommodate an LCD boosterpack module for the display.

3.1.3 Analog Circuit Design

The following figures are schematics and oscilloscope readings from the amplifier/filter circuit used to transform the output of the electret microphone to a clean, readable sinusoid. In early designs, it was unsure what audio receiver would be used. After the electret microphone was decided on, there were still things like adequate gain levels that needed to be tested. These designs can be seen in figure 2. After several iterations were designed and tested, the schematic in Figure 2a was decided to be the final circuit design. Figure 2a is a two-stage amplifier/filter circuit with a gain of 100 that amplifies and filters the output of an electret microphone and biases it around approximately 1.6 volts. Figure 2b is an earlier iteration of figure 2a, with a gain level of 300. Figure 2c is the theoretical output of figure 2b, using a 9 volt input. Figure 2d is the actual output of the preliminary design in figure 2b; clipping of the sinusoid can be observed on both sides of the wave. Because of this, it was decided that a 3.3 volt input, as well as a gain of 100 opposed to 300, would suffice.

The raw output of the microphone is normally a very small AC voltage signal, roughly 20 mV, centered around 0 volts. Since the ADC can only read AC signals above 0 volts, the purpose of this circuit is to give the electret microphone output a DC bias so that all signals it outputs are above 0 volts. The circuit also amplifies and cleans up the signal to make it easier to read for the ADC.

Figure 2a: Final Microphone Amplifier/Filter Schematic

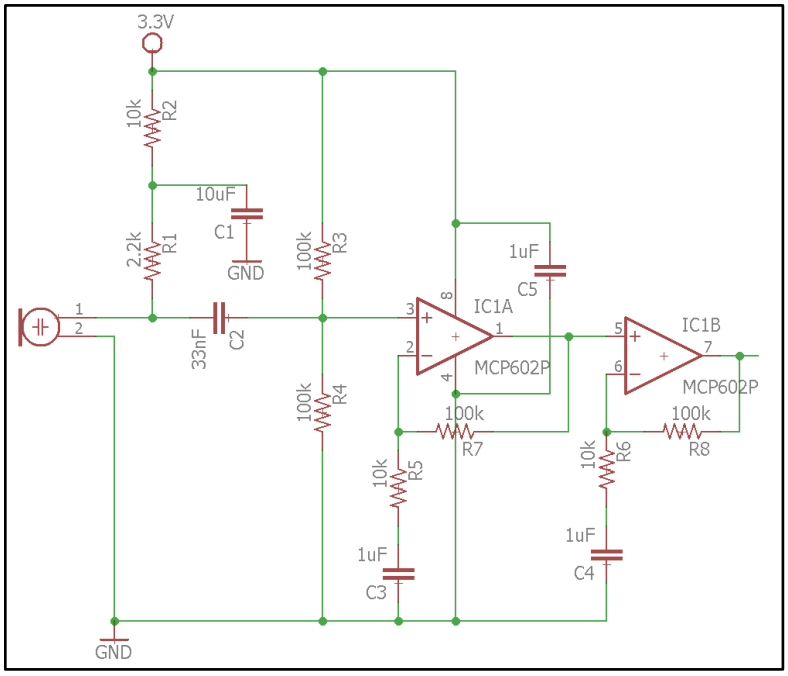


Figure 2b: Preliminary Microphone Amplifier/Filter Schematic

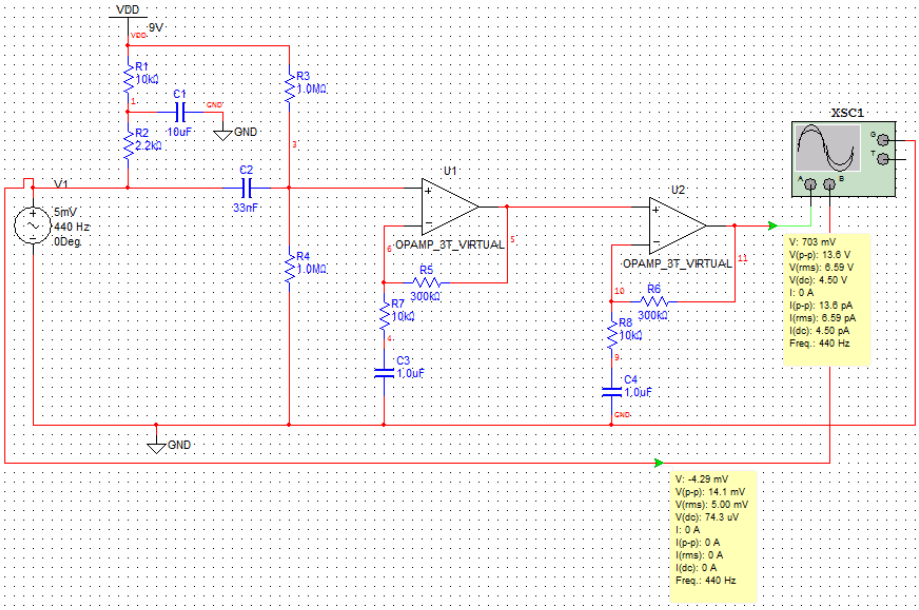


Figure 2c: Preliminary Microphone Amplifier/Filter Theoretical Output

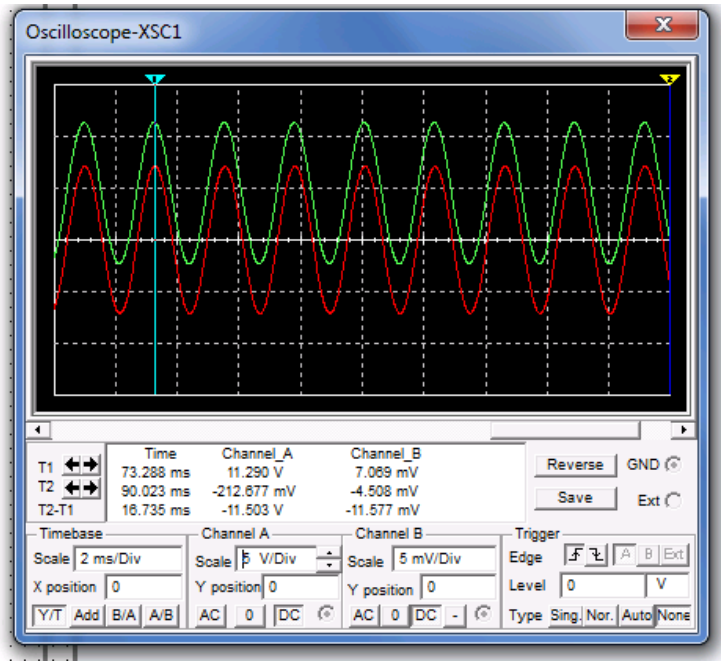


Figure 2d: Preliminary Microphone Amplifier/Filter Output (Clipping)

3.1.4 Sampling/Fourier Transform Design

The first concern with implementing the Digital Music Transcriber was choosing a appropriate sampling rate to record an input audio signal. The sampling rate needed to be such that it can record long enough to identify a low frequency note, while still sampling quickly enough to get several samples per period on a high frequency note. Sampling at 10kHz, with a buffer size of 1024 samples allows for calculating approximately 10 notes per second at an appropriate rate to identify each.

Figure 3 shows a note near the bottom of the piano keyboard (and is representative of the lowest range most standard instruments will produce). Sampled at 10kHz, just over two entire periods are recorded, which provides enough data to calculate a frequency. Alternatively, Figure 4 shows a note near the top of the piano keyboard (and is representative of the highest range most standard instruments will produce). Again, sampling at 10kHz allows for 3-4 samples per period. This is double the Nyquist rate and is sufficient to identify a frequency.

Figure 3: Sampling a 27.5Hz Note (A_0)

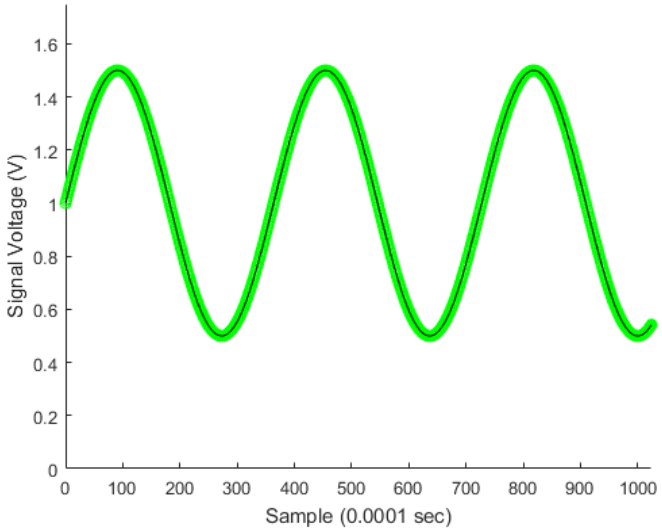
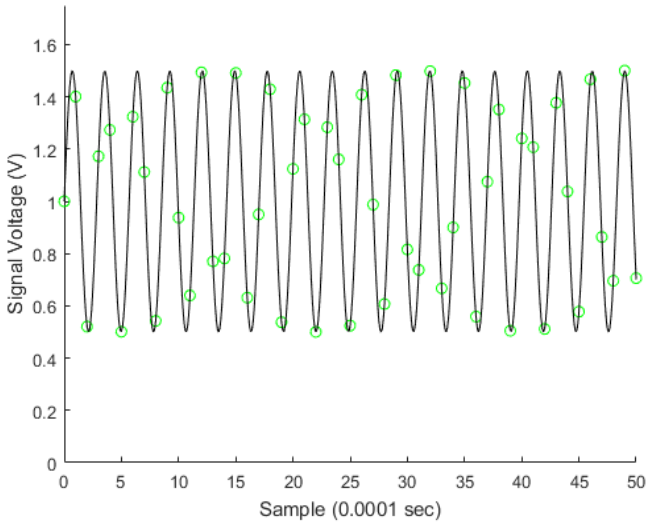


Figure 4: Sampling a 3520Hz Note (A_7)



The sampling was implemented by an interrupt service routine(ISR) on the C2000 microcontroller. The ISR was triggered by the EPWM clock, which was configured with prescale values to cycle at 10kHz. Every interrupt, a reading was taken from the analog-to-digital converter(ADC) and stored in a buffer. Upon filling the buffer, its address was swapped with that of a secondary buffer and that buffer began filling (note the separate buffers in Figure 1, the project block diagram). While each

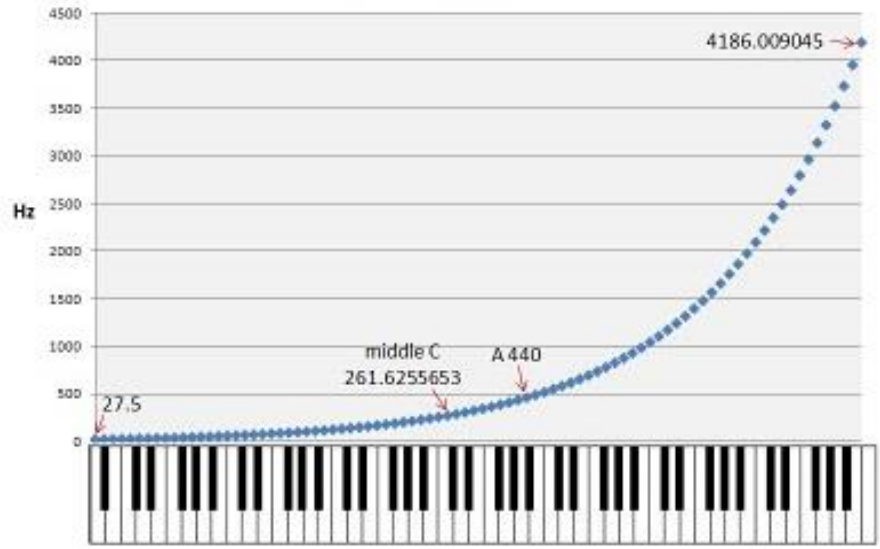
buffer is being filled, the other is passed into a fast Fourier transform algorithm which calculated its frequency.

As for the fast Fourier transform(FFT) algorithm, Texas Instruments provides a large library of ready-made, hardware-optimized functions in their ControlSuite software package (http://www.ti.com/tool/CONTROLSUITE). The Digital Music Transcriber makes use of the ControlSuite's in-place real Fast Fourier Transform function. The function returns a magnitude buffer with the found frequencies, which can then be scanned for the dominant frequency and converted to a musical scale degree. This conversion can be done by a simple logarithmic equation (Figure 5), where *n* represents the *n*th note from the bottom of a piano keyboard and *f* represents the frequency. This conversion can also be seen graphically in Figure 6.

Figure 5: Frequency to Scale Degree Conversion

$$n = 12log_2(\frac{f}{440\ Hz}) + 49$$

Figure 6: Logarithmic Frequency Conversion



3.1.5 Interface/LCD Design

There were a few options when looking for ways to display the note that it determined based on the fast fourier transform calculation. These options included an attached LCD screen, a web based application that could be viewed from a phone, or another external display. External displays proved expensive and too large for our needs, while a web based application would require the user to have an internet connection, so the logical choice was to use an attached LCD screen. Luckily, the F28377S launchpad provides a very accessible booster pack interface, allowing for many booster packs sponsored

through Texas Instruments. One such booster pack, the Kentec QVGA Display Booster Pack, boasted a color 320 by 240 pixel screen. We decided on this LCD screen for the interface.

Receiving an integer note from the output of the logarithmic function, the interface needed to determine the actual note on the given staff and whether the note was a sharp or not. In our system, we decided to only represents notes as their naturals, or their sharp counterparts. However, because of how each octave of music works, not every not has a sharp and a natural. To quickly determine where an integer note belonged on the staff and if it was a sharp or a natural, I created two lookup tables that translated the integer note to a number representing the placement on the staff, and a number representing a sharp or a natural. This allowed us to display notes with little to no additional processing by the microcontroller, which was important so that we could display notes in real time.

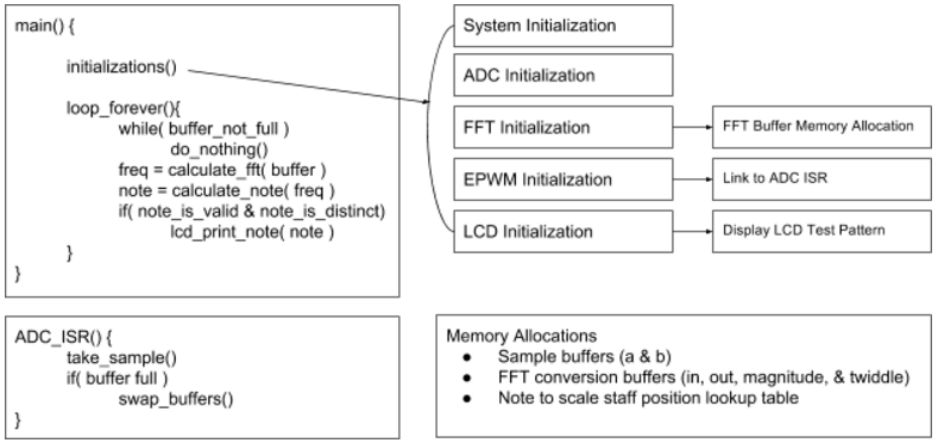
In addition to translating the integer note, the microcontroller had to decide when it was hearing background noise, or actual notes. This problem was fixed by implementing a system where a note would only be displayed if it heard three of the exact same notes in a row. Due to the quick sampling, this still allowed for near instant note population but greatly reduced background noise and the number of notes that displayed on the screen. Also in order to maximise the amount of notes on the screen, a system was implemented where if the same note was heard consistently without a pause, the screen would not keep displaying that same note over and over. This was a step toward displaying notes of different lengths, but for now the screen only shows quarter notes.

3.1.6 Code Structure

The following psuedo-code (**Figure 7**) outlines the main program flow of the Digital Music Transcriber. The main components are the initializations, the main program loop, and the ADC interrupt. this diagram is highly abstracted and each component includes several nested functions. Each initialization and calculation function includes a great deal of additional logic that is not shown here.

The main program flow consists of the eternal loop, interrupted regularly by the ADC interrupt service routine to take samples. These two module trade execution constantly and are designed in such a way that the main program will not continue until after the buffer is full, and the calculation will complete before the buffer finishes filling.

Figure 7: Code structure



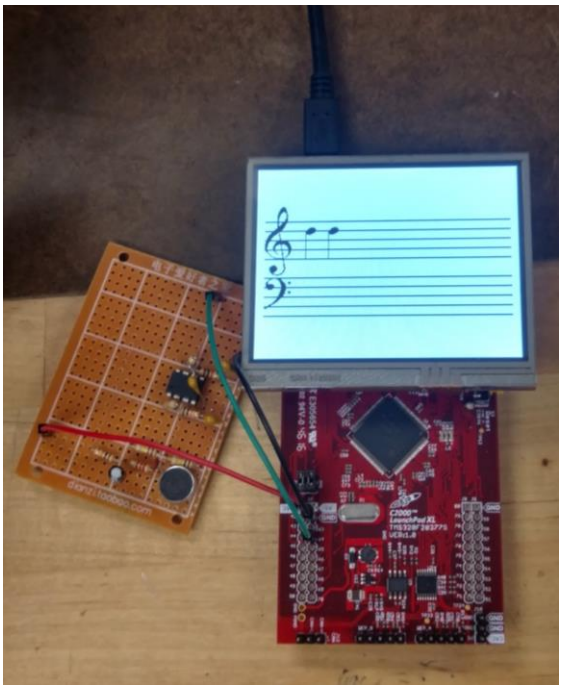
3.2 Results

The final product consists of a microcontroller board, with LCD screen attached, and a secondary analog filter circuit board. Performance-wise the final results meets all the design goals. These goals are user-defined and easily verifiable by trained musicians, as shown in Table 1 under Section 2.3. Each note is accurately identified and done so with no noticeable delay. Furthermore, the display outputs the musical sequence on a grand staff using traditional notation. Thus, it is easily understandable by anyone with a minimal musical background.

Verification was completed by testing with an electronic pitch generator. Frequencies were instantly identified and outputted correctly, thereby meeting all the initial requirements. Further testing was completed with simulated instruments, but this was more difficult to verify, as the environment is frequently too noisy and too prone to pick up harmonics.

As the main goal of the project is to user-satisfaction, there is little data to prove the verification. The actual time delay is minimal enough not to be noticed by user, and the pitch is always accurate with simulated pitches. Needless to say, there is further work to be done to achieve this verification in all states of product usage; this is to be discussed in Section 5.2.

Figure 8: Final Product



4 Management

4.1 Project Milestones

The project timeline was loosely designed to provide a rough framework for the ordering of design component development. The original plan was to have a more segregated design philosophy with less overlap; however, as the chart below shows, most of our goals overlapped significantly. This, in itself, is not an issue as the project as a whole kept on track. This is due to several components getting an early start to account for the overrun on others. For example, due to the filter circuit being more complicated than initially anticipated, its design took much longer, but work was able to continue ahead on other components, even without a completed filter circuit. Overall, although many components did get done one time, but the ones that did not did not delay the start of work on other components as each component was relatively independent.

Table 2: Project Timeline and Milestones (Gantt chart)

Gray = Planned Yellow = In Progress Green = Complete Red = Behind	January			February					March				April				May	
	1/16 - 1/20	1/23 - 1/27	1/30 - 2/3	2/6 - 2/10	2/13 - 2/17	2/20 - 2/24	2/20 - 2/24	2/27 - 3/3	3/6 - 3/10	3/13 - 3/17	3/20 - 3/24	3/27 - 3/31	4/3 - 4/7	4/10 - 4/14	4/17 - 4/21	4/24 - 4/28	5/1 - 5/5	5/8 - 5/12
Implement hardware to interface with an input audio signal																		
Discretize the signal and store it into memory.																		
Perform a fast Fourier Transform to obtain a frequency																		
Translate frequency to pitch																		
Implement input buffer																		
Perform FFT/calculation on buffered input																		
Reduce buffer size to allow for near realtime calculations																		
Display resulting pitch on an LCD																		
Employ musical notation to display the pitch on a staff																		

4.2 Encountered Challenges and Lessons Learned

4.2.1 Analog filter problems:

The first problem that arose with the analog circuit was how to receive the audio signal. This issue was the earliest obstacle that needed to be overcome in terms of the analog circuit design. We considered picking up the signal given off by electronic instruments via a ¼" amp jack cable. However, had we gone through with this design, only electronic instruments would be able to be used on the Digital Music Transcriber. After several brainstorming sessions and discussions, we decided that a universal microphone would be the best option. In order to keep the design of the device small and compact, an electret microphone was chosen as our audio signal receiver.

After the issue of audio receiving was resolved, the next hindrance in the analog circuit design was how to amplify and clean up the signal from the microphone. The electret microphone output values ranging from approximately 5 to 25 mV; this signal was too small and distorted to be properly converted into a digital signal. The final difficulty in this stage of the project was choosing a proper gain level. Using only a single op-amp did not provide enough gain for the ADC to read the signal. Thus, a 2-stage amplifier design was chosen. Because of this, we were unable to allow for a simple way for the

end-user to alter the gain, and a static gain value had to be chosen. After many hours of research, advisor meetings, simulations, lab testing, and team discussion, a design was chosen. This design, as well as more information, can be seen in Figure 2. The design process of the filter/amplifier circuit taught the group a great deal about how filters and amplifiers are used in electronics.

4.2.2 Real Time Sampling Problem

A major part of the embedded code design was deciding upon and implementing real-time sampling. While much of this was discussed in Section 3.1.4 and shown in Figure 3 and 4, it is worth noting the lessons learned here.

Part of the problem was that the C2000 architecture in and of itself is nearly indecipherable; Texas Instruments provides little user-friendly or "getting started" documentation. Instead, they provide a vast amount of example projects in their program market as ControlSuite. When designing the real-time components, I relied heavily on example projects and the final results is a highly modified version of about four of the example projects merged together; one for the Fourier transform, one for the ADC, one for the EPWM clock, and a general startup routine for this particular microcontroller. (This doesn't even account for the part of the codebase dedicated to driving the LCD.) After working on this, I now feel much more comfortable working in TI's environment, but I was quite overwhelmed at first.

After deciding on a sampling frequency as noted in Section 3.1.4, prescaler, and counter values had to be determined. Again, this was mostly determined by trial and error by modifying the pre established values in TI's example programs. After deciding on values, it became apparent that the main clock/16 was the count frequency. This appears to be the default, although I could not find it noted in the documentation.

In the end, most of the development was done by trial and error, due to my own unfamiliarity with the platform. Were I to repeat this, I would have a much firmer grasp and be able to approach the problem with a pre-planned strategy.

4.2.3 LCD Screen Setup Problem

The biggest challenge on the interface end of the project was getting the LCD screen communication with the microcontroller setup and determining the correct setup bits for the LCD screen's numerous configurations. A lack of prior knowledge of SPI communication and working with microcontrollers as a whole added hours of time to determine the correct configurations, requiring reading through pages of documentation. Adding to this problem was the sometimes inaccuracy of the documentation itself. The prime example being in the LCD driver documentation, a bit was described to work in the opposite of how it actually worked. Through trial and error I eventually determined the error and resolved the configuration problem.

4.3 Team Ethics Discussion

Team ethics ran quite smoothly this semester. The project was very well segmented into individual components; analog circuit design, ADC/FFT programming, and interface/LCD programming. Each team member completed their part diligently.

Occasionally, deadlines were missed or meetings were skipped by accident, but is to be

expected in a semester-long project. This is especially true as all team members are seniors who are active in other projects and organizations as well. The few times this happened, each team member was quick to rectify the mistake and didn't let it fall in the way of long-term progress.

4.4 Budget

The team adhered to the prescribed budget, as planned. There were no additional or unforeseen costs. The main components (the launchpad, LCD screen, and analog components) were all planned and the reset of the materials and equipment were supplied from the lab or provided from each team member's personal possessions (i.e. the breadboards). Overall, this is very close to the budget initially planned. The teams left freedom for flexible costs of components for the filter circuit, and the other costs were all accounted for. There were no unexpected costs that were not initially planned.

The total expenditure for this semester was $108.50; however, this takes into account that we purchased two LaunchPad board for development and extra quantities of circuit components. The actual cost of a single final product would be closer to $60.

Table 3: Project Budget Example

	Items	Unit cost	Quantity	Total cost	Comments
Parts	F28377S LaunchPad	$29.99	2	$58.98	C2000-series microcontroller launchpad
	LCD Screen	$24.99	1	$24.99	Kentec QVGA Display Boosterpack
	33nF Capacitor	$0.778	5	$3.89	Coupling capacitor for AC and DC signals
	1uF Capacitor	$0.598	5	$2.99	monolithic capacitors for decoupling AC and DC signals.
	10uF Capacitor	$0.199	10	$1.99	Used in VDD voltage divider
	2.2KOhm Resistor	$0.152	10	$1.52	Used in VDD voltage divider
	10KOhm Resistor	$0.152	10	$1.52	Used for setting gain values
	100KOhm Resistor	$0.152	10	$1.52	Used for setting gain values
	Perfboard	$0.938	5	$4.69	Used to decrease the size of the final product.
	Microphone	$0.84	5	$4.20	Omni-directional PCB mount 9.7mm electret microphone
	Op-Amp	$1.11	2	$2.22	MCP 6022 8-pin Dual op amp 10MHz G/BW Rail-Rail
Equipment	Breadboard	$0.00	2	$0.00	personally owned, shared with other lab projects
	Oscilloscope	$ 0.00	~	$ 0.00	available in Senior Design lab
Software	TI Code Composer Suite	$ 0.00	~	$ 0.00	Free from Texas Instruments
			Total cost	**$ 108.50**	

4.5 Funding Source

Due to its low cost, and the team's self-interest in the subject matter, the project was entirely self-funded by the team.

4.6 Human Safety Assessment

The risk factor associated with the product is low. The worst possible outcome is merely that the end user is dissatisfied with the product. One consideration is to make sure the hardware is properly grounded and does not provide any electrical hazards. Furthermore, the device should not provide any harmful feedback to any instruments that are connected to it. Some care must also be taken to prevent the device from producing any undue heat dissipation, although the risk of this happening is very slim. In building the device, care must be taken while using any lead based soldering as well. This can easily be avoided however by following simple lab safety procedures. Overall, the device is build correctly and within safety regulations, and no harm should come to the producers of the device or the users of the device.

4.7 Member Credentials and Responsibilities

4.7.1 Teamwork

The team includes a diverse spectrum of experience in both software and hardware. The Digital Music Transcriber requires expertise in low-level circuit design, and high-level software interface design. The team is also uniquely qualified to complete this project as they are all trained musicians and have a distinct passion for music.

4.7.2 Stuart Miller (sm67c@mst.edu, CpE major):

Member profile: Stuart has had experience with software engineering on embedded systems through classwork and internships in industry. He will be working on most of the embedded software and the signal processing algorithms implemented on the microcontroller.

4.7.3 Andrew Donaldson (atdtk5@mst.edu, EE major):

Member profile: Andrew has extensive experience with breadboarding, electronics, and devices and is well rounded in all facets of Electrical Engineering, He will primarily be working on the design and construction of the circuit board.

4.7.4 Patrick Bremehr (pmbb83@mst.edu, CpE/CS major):

Member profile: Patrick has had a multitude of experience designing web based applications and will mainly be responsible for creating the interface of the device in order to show the user the transcribed notes.

5 Conclusions and Future Work

5.1 Conclusions and Lessons Learned

The primary takeaway from this project is a familiarity with Texas Instruments' development environment. None of the team members had any experience developing for TI processors before and learned a significant amount throughout the project. Were the team to approach this problem again, we would undoubtedly be able to start with a much more pre-planned strategy, having a better knowledge of the peculiarities of TI's development environment. Furthermore, the team would be able to spend far less time familiarizing themselves with the platform.

In addition the the aforementioned technical knowledge, the team gained valuable insight regarding managing a long-term project. The foresight necessary to plan out the next steps in the project and to prepare weekly summaries was a valuable experience for the team. Having to be cognizant of such duties forced the team to manage themselves better as well. All in all, this experience proved beneficial and highly relevant to those that we will soon be taking on as we graduate and transition to careers in industry.

5.2 Suggested Improvements

Our current progress, although while noteworthy, leaves much room for improvement. While this project serves as a good proof-of-concept, it leaves a lot of room for future development. The first major addition would be the ability to detect harmonics in input frequencies; this added the ability to detect chords, as the might be played on a piano or guitar. While it would be simple enough to detect secondary and tertiary peaks on the FFT output, fine-tuning the algorithm would take a large amount of testing with real instruments.

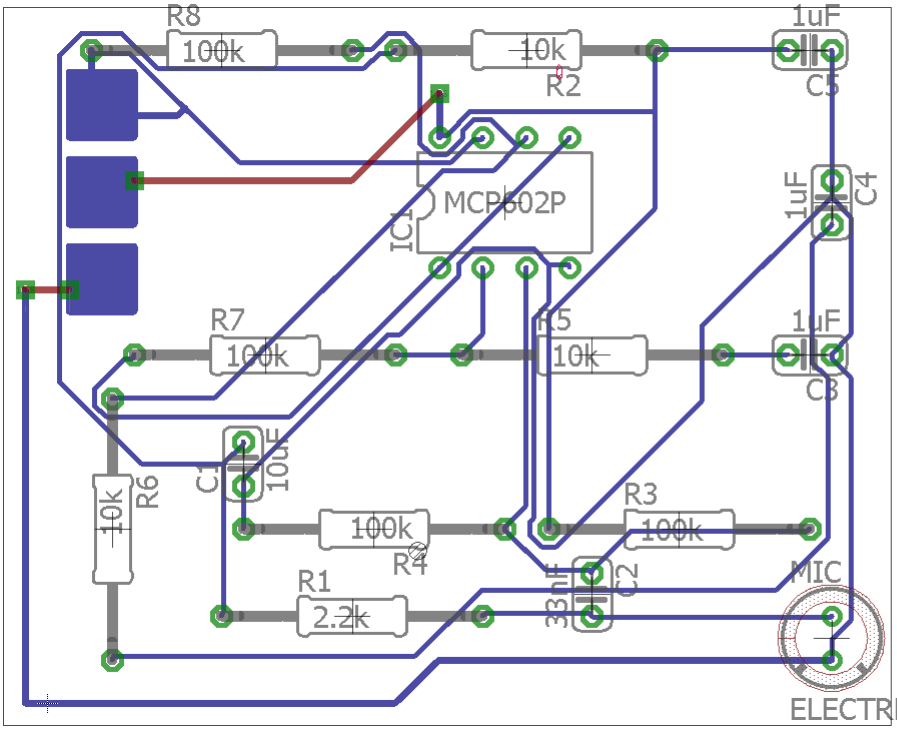
Additionally, we would have liked to have time to better establish note tempo. The ability to distinguish between quarter notes and half notes, for example, and to set a tempo would greatly increase the quality of the transcriber output. It may even be possible to implement to detect a tempo from the frequency of observed notes. This would be a considerable addition and take quite some time to get right, as it involves adapting on-the-go, and a a great deal more processing.

6 References

1. A. Kalivretenos, "The Importance of Music Education," in TheHumanist.com, TheHumanist.com, 2015. [Online]. Available: https://thehumanist.com/features/articles/the-importance-of-music-education. Accessed: Nov. 28, 2016.

7 Appendices

7.1 PCB Diagram



7.2 Source Code

```
/*
 * main.c
 *
 */

#include "F28x_Project.h"
#include "LCD_Functions.h"
#include "adc_intf.h"
#include "main.h"
#include "math.h"
#include "float.h"

#define RFFT_STAGES     ( 10 )
#define RFFT_SIZE       ( 1 << RFFT_STAGES ) // 1024
#define PI              ( 3.14159 )
#define RESULTS_BUF_SZ  ( RFFT_SIZE )

RFFT_F32_STRUCT rfft;
char printstr[10];
float32 buf_a[ RESULTS_BUF_SZ ];
float32 buf_b[ RESULTS_BUF_SZ ];
volatile Uint16 bufferFull               = 0;
Uint16 buf_idx                  = 0;
float32* samp_buf                = buf_a;
float32* conv_buf                = buf_b;
float freq, freq2;

#pragma DATA_SECTION( RFFTin1Buff, "RFFTdata1" );   // Buffer alignment for the input array,
float32 RFFTin1Buff[ RFFT_SIZE ];                   // RFFT_f32u(optional), RFFT_f32(required)
                                                    // Output of FFT overwrites input if
                                                    // RFFT_STAGES is ODD
#pragma DATA_SECTION( RFFToutBuff, "RFFTdata2" );
float32 RFFToutBuff[RFFT_SIZE ];                    // Output of FFT here if RFFT_STAGES is EVEN

#pragma DATA_SECTION( RFFTmagBuff, "RFFTdata3" );
float32 RFFTmagBuff[ RFFT_SIZE / 2 + 1 ];           // Additional Buffer used in Magnitude calc

#pragma DATA_SECTION( RFFTF32Coef, "RFFTdata4" );
float32 RFFTF32Coef[ RFFT_SIZE ];

extern Uint16 LOOKUP_NOTE_NUM[];
extern Uint16 LOOKUP_SHARP[];

void main(void)
{

   InitSysCtrl();
   InitGpio();
   Init_LCD_Pins();
   DINT;
   InitPieCtrl();

   IER = 0x0000;
   IFR = 0x0000;

   InitPieVectTable();
   LCD_Init();

   Init_StartUp();
```

```
    init_fft();

    // Map ISR function
    EALLOW;
    PieVectTable.ADCA1_INT = &adca1_isr; //function for ADCA interrupt 1
    EDIS;

    // Init ADC Modules
    ConfigureADC();
    ConfigureEPWM();
    SetupADCEpwm(2);

    // Enable global Interrupts and higher priority real-time debug events:
    IER |= 0x0001; //Enable group 1 interrupts
    EINT;  // Enable Global interrupt INTM
    ERTM;  // Enable Global realtime interrupt DBGM

    // enable PIE interrupt
    PieCtrlRegs.PIEIER1.bit.INTx1 = 1;

    // sync ePWM
    EALLOW;
    CpuSysRegs.PCLKCR0.bit.TBCLKSYNC = 1;

    // Start ePWM
    EPwm1Regs.ETSEL.bit.SOCAEN = 1;  // enable SOCA
    EPwm1Regs.TBCTL.bit.CTRMODE = 0; // unfreeze, and enter up count mode

    freq = 0;
    freq2 = -1;

    Uint16 noteBuffer[9];
    Uint16 sharpBuffer[9];
    Uint16 noteIter = 0;

    Uint16 listenBuffer[] = {100, 100, 100};
    Uint16 listensBuffer[] = {2, 2, 2};

    for(;;){
        // Calculate frequency
        freq = calc_fft();

        // Convert frequency to scale degree
        // https://en.wikipedia.org/wiki/Piano_key_frequencies
        // Due to the microphone's problems with low frequencies, do not display very low frequencies
        Uint16 note;
        if (freq > 100.0){
            note = round( 12 * log2( freq / 440 ) + 49 );
        }
        else {
            note = 500;
        }
        // Convert note to LCD offset
        Uint16 noteOffset = note - 15;

        if  (listenBuffer[2]  ==  100  &&  listenBuffer[2]  ==  listenBuffer[1]  &&  listenBuffer[2]  ==
listenBuffer[0]){
            if (LOOKUP_NOTE_NUM[noteOffset] == noteBuffer[(noteIter == 0 ? 9 : noteIter - 1)] &&
                LOOKUP_SHARP[noteOffset] == sharpBuffer[(noteIter == 0 ? 9 : noteIter - 1)]){
                // Wait for ADC ISR to fill buffer
                while( bufferFull != 1 ){;}

                // Reset status for next reading
                bufferFull = 0;
                freq2 = freq;
                continue;
```

```
            }
        }

        // If screen is full, clear the screen
        if (noteIter == 9){
            noteIter = 0;
            for(;noteIter < 9; noteIter++){
                LCD_DrawNote(noteIter, noteBuffer[noteIter], sharpBuffer[noteIter], 0xFFFF);
            }
            noteIter = 0;
        }

        // Update the listen buffers
        listenBuffer[2] = listenBuffer[1];
        listenBuffer[1] = listenBuffer[0];
        listenBuffer[0] = LOOKUP_NOTE_NUM[noteOffset];
        listensBuffer[2] = listensBuffer[1];
        listensBuffer[1] = listensBuffer[0];
        listensBuffer[0] = LOOKUP_SHARP[noteOffset];

        // If the listen buffer now agrees, display the note
        if (listenBuffer[2] == listenBuffer[1] && listenBuffer[2] == listenBuffer[0] &&
            listensBuffer[2] == listensBuffer[1] && listensBuffer[2] == listensBuffer[0]){
            if (noteOffset >= 0 && noteOffset < 53){
                LCD_DrawNote(noteIter, LOOKUP_NOTE_NUM[noteOffset], LOOKUP_SHARP[noteOffset], 0x0000);
                noteBuffer[noteIter] = LOOKUP_NOTE_NUM[noteOffset];
                sharpBuffer[noteIter] = LOOKUP_SHARP[noteOffset];
                noteIter++;
            }

            // Clear the listen buffer to avoid repeat notes too quickly
            listenBuffer[2] = 100;
            listenBuffer[1] = 100;
            listenBuffer[0] = 100;
            listensBuffer[2] = 2;
            listensBuffer[1] = 2;
            listensBuffer[0] = 2;
        }

        // Wait for ADC ISR to fill buffer
        while( bufferFull != 1 ){;}

        // Reset status for next reading
        bufferFull = 0;
        freq2 = freq;

        }
    }

void init_fft()
    {
    // Declarations
    Uint16  i;

    // Clear input buffers:
    for( i = 0; i < RFFT_SIZE; i++ )
        {
        RFFTin1Buff[i] = 0.0f;
        }

    rfft.FFTSize   = RFFT_SIZE;
    rfft.FFTStages = RFFT_STAGES;
    rfft.InBuf     = &RFFTin1Buff[ 0 ];  // Input buffer
    rfft.OutBuf    = &RFFToutBuff[ 0 ];  // Output buffer
    rfft.CosSinBuf = &RFFTF32Coef[ 0 ];  // Twiddle factor buffer
    rfft.MagBuf    = &RFFTmagBuff[ 0 ];  // Magnitude buffer
```

```
    // Calculate twiddle factor
    RFFT_f32_sincostable( &rfft );
    }

/*********************************************************
* Calculate FFT
*********************************************************/
Uint16 calc_fft()
    {
    // Declarations
    Uint16  i;
    Uint32  fo;
    Uint16 zero_cnt = 0;

    // Clean up buffers
    for( i = 0; i < RFFT_SIZE; i++ )
        {
        RFFToutBuff[ i ] = 0;
        RFFTin1Buff[ i ] = conv_buf[ i ];
        if( conv_buf[ i ] == 0)
            {
            zero_cnt++;
            }
        }

    // If we receive more than 3/4 zeros, its probably just no signal
    if(zero_cnt > ( 3 * RFFT_SIZE / 4 ) )
        {
        return 0;
        }

    // Clean up magnitude buffer
    for( i = 0; i < (RFFT_SIZE / 2 + 1); i++ )
        {
        RFFTmagBuff[i] = 0;
        }

    // Calculate real FFT
    rfft.InBuf = conv_buf;
    RFFT_f32u( &rfft );

    // Calculate magnitude
    RFFT_f32_mag( &rfft );

    // Find peak
    Uint16 max = 1;
    for( i = max; i < (RFFT_SIZE / 2 + 1); i++ )
        {
        if( RFFTmagBuff[ i ] > RFFTmagBuff[ max ] )
            {
            max = i;
            }
        }

    // Get output frequency
    fo = (Uint32)SAMP_FREQ * max / RFFT_SIZE * 2;

    return fo;
    }


/*********************************************************
* ADC1 Interrupt Service Routine
*********************************************************/
interrupt void adca1_isr(void)
```

```
    {

    // Get input from ADC
    samp_buf[ buf_idx ] = ( float32 )AdcaResultRegs.ADCRESULT0;

    // Did we fill the buffer yet?
    buf_idx++;
    if( buf_idx >= RESULTS_BUF_SZ )
        {
        buf_idx     = 0;
        bufferFull  = 1;
        // Swap buffers for conversion
        if( samp_buf == buf_a )
            {
            samp_buf = buf_b;
            conv_buf = buf_a;
            }
        else
            {
            samp_buf = buf_a;
            conv_buf = buf_b;
            }
        // Toggle GPIO pin to verify interrupt frequency
        #if _DEBUG
        toggle++;
        GPIO_writePin( 92, ( toggle % 2 == 0 ) );
        #endif
        }

    // Clear interrupt flag
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;
    PieCtrlRegs.PIEACK.all = 0x0001;
    }



/*
 * LCD_Functions.c
 *
 */

#include "F28x_Project.h"
#include "LCD_Functions.h"

Uint16                                          LOOKUP_NOTE_NUM[]                                       =
{0,1,1,2,2,3,4,4,5,5,6,6,7,8,8,9,9,10,11,11,12,12,13,13,14,15,15,16,16,17,18,18,19,19,20,20,21,22,22,23,23
,24,25,25,26,26,27,27,28,29,29,30,30};
Uint16 LOOKUP_SHARP[] =    {0,0,1,0,1,0,0,1,0,1,0,1,0,0,1,0,1,0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1,
0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1};

void LCD_Init(void) {
    Uint32 ulCount;

    LCD_selectLCD();

    //while(SpiaRegs.SPIFFTX.bit.TXFFST != 0) { }

    // Enter sleep mode (if we are not already there).
    //
    LCD_writeCommand(SSD2119_SLEEP_MODE_1_REG);
    LCD_writeData(0x0001);

    //
    // Set initial power parameters.
    //
    LCD_writeCommand(SSD2119_PWR_CTRL_5_REG);
```

```
LCD_writeData(0x00B2);
LCD_writeCommand(SSD2119_VCOM_OTP_1_REG);
LCD_writeData(0x0006);

//
// Start the oscillator.
//
LCD_writeCommand(SSD2119_OSC_START_REG);
LCD_writeData(0x0001);

//
// Set pixel format and basic display orientation (scanning direction).
//
LCD_writeCommand(SSD2119_OUTPUT_CTRL_REG);
//LCD_writeData(0x30EF);
//LCD_writeData(0x62EF);
LCD_writeData(0x72EF);
LCD_writeCommand(SSD2119_LCD_DRIVE_AC_CTRL_REG);
LCD_writeData(0x0600);

LCD_writeCommand(SSD2119_DISPLAY_CTRL_REG);
LCD_writeData(0x0023);

//
// Exit sleep mode.
//
LCD_writeCommand(SSD2119_SLEEP_MODE_1_REG);
LCD_writeData(0x0000);

//
// Delay 30mS
//
LCD_custom_delay(30);

//
// Enable the display.
//
LCD_writeCommand(SSD2119_DISPLAY_CTRL_REG);
LCD_writeData(0x0033);

//
// Configure pixel color format and MCU interface parameters.
//
LCD_writeCommand(SSD2119_ENTRY_MODE_REG);
LCD_writeData(ENTRY_MODE_DEFAULT);

//
// Set analog parameters.
//
//LCD_writeCommand(SSD2119_SLEEP_MODE_2_REG);
//LCD_writeData(0x0999);
LCD_writeCommand(SSD2119_ANALOG_SET_REG);
LCD_writeData(0x3800);

//
// Set VCIX2 voltage to 6.1V.
//
LCD_writeCommand(SSD2119_PWR_CTRL_2_REG);
LCD_writeData(0x0005);

//
// Configure gamma correction.
//
LCD_writeCommand(SSD2119_GAMMA_CTRL_1_REG);
LCD_writeData(0x0000);
LCD_writeCommand(SSD2119_GAMMA_CTRL_2_REG);
```

```
    LCD_writeData(0x0303);
    LCD_writeCommand(SSD2119_GAMMA_CTRL_3_REG);
    LCD_writeData(0x0407);
    LCD_writeCommand(SSD2119_GAMMA_CTRL_4_REG);
    LCD_writeData(0x0301);
    LCD_writeCommand(SSD2119_GAMMA_CTRL_5_REG);
    LCD_writeData(0x0301);
    LCD_writeCommand(SSD2119_GAMMA_CTRL_6_REG);
    LCD_writeData(0x0403);
    LCD_writeCommand(SSD2119_GAMMA_CTRL_7_REG);
    LCD_writeData(0x0707);
    LCD_writeCommand(SSD2119_GAMMA_CTRL_8_REG);
    LCD_writeData(0x0400);
    LCD_writeCommand(SSD2119_GAMMA_CTRL_9_REG);
    LCD_writeData(0x0a00);
    LCD_writeCommand(SSD2119_GAMMA_CTRL_10_REG);
    LCD_writeData(0x1000);

    //
    // Configure Vlcd63 and VCOM1.
    //
    LCD_writeCommand(SSD2119_PWR_CTRL_3_REG);
    LCD_writeData(0x000A);
    LCD_writeCommand(SSD2119_PWR_CTRL_4_REG);
    LCD_writeData(0x2E00);

    //
    // Set the display size and ensure that the GRAM window is set to allow
    // access to the full display buffer.
    //
    LCD_writeCommand(SSD2119_V_RAM_POS_REG);
    LCD_writeData((uint16_t)(LCD_VERTICAL_MAX - 1) << 8);
    LCD_writeCommand(SSD2119_H_RAM_START_REG);
    LCD_writeData(0x0000);
    LCD_writeCommand(SSD2119_H_RAM_END_REG);
    LCD_writeData(LCD_HORIZONTAL_MAX - 1);
    LCD_writeCommand(SSD2119_X_RAM_ADDR_REG);
    LCD_writeData(0x00);
    LCD_writeCommand(SSD2119_Y_RAM_ADDR_REG);
    LCD_writeData(0x00);


    //
    // Clear the contents of the display buffer.
    //
    LCD_writeCommand(SSD2119_RAM_DATA_REG);
    for(ulCount = 0; ulCount < 76800; ulCount++)
    {
        LCD_writeData(0xFFFF); //White
        //LCD_writeData(0x0000);    // Black
        //LCD_writeData(0xF800);    // Red
        //LCD_writeData(0x07E0);    // Green
        //LCD_writeData(0x001F);    // Blue
    }

    //
    // Deselect the LCD for SPI comunication.
    //
    LCD_deselectLCD();
}

void LCD_writeCommand(uint16_t command) {

    uint16_t dummy_receive;

    //Wait for SPI to clear
```

```
    while(SpiaRegs.SPIFFTX.bit.TXFFST != 0) { }

    //Set LCD_SDC signal low, indicating commands
    GPIO_WritePin(LCD_SDC, 0);

    //Transmit Command
    spi_xmit(command & 0xFF);

    //Wait for RX buffer to fill with something
    while(SpiaRegs.SPIFFRX.bit.RXFFST == 0) {}

    //Dummy Read Receive Buffer
    dummy_receive = SpiaRegs.SPIRXBUF;

    while(SpiaRegs.SPIFFTX.bit.TXFFST != 0) { }

    GPIO_WritePin(LCD_SDC, 1);
}

void LCD_writeData(uint16_t data) {
    uint16_t dummy_receive;

    //Wait for SPI to clear
    while(SpiaRegs.SPIFFTX.bit.TXFFST != 0) { }

    //High byte
    spi_xmit(data >> 8);

    //Wait for RX buffer to fill with something
    while(SpiaRegs.SPIFFRX.bit.RXFFST == 0) {}

    //Dummy Read Receive Buffer
    dummy_receive = SpiaRegs.SPIRXBUF;

    // Wait for SPI to clear
    while(SpiaRegs.SPIFFTX.bit.TXFFST != 0) { }

    //Low Byte
    spi_xmit(data & 0xff);

    //Wait for RX buffer to fill with something
    while(SpiaRegs.SPIFFRX.bit.RXFFST == 0) {}

    //Dummy Read Receive Buffer
    dummy_receive = SpiaRegs.SPIRXBUF;
}

void LCD_selectLCD() {
    while(SpiaRegs.SPIFFTX.bit.TXFFST != 0) { }
    GPIO_WritePin(LCD_SCS, 0);
}

void LCD_deselectLCD() {
    while(SpiaRegs.SPIFFTX.bit.TXFFST != 0) { }
    GPIO_WritePin(LCD_SCS, 1);
}

void LCD_DrawPixel(int16_t X, int16_t Y, uint16_t Value) {
        LCD_selectLCD();
        LCD_writeCommand(SSD2119_X_RAM_ADDR_REG);
        LCD_writeData(MAPPED_X(X, Y));
        LCD_writeCommand(SSD2119_Y_RAM_ADDR_REG);
        LCD_writeData(MAPPED_Y(X, Y));
        LCD_writeCommand(SSD2119_RAM_DATA_REG);
        LCD_writeData(Value);
        LCD_deselectLCD();
```

```
}

void LCD_DrawHorizLine(uint16_t x0, uint16_t x1, uint16_t y, uint16_t Value){
    uint16_t xi = x0;
    LCD_selectLCD();
    LCD_writeCommand(SSD2119_ENTRY_MODE_REG);
    LCD_writeData(ENTRY_MODE_HORIZ);
    LCD_writeCommand(SSD2119_X_RAM_ADDR_REG);
    LCD_writeData(MAPPED_X(xi, y));
    LCD_writeCommand(SSD2119_Y_RAM_ADDR_REG);
    LCD_writeData(MAPPED_Y(xi, y));
    LCD_writeCommand(SSD2119_RAM_DATA_REG);
    for(; xi <= x1; xi++){
        LCD_writeData(Value);
    }
    LCD_deselectLCD();
}

void LCD_DrawVertLine(uint16_t x, uint16_t y0, uint16_t y1, uint16_t Value){
    uint16_t yi = y0;
    LCD_selectLCD();
    LCD_writeCommand(SSD2119_ENTRY_MODE_REG);
    LCD_writeData(ENTRY_MODE_VERT);
    LCD_writeCommand(SSD2119_X_RAM_ADDR_REG);
    LCD_writeData(MAPPED_X(x, yi));
    LCD_writeCommand(SSD2119_Y_RAM_ADDR_REG);
    LCD_writeData(MAPPED_Y(x, yi));
    LCD_writeCommand(SSD2119_RAM_DATA_REG);
    for(; yi <= y1; yi++){
        LCD_writeData(Value);
    }
    LCD_deselectLCD();
}

void LCD_DrawRect(uint16_t x0, uint16_t x1, uint16_t y0, uint16_t y1, uint16_t Value) {
    uint16_t yi = y0;
    for(; yi <= y1; yi++){
        LCD_DrawHorizLine(x0, x1, yi, Value);
    }
}

void LCD_DrawStaff(){
    uint16_t x_start, x_end, y_start, yin, i;
    x_start = 10;
    x_end = 310;
    y_start = 51;

    i = 0;
    for(; i < 5; i++){
        yin = y_start + i * 12;
        LCD_DrawHorizLine(x_start, x_end, yin, 0x0000);
        LCD_DrawHorizLine(x_start, x_end, yin+1, 0x0000);
    }

    i = 0;
    y_start = 123;
    for(; i < 5; i++){
        yin = y_start + i * 12;
        LCD_DrawHorizLine(x_start, x_end, yin, 0x0000);
        LCD_DrawHorizLine(x_start, x_end, yin+1, 0x0000);
    }
    LCD_DrawTrebleClef();
    LCD_DrawBassClef();
}

void LCD_DrawNote(uint16_t xint, uint16_t yint, uint16_t sharp, uint16_t Value){
```

```
    //Ten Pixels tall, 12 wide
    uint16_t xpos, ypos;
    xpos = 60 + xint * 28;

    //ypos
    //0 = Bottom B2
    //30 = Top D
    ypos = 17 + 6 * (yint);

    //Determine additional lines to draw
    if (yint < 4) {
        LCD_DrawHorizLine(xpos-2, xpos+14, 39, Value);
        LCD_DrawHorizLine(xpos-2, xpos+14, 40, Value);
        if (yint < 2) {
            LCD_DrawHorizLine(xpos-2, xpos+14, 27, Value);
            LCD_DrawHorizLine(xpos-2, xpos+14, 28, Value);
        }
    }

    if (yint == 15){
        LCD_DrawHorizLine(xpos-2, xpos+14, 111, Value);
        LCD_DrawHorizLine(xpos-2, xpos+14, 112, Value);
    }

    if (yint > 26){
        LCD_DrawHorizLine(xpos-2, xpos+14, 183, Value);
        LCD_DrawHorizLine(xpos-2, xpos+14, 184, Value);
        if (yint > 28){
            LCD_DrawHorizLine(xpos-2, xpos+14, 195, Value);
            LCD_DrawHorizLine(xpos-2, xpos+14, 196, Value);
        }
    }


    //Draw Actual Note
    LCD_DrawHorizLine(xpos+2, xpos+6, ypos, Value);
    LCD_DrawHorizLine(xpos+1, xpos+8, ypos+1, Value);
    LCD_DrawHorizLine(xpos, xpos+10, ypos+2, Value);
    LCD_DrawHorizLine(xpos, xpos+11, ypos+3, Value);
    LCD_DrawHorizLine(xpos, xpos+11, ypos+4, Value);
    LCD_DrawHorizLine(xpos+1, xpos+12, ypos+5, Value);
    LCD_DrawHorizLine(xpos+2, xpos+12, ypos+6, Value);
    LCD_DrawHorizLine(xpos+3, xpos+12, ypos+7, Value);
    LCD_DrawHorizLine(xpos+4, xpos+11, ypos+8, Value);
    LCD_DrawHorizLine(xpos+6, xpos+10, ypos+9, Value);

    //Draw Note Tail
    if (yint > 20 || (yint > 8 && yint < 15)){
        //Lower Tail
        LCD_DrawVertLine(xpos, ypos-32, ypos+4, Value);
    }
    else {
        //Upper Tail
        LCD_DrawVertLine(xpos+12, ypos+4, ypos+40, Value);
    }

    if (sharp == 1){
        LCD_DrawSharp(xpos, ypos, Value);
    }

    if (Value == 0xFFFF){
        LCD_RedrawStaff(xpos, yint);
    }
}

void LCD_DrawTrebleClef(void){
```

```
    LCD_DrawTrebleTail();
    LCD_DrawTrebleSpiral();
    LCD_DrawTrebleOutSpiral();

}

void LCD_DrawTrebleTail(void){
    LCD_DrawHorizLine(20, 22, 116, 0x0000);
    LCD_DrawHorizLine(18, 24, 115, 0x0000);
    LCD_DrawHorizLine(17, 24, 114, 0x0000);
    LCD_DrawHorizLine(15, 25, 113, 0x0000);
    LCD_DrawHorizLine(16, 25, 112, 0x0000);
    LCD_DrawHorizLine(16, 25, 111, 0x0000);
    LCD_DrawHorizLine(16, 25, 110, 0x0000);
    LCD_DrawHorizLine(16, 24, 109, 0x0000);
    LCD_DrawHorizLine(17, 23, 108, 0x0000);
    LCD_DrawHorizLine(17, 22, 107, 0x0000);
    LCD_DrawHorizLine(18, 20, 106, 0x0000);
    LCD_DrawHorizLine(19, 22, 105, 0x0000);
    LCD_DrawHorizLine(20, 30, 104, 0x0000);
    LCD_DrawHorizLine(24, 27, 103, 0x0000);
    LCD_DrawHorizLine(28, 31, 105, 0x0000);
    LCD_DrawHorizLine(30, 32, 106, 0x0000);
    LCD_DrawHorizLine(32, 33, 107, 0x0000);
    LCD_DrawHorizLine(32, 34, 108, 0x0000);
    LCD_DrawHorizLine(33, 34, 109, 0x0000);
    LCD_DrawHorizLine(33, 34, 110, 0x0000);

    LCD_DrawVertLine(35, 111, 114, 0x0000);
    LCD_DrawVertLine(34, 111, 118, 0x0000);
    LCD_DrawVertLine(33, 114, 125, 0x0000);
    LCD_DrawVertLine(32, 122, 132, 0x0000);
    LCD_DrawVertLine(31, 129, 137, 0x0000);
    LCD_DrawVertLine(30, 137, 148, 0x0000);

    LCD_DrawHorizLine(22, 33, 125, 0x0000);
    LCD_DrawHorizLine(20, 22, 126, 0x0000);
    LCD_DrawHorizLine(18, 20, 127, 0x0000);
    LCD_DrawHorizLine(17, 18, 128, 0x0000);
    LCD_DrawHorizLine(16, 17, 129, 0x0000);
    LCD_DrawHorizLine(15, 16, 130, 0x0000);
    LCD_DrawHorizLine(14, 15, 131, 0x0000);
    LCD_DrawHorizLine(13, 14, 132, 0x0000);
}

void LCD_DrawTrebleSpiral(void){
    LCD_DrawVertLine(26, 130, 130, 0x0000);
    LCD_DrawVertLine(25, 130, 131, 0x0000);
    LCD_DrawVertLine(24, 131, 132, 0x0000);
    LCD_DrawVertLine(23, 132, 133, 0x0000);
    LCD_DrawVertLine(22, 133, 144, 0x0000);
    LCD_DrawVertLine(21, 134, 142, 0x0000);
    LCD_DrawVertLine(20, 136, 140, 0x0000);
    LCD_DrawVertLine(23, 140, 145, 0x0000);
    LCD_DrawVertLine(24, 141, 146, 0x0000);
    LCD_DrawVertLine(25, 142, 147, 0x0000);
    LCD_DrawVertLine(26, 142, 148, 0x0000);
    LCD_DrawVertLine(27, 143, 148, 0x0000);
    LCD_DrawVertLine(31, 143, 148, 0x0000);
    LCD_DrawVertLine(32, 143, 148, 0x0000);
    LCD_DrawVertLine(33, 143, 148, 0x0000);
    LCD_DrawVertLine(34, 142, 148, 0x0000);
    LCD_DrawVertLine(35, 142, 148, 0x0000);
    LCD_DrawVertLine(36, 141, 148, 0x0000);
    LCD_DrawVertLine(37, 140, 147, 0x0000);
```

```
    LCD_DrawVertLine(38, 138, 146, 0x0000);
    LCD_DrawVertLine(39, 129, 145, 0x0000);
    LCD_DrawVertLine(40, 130, 144, 0x0000);
    LCD_DrawVertLine(41, 132, 142, 0x0000);
    LCD_DrawVertLine(42, 135, 139, 0x0000);
    LCD_DrawVertLine(38, 129, 130, 0x0000);

    LCD_DrawHorizLine(34, 36, 126, 0x0000);
    LCD_DrawHorizLine(36, 37, 127, 0x0000);
    LCD_DrawHorizLine(37, 38, 128, 0x0000);

}

void LCD_DrawTrebleOutSpiral(void){
    LCD_DrawVertLine(13, 133, 136, 0x0000);
    LCD_DrawVertLine(12, 133, 150, 0x0000);
    LCD_DrawVertLine(11, 134, 148, 0x0000);
    LCD_DrawVertLine(10, 137, 144, 0x0000);
    LCD_DrawVertLine(13, 140, 152, 0x0000);
    LCD_DrawVertLine(14, 144, 154, 0x0000);
    LCD_DrawVertLine(15, 146, 155, 0x0000);
    LCD_DrawVertLine(16, 148, 156, 0x0000);
    LCD_DrawVertLine(17, 149, 157, 0x0000);
    LCD_DrawVertLine(18, 151, 158, 0x0000);
    LCD_DrawVertLine(19, 152, 159, 0x0000);
    LCD_DrawVertLine(20, 153, 160, 0x0000);
    LCD_DrawVertLine(21, 154, 161, 0x0000);
    LCD_DrawVertLine(22, 155, 162, 0x0000);
    LCD_DrawVertLine(23, 155, 163, 0x0000);
    LCD_DrawVertLine(24, 156, 163, 0x0000);
    LCD_DrawVertLine(25, 157, 181, 0x0000);
    LCD_DrawVertLine(26, 158, 184, 0x0000);
    LCD_DrawVertLine(27, 154, 166, 0x0000);
    LCD_DrawVertLine(28, 159, 167, 0x0000);
    LCD_DrawVertLine(29, 160, 168, 0x0000);
    LCD_DrawVertLine(30, 161, 169, 0x0000);
    LCD_DrawVertLine(31, 162, 170, 0x0000);
    LCD_DrawVertLine(32, 163, 171, 0x0000);
    LCD_DrawVertLine(33, 164, 172, 0x0000);
    LCD_DrawVertLine(34, 166, 174, 0x0000);
    LCD_DrawVertLine(35, 168, 176, 0x0000);
    LCD_DrawVertLine(36, 170, 186, 0x0000);
    LCD_DrawVertLine(37, 172, 182, 0x0000);
    LCD_DrawVertLine(35, 183, 189, 0x0000);
    LCD_DrawVertLine(34, 184, 190, 0x0000);
    LCD_DrawVertLine(33, 185, 191, 0x0000);
    LCD_DrawVertLine(32, 185, 191, 0x0000);
    LCD_DrawVertLine(31, 184, 191, 0x0000);
    LCD_DrawVertLine(30, 183, 190, 0x0000);
    LCD_DrawVertLine(29, 182, 189, 0x0000);
    LCD_DrawVertLine(28, 180, 188, 0x0000);
    LCD_DrawVertLine(27, 177, 186, 0x0000);

    LCD_DrawVertLine(28, 143, 155, 0x0000);
    LCD_DrawVertLine(29, 142, 151, 0x0000);
}

void LCD_DrawBassClef(void){

    LCD_DrawHorizLine(15, 19, 85, 0x0000);
    LCD_DrawHorizLine(14, 20, 86, 0x0000);
    LCD_DrawHorizLine(13, 21, 87, 0x0000);
    LCD_DrawHorizLine(12, 22, 88, 0x0000);
    LCD_DrawHorizLine(12, 22, 89, 0x0000);
    LCD_DrawHorizLine(12, 22, 90, 0x0000);
    LCD_DrawHorizLine(12, 21, 91, 0x0000);
```

```
    LCD_DrawHorizLine(12, 20, 92, 0x0000);
    LCD_DrawHorizLine(12, 14, 93, 0x0000);
    LCD_DrawHorizLine(12, 14, 94, 0x0000);
    LCD_DrawHorizLine(13, 14, 95, 0x0000);
    LCD_DrawHorizLine(14, 15, 96, 0x0000);
    LCD_DrawHorizLine(15, 16, 97, 0x0000);
    LCD_DrawHorizLine(16, 19, 98, 0x0000);
    LCD_DrawHorizLine(17, 28, 99, 0x0000);
    LCD_DrawHorizLine(25, 30, 98, 0x0000);
    LCD_DrawHorizLine(27, 31, 97, 0x0000);
    LCD_DrawHorizLine(28, 32, 96, 0x0000);
    LCD_DrawHorizLine(29, 33, 95, 0x0000);
    LCD_DrawHorizLine(29, 34, 94, 0x0000);
    LCD_DrawHorizLine(30, 35, 93, 0x0000);
    LCD_DrawHorizLine(30, 35, 92, 0x0000);
    LCD_DrawHorizLine(30, 36, 91, 0x0000);
    LCD_DrawHorizLine(31, 36, 90, 0x0000);
    LCD_DrawHorizLine(31, 36, 89, 0x0000);
    LCD_DrawHorizLine(31, 36, 88, 0x0000);
    LCD_DrawHorizLine(31, 36, 87, 0x0000);
    LCD_DrawHorizLine(31, 36, 86, 0x0000);
    LCD_DrawHorizLine(30, 36, 85, 0x0000);
    LCD_DrawHorizLine(30, 36, 84, 0x0000);
    LCD_DrawHorizLine(30, 35, 83, 0x0000);
    LCD_DrawHorizLine(29, 35, 82, 0x0000);
    LCD_DrawHorizLine(29, 35, 81, 0x0000);
    LCD_DrawHorizLine(29, 35, 80, 0x0000);
    LCD_DrawHorizLine(28, 34, 79, 0x0000);
    LCD_DrawHorizLine(28, 33, 78, 0x0000);
    LCD_DrawHorizLine(27, 33, 77, 0x0000);
    LCD_DrawHorizLine(27, 32, 76, 0x0000);
    LCD_DrawHorizLine(26, 31, 75, 0x0000);
    LCD_DrawHorizLine(26, 30, 74, 0x0000);
    LCD_DrawHorizLine(25, 29, 73, 0x0000);
    LCD_DrawHorizLine(24, 28, 72, 0x0000);
    LCD_DrawHorizLine(23, 27, 71, 0x0000);
    LCD_DrawHorizLine(22, 26, 70, 0x0000);
    LCD_DrawHorizLine(21, 25, 69, 0x0000);
    LCD_DrawHorizLine(20, 23, 68, 0x0000);
    LCD_DrawHorizLine(18, 22, 67, 0x0000);
    LCD_DrawHorizLine(16, 20, 66, 0x0000);
    LCD_DrawHorizLine(15, 18, 65, 0x0000);
    LCD_DrawHorizLine(14, 17, 64, 0x0000);
    LCD_DrawHorizLine(13, 16, 63, 0x0000);
    LCD_DrawHorizLine(11, 14, 62, 0x0000);
    LCD_DrawHorizLine(10, 12, 61, 0x0000);

    //Small Circles
    LCD_DrawHorizLine(40, 41, 95, 0x0000);
    LCD_DrawHorizLine(39, 42, 94, 0x0000);
    LCD_DrawHorizLine(39, 42, 93, 0x0000);
    LCD_DrawHorizLine(40, 41, 92, 0x0000);

    LCD_DrawHorizLine(40, 41, 83, 0x0000);
    LCD_DrawHorizLine(39, 42, 82, 0x0000);
    LCD_DrawHorizLine(39, 42, 81, 0x0000);
    LCD_DrawHorizLine(40, 41, 80, 0x0000);
}

void LCD_DrawSharp(uint16_t x, uint16_t y, uint16_t Value){
    //from x-14, to x-3
    //Bottom line
    LCD_DrawHorizLine(x-11, x-10, y-5, Value);
    LCD_DrawHorizLine(x-11, x-7, y-4, Value);
    LCD_DrawHorizLine(x-11, x-4, y-3, Value);
    LCD_DrawHorizLine(x-11, x-2, y-2, Value);
```

```
    LCD_DrawHorizLine(x-9, x-2, y-1, Value);
    LCD_DrawHorizLine(x-6, x-2, y, Value);
    LCD_DrawHorizLine(x-3, x-2, y+1, Value);

    //Top line
    LCD_DrawHorizLine(x-11, x-10, y+7, Value);
    LCD_DrawHorizLine(x-11, x-7, y+8, Value);
    LCD_DrawHorizLine(x-11, x-4, y+9, Value);
    LCD_DrawHorizLine(x-11, x-2, y+10, Value);
    LCD_DrawHorizLine(x-9, x-2, y+11, Value);
    LCD_DrawHorizLine(x-6, x-2, y+12, Value);
    LCD_DrawHorizLine(x-3, x-2, y+13, Value);

    //First Line
    LCD_DrawVertLine(x-9, y-11, y+18, Value);
    LCD_DrawVertLine(x-8, y-11, y+18, Value);

    //First Line
    LCD_DrawVertLine(x-5, y-9, y+20, Value);
    LCD_DrawVertLine(x-4, y-9, y+20, Value);

}

void Init_StartUp(){
    LCD_DrawStaff();
    Uint16 x = 0;
    Uint16 i = 0;

    for(; i < 55; i++){
        if(x == 9){
            x = 0;
            for(; x < 9; x++){
                LCD_DrawNote(x, LOOKUP_NOTE_NUM[i-9+x], LOOKUP_SHARP[i-9+x], 0xFFFF);
            }
            x = 0;
        }
        if (i < 53) {
            LCD_DrawNote(x, LOOKUP_NOTE_NUM[i], LOOKUP_SHARP[i], 0x0000);
        }
        x++;
        LCD_custom_delay(100);
    }

}

void LCD_RedrawStaff(uint16_t x, uint16_t yint){
    uint16_t ypos = 17 + 6 * (yint);

    // Redraw part of note on line
    if (yint % 2 == 1){
        if ((yint >= 5 && yint <= 13) || (yint >= 16 && yint <= 26)){
            LCD_DrawHorizLine(x-12, x+12, ypos+4, 0x0000);
            LCD_DrawHorizLine(x-12, x+12, ypos+5, 0x0000);
        }
        if ((yint <= 27 && yint >= 19) || (yint <= 15 && yint >= 7)){
            LCD_DrawHorizLine(x-12, x, ypos-7, 0x0000);
            LCD_DrawHorizLine(x-12, x, ypos-8, 0x0000);
        }
        if ((yint <= 23 && yint >= 15) || (yint <= 11 && yint >= 3)){
            LCD_DrawHorizLine(x-12, x, ypos+16, 0x0000);
            LCD_DrawHorizLine(x-12, x, ypos+17, 0x0000);
        }
    }
    else {
        if ((yint <= 24 && yint >= 16) || (yint <= 12 && yint >= 4)){
            LCD_DrawHorizLine(x-12, x, ypos+10, 0x0000);
```

```
            LCD_DrawHorizLine(x-12, x, ypos+11, 0x0000);
        }
        if ((yint <= 26 && yint >= 18) || (yint <= 14 && yint >= 6)){
            LCD_DrawHorizLine(x-12, x, ypos-2, 0x0000);
            LCD_DrawHorizLine(x-12, x, ypos-1, 0x0000);
        }
    }


    //Redraw Staff from lines
    uint16_t tailpos;
    if (yint > 20 || (yint > 8 && yint < 15)){
        //Lower Tail
        tailpos = x;
    }
    else {
        //Upper Tail
        tailpos = x+12;
    }

    //Redraw staff along tail line
    LCD_DrawVertLine(tailpos, 51, 52, 0x0000);
    LCD_DrawVertLine(tailpos, 63, 64, 0x0000);
    LCD_DrawVertLine(tailpos, 75, 76, 0x0000);
    LCD_DrawVertLine(tailpos, 87, 88, 0x0000);
    LCD_DrawVertLine(tailpos, 99, 100, 0x0000);
    LCD_DrawVertLine(tailpos, 123, 124, 0x0000);
    LCD_DrawVertLine(tailpos, 135, 136, 0x0000);
    LCD_DrawVertLine(tailpos, 147, 148, 0x0000);
    LCD_DrawVertLine(tailpos, 159, 160, 0x0000);
    LCD_DrawVertLine(tailpos, 171, 172, 0x0000);

}

//
// spi_xmit - Transmit value via SPI
//
void spi_xmit(Uint16 a)
{
    SpiaRegs.SPITXBUF = (a & 0xFF) << 8;
}

//
// spi_fifo_init - Initialize SPIA FIFO
//
void spi_custom_init()
{
    //Clear reset to start configuration
    SpiaRegs.SPICCR.bit.SPISWRESET = 0;

    //Choose Master/Slave
    SpiaRegs.SPICTL.bit.MASTER_SLAVE = 1;
    SpiaRegs.SPICTL.bit.TALK = 1;

    //Choose SPICLK polarity and CLK_PHase
    SpiaRegs.SPICCR.bit.CLKPOLARITY = 1;
    SpiaRegs.SPICTL.bit.CLK_PHASE = 0;
    SpiaRegs.SPICCR.bit.SPILBK = 0;

    SpiaRegs.SPICTL.bit.SPIINTENA = 0;

    //Set Baud Rate
    SpiaRegs.SPIBRR.bit.SPI_BIT_RATE = 9;

    //Set the spi character length
    SpiaRegs.SPICCR.bit.SPICHAR = 0x7;
```

```
    //CLear the SPI Flags
    SpiaRegs.SPISTS.bit.OVERRUN_FLAG = 0;
    SpiaRegs.SPISTS.bit.INT_FLAG = 0;

    //Enable SPISTE inversion?
    //GPIO_WritePin(57, 1);

    // Set FREE bit
    // Halting on a breakpoint will not halt the SPI
    SpiaRegs.SPIPRI.bit.FREE = 1;

    //
    // Initialize SPI FIFO registers
    //
    SpiaRegs.SPIFFTX.all = 0xE040;
    SpiaRegs.SPIFFRX.all = 0x2044;
    SpiaRegs.SPIFFCT.all = 0x0;

    //Unclear reset to finish
    SpiaRegs.SPICCR.bit.SPISWRESET = 1;


}

void spi_fifo_init()
{
    spi_custom_init();
}

void Init_LCD_Pins(void) {

    GPIO_SetupPinMux(57, GPIO_MUX_CPU1, 1); //SPISTEA
    GPIO_SetupPinMux(58, GPIO_MUX_CPU1, 15); //SPISIMOA - LCD_SDI
    GPIO_SetupPinMux(59, GPIO_MUX_CPU1, 15); //SPISOMIA
    GPIO_SetupPinMux(60, GPIO_MUX_CPU1, 15); //SPICLKA - LCD_SCL
    GPIO_SetupPinMux(61, GPIO_MUX_CPU1, 0); //LCD_SDC
    GPIO_SetupPinMux(72, GPIO_MUX_CPU1, 0); //LCD_SCS
    GPIO_SetupPinMux(12, GPIO_MUX_CPU1, 0); //LCD_PWM
    GPIO_SetupPinMux(20, GPIO_MUX_CPU1, 0); //LCD_RESET, maybe

    //LCD_RESET
    GPIO_SetupPinOptions(20, GPIO_OUTPUT, GPIO_PULLUP | GPIO_ASYNC);
    GPIO_WritePin(LCD_RESET, 0);

    //LCD_SDC
    GPIO_SetupPinOptions(61, GPIO_OUTPUT, GPIO_PULLUP | GPIO_ASYNC);
    GPIO_WritePin(LCD_SDC, 0);

    //LCD_SCS
    GPIO_SetupPinOptions(72, GPIO_OUTPUT, GPIO_PULLUP | GPIO_ASYNC);
    GPIO_WritePin(LCD_SCS, 1);

    //LCD_PWM
    GPIO_SetupPinOptions(12, GPIO_OUTPUT, GPIO_PULLUP | GPIO_ASYNC);

    //SPI Pins
    GPIO_SetupPinOptions(57, GPIO_OUTPUT, GPIO_INVERT | GPIO_ASYNC); //SPISTEA
    GPIO_SetupPinOptions(58, GPIO_OUTPUT, GPIO_PULLUP | GPIO_ASYNC); //SPISIMOA
    GPIO_SetupPinOptions(59, GPIO_INPUT, GPIO_PULLUP | GPIO_ASYNC); //SPISOMIA
    GPIO_SetupPinOptions(60, GPIO_OUTPUT, GPIO_PULLUP | GPIO_ASYNC); //SPICLKA

    spi_fifo_init();

    //Set LCD Backlight high to enable
    GPIO_WritePin(LCD_PWM, 1);
```

```
    //Set to default
    GPIO_WritePin(LCD_SDC, 1);
    GPIO_WritePin(LCD_SCS, 0);

    LCD_custom_delay(1);

    //Deassert the lcd reset signal
    GPIO_WritePin(LCD_RESET, 1);

    LCD_custom_delay(1);


}

void LCD_custom_delay(uint16_t msec)
{
    uint32_t i=0;
    uint32_t time=(msec/10)*(SYSTEM_CLOCK_SPEED/(2*15*100));
    for(i=0;i<time;i++);
}


/**************************************************************************************************************
***
* ADC Interface
*
* Author:
* Stuart Miller
* Missouri University of Science & Technology
* Computer Engineering
* 2017
*
**************************************************************************************************************
**/


/*********************************************************
* Headers
*********************************************************/
#include "adc_intf.h"
#include "F28x_Project.h"


/*********************************************************
* Global Variables
*********************************************************/


/*********************************************************
* Configure ADC
*********************************************************/
void ConfigureADC(void)
    {
    EALLOW;                                     // allow writing to protected registers
    AdcaRegs.ADCCTL2.bit.PRESCALE = 6;          // set ADCCLK divider to /4
    AdcSetMode( ADC_ADCA, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE );
    AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;       // set pulse position to late
    AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;          // power up the ADC
    EDIS;                                       // allow writing to protected registers

    // 1ms delay for powerup time
    volatile uint32_t j;
    for(j=10000; j>0; j--);
    //DELAY_US(1000);
    }
```

```
/************************************************************
* Setup for EPWM trigger
************************************************************/
void SetupADCEpwm(Uint16 channel)
{
    // determine minimum acquisition window (in SYSCLKS) based on resolution
    Uint16 acqps;
    if( ADC_RESOLUTION_12BIT == AdcaRegs.ADCCTL2.bit.RESOLUTION )
        {
        acqps = 14; // 75ns
        }
    else // resolution is 16-bit
        {
        acqps = 63; // 320ns
        }
    EALLOW;                                     // allow writing to protected registers
    AdcaRegs.ADCSOC0CTL.bit.CHSEL = channel;    // SOC0 will convert pin A0
    AdcaRegs.ADCSOC0CTL.bit.ACQPS = acqps;      // sample window is 100 SYSCLK cycles
    AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 5;        // trigger on ePWM1 SOCA/C
    AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 0;      // end of SOC0 will set INT1 flag
    AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1;        // enable INT1 flag
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;      // make sure INT1 flag is cleared
    EDIS;                                       // disallow writing to protected registers
}

/************************************************************
* Setup for software trigger
************************************************************/
void SetupADCSoftware(void)
    {
    // Set minimum acquisition window (in SYSCLKS) based on resolution
    Uint16 acqps;
    if( ADC_RESOLUTION_12BIT == AdcaRegs.ADCCTL2.bit.RESOLUTION )
        {
        acqps = 14; // 75ns
        }
    else //resolution is 16-bit
        {
        acqps = 63; // 320ns
        }

    EALLOW;                                 // allow writing to protected registers
    AdcaRegs.ADCSOC0CTL.bit.CHSEL = 0;      // SOC0 will convert pin A0 (pin 27 on our launchpad)
    AdcaRegs.ADCSOC0CTL.bit.ACQPS = acqps;  // sample window is acqps + 1 SYSCLK cycles
    AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 1;  // end of SOC1 will set INT1 flag
    AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1;    // enable INT1 flag
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;  // make sure INT1 flag is cleared
    EDIS;                                   // disallow writing to protected registers
    }


/************************************************************
* ConfigureEPWM - Configure EPWM SOC and compare values
************************************************************/
void ConfigureEPWM(void)
{
    EALLOW;                                 // allow writing to protected registers
    CpuSysRegs.PCLKCR2.bit.EPWM1 = 1;       // enable epwm clock
    EPwm1Regs.ETSEL.bit.SOCAEN = 0;         // disable SOC on A group
    EPwm1Regs.ETSEL.bit.SOCASEL = 4;        // select SOC on up-count
    EPwm1Regs.ETPS.bit.SOCAPRD = 1;         // generate pulse on 1st event
    EPwm1Regs.CMPA.bit.CMPA = EPWM_CNTS / 2;// set compare A value
    EPwm1Regs.TBPRD = EPWM_CNTS;            // set period
    EPwm1Regs.TBCTL.bit.CTRMODE = 3;        // freeze counter
    EDIS;                                   // disallow writing to protected registers
```

```
}

/*************************************************************
* Set the resolution and signalmode for a given ADC. This will
* ensure that the correct trim is loaded.
*************************************************************/
void AdcSetMode( Uint16 adc, Uint16 resolution, Uint16 signalmode )
    {
    Uint16 adcOffsetTrimOTPIndex;           // index into OTP table of ADC offset trims
    Uint16 adcOffsetTrim;                   // temporary ADC offset trim

    // Re-populate INL trim
    CalAdcINL( adc );

    if( 0xFFFF != *( ( Uint16* )GetAdcOffsetTrimOTP ) )
    {
        //
        //offset trim function is programmed into OTP, so call it
        //

        //
        //calculate the index into OTP table of offset trims and call
        //function to return the correct offset trim
        //
        adcOffsetTrimOTPIndex = 4*adc + 2*resolution + 1*signalmode;
        adcOffsetTrim = (*GetAdcOffsetTrimOTP)(adcOffsetTrimOTPIndex);
    }
    else
    {
        //
        //offset trim function is not populated, so set offset trim to 0
        //
        adcOffsetTrim = 0;
    }

    //
    //Apply the resolution and signalmode to the specified ADC.
    //Also apply the offset trim and, if needed, linearity trim correction.
    //
    switch(adc)
    {
        case ADC_ADCA:
            AdcaRegs.ADCCTL2.bit.RESOLUTION = resolution;
            AdcaRegs.ADCCTL2.bit.SIGNALMODE = signalmode;
            AdcaRegs.ADCOFFTRIM.all = adcOffsetTrim;
            if(ADC_RESOLUTION_12BIT == resolution)
            {
                //
                //12-bit linearity trim workaround
                //
                AdcaRegs.ADCINLTRIM1 &= 0xFFFF0000;
                AdcaRegs.ADCINLTRIM2 &= 0xFFFF0000;
                AdcaRegs.ADCINLTRIM4 &= 0xFFFF0000;
                AdcaRegs.ADCINLTRIM5 &= 0xFFFF0000;
            }
        break;
        /*
        case ADC_ADCB:
            AdcbRegs.ADCCTL2.bit.RESOLUTION = resolution;
            AdcbRegs.ADCCTL2.bit.SIGNALMODE = signalmode;
            AdcbRegs.ADCOFFTRIM.all = adcOffsetTrim;
            if(ADC_RESOLUTION_12BIT == resolution)
            {
                //
                //12-bit linearity trim workaround
                //
```

```
                AdcbRegs.ADCINLTRIM1 &= 0xFFFF0000;
                AdcbRegs.ADCINLTRIM2 &= 0xFFFF0000;
                AdcbRegs.ADCINLTRIM4 &= 0xFFFF0000;
                AdcbRegs.ADCINLTRIM5 &= 0xFFFF0000;
            }
        break;
        case ADC_ADCC:
            AdccRegs.ADCCTL2.bit.RESOLUTION = resolution;
            AdccRegs.ADCCTL2.bit.SIGNALMODE = signalmode;
            AdccRegs.ADCOFFTRIM.all = adcOffsetTrim;
            if(ADC_RESOLUTION_12BIT == resolution)
            {
                //
                //12-bit linearity trim workaround
                //
                AdccRegs.ADCINLTRIM1 &= 0xFFFF0000;
                AdccRegs.ADCINLTRIM2 &= 0xFFFF0000;
                AdccRegs.ADCINLTRIM4 &= 0xFFFF0000;
                AdccRegs.ADCINLTRIM5 &= 0xFFFF0000;
            }
        break;
        case ADC_ADCD:
            AdcdRegs.ADCCTL2.bit.RESOLUTION = resolution;
            AdcdRegs.ADCCTL2.bit.SIGNALMODE = signalmode;
            AdcdRegs.ADCOFFTRIM.all = adcOffsetTrim;
            if(ADC_RESOLUTION_12BIT == resolution)
            {
                //
                //12-bit linearity trim workaround
                //
                AdcdRegs.ADCINLTRIM1 &= 0xFFFF0000;
                AdcdRegs.ADCINLTRIM2 &= 0xFFFF0000;
                AdcdRegs.ADCINLTRIM4 &= 0xFFFF0000;
                AdcdRegs.ADCINLTRIM5 &= 0xFFFF0000;
            }
        break;
        */
    }
}

//
// CalAdcINL - Loads INL trim values from OTP into the trim registers of the
//             specified ADC. Use only as part of AdcSetMode function, since
//             linearity trim correction is needed for some modes.
//
void CalAdcINL(Uint16 adc)
{
    switch(adc)
    {
        case ADC_ADCA:
            if(0xFFFF != *((Uint16*)CalAdcaINL))
            {
                //
                //trim function is programmed into OTP, so call it
                //
                (*CalAdcaINL)();
            }
            else
            {
                //
                //do nothing, no INL trim function populated
                //
            }
            break;
        case ADC_ADCB:
            if(0xFFFF != *((Uint16*)CalAdcbINL))
```

```
            {
                //
                //trim function is programmed into OTP, so call it
                //
                (*CalAdcbINL)();
            }
            else
            {
                //
                //do nothing, no INL trim function populated
                //
            }
            break;
        case ADC_ADCC:
            if(0xFFFF != *((Uint16*)CalAdccINL))
            {
                //
                //trim function is programmed into OTP, so call it
                //
                (*CalAdccINL)();
            }
            else
            {
                //
                //do nothing, no INL trim function populated
                //
            }
            break;
        case ADC_ADCD:
            if(0xFFFF != *((Uint16*)CalAdcdINL))
            {
                //
                //trim function is programmed into OTP, so call it
                //
                (*CalAdcdINL)();
            }
            else
            {
                //
                //do nothing, no INL trim function populated
                //
            }
            break;
    }
}
```