REAL-TIME DIGITAL EFFECTS PROCESSING USING iOS

Jonah Clinard
Advisor: Dr. Wayne Pilkington
CPE 462 – Senior Project
California Polytechnic State University, San Luis Obispo
Computer Engineering Program

Table of Contents

List of Tables and Figures

Acknowledgements

I'd like to thank Dr. Pilkington, and Taylor Collins, for your help, guidance, patience, and incredible support, without you this project would have been nothing more than a dream. My family for your unconditional love. My friends for the humor and encouragement. Finally I'd like to thank Music, for being my voice when words fall short.

Abstract

In today's society, we are seeing incredible improvements in terms of creating smaller technological devices that behave more and more like the personal computers of yesterday. Mobile "Smart" devices, in particular, are becoming incredibly powerful not just in terms of processing power, but in the fact that they are able to provide assistance to users in their everyday lives. Application developers are now able utilize the power and size of these devices, to create and realize ideas that would have been previously viewed as impossible. This project applies the fields of digital signal processing, music, and mobile application development, to effectively create a Real-Time digital effects processing application running on iOS.

I. Introduction

This project documents the overall development of a software application running on iOS, to perform real-time digital effects processing on a signal obtained from the devices onboard microphone input.

Motivation

Smartphones have become so widely available and powerful that a recent reporting (March 2015) showed that there were nearly 187.5 million smart phone users in the United States [1]. This number of users offers mobile application developers an incredibly large base to share their creative visions with. Taking into consideration the number of users that have smart phones, and the raw power of the hardware that these devices contain, the possibilities for application development become ever increasing.

Having an application on a smart device perform digital effect processing offers an incredible convenience and cost savings to the end user. On average, effects units targeted towards guitar players will cost around $80 - $100. These effect units will generally provide a user with a single effect, or process a signal in single way. This results in the consumer paying potentially hundreds of dollars on effects processors alone. These effects units will also have to be hauled around with the user everywhere they go, which can become a fairly large inconvenience if the user has multiple effects units.

Having an application that would provide these same effects and that could be used on a smart device, would offer to the user the ability to carry all their effects processing needs on a single device, and this device would be something they already used and carried in their everyday lives.

Definition of intended/user/customer/context

The intended user for this application is anyone who is interested in using a mobile application to provide a digital effects solution to be used with a musical instrument. The intended purpose of this application is to provide the user with a functional and intuitive method for applying digital effects to a musical signal, though the application will have no restrictions on the context of either the input or output signal. Necessary requirements and restrictions on the use of this application are as follows:

- The user must have access to and have the general knowledge of how to operate an iOS device.
    - This is necessary to ensure a positive user's experience with the application. Without access to/general knowledge of how to operate an iOS device the user will not be able to fully benefit from the features/functionality offered by this application.
- The user must have access and the ability to connect some analog input signal to the iOS device being used. – This is necessary in order to actually obtain a signal to process. The input signal may be connected to the iOS device in a multitude of ways, this project will seek to take the analog signal and sample it via the 3.5mm headphone jack onboard the iOS device.
- The user should have a musical instrument that provides some form of an analog output signal, such as an electric guitar, bass, keyboard etc. – This is necessary in order to ensure an engaging and positive experience for the user.

Market Alternatives

Other alternatives to this application do exist, the most popular for iOS devices is Garageband by Apple. Garageband offers users the ability to record and create music on their mobile iOS devices. In addition to the recording capabilities, Garageband also offers a number of virtual instruments and digital effects to the user. However, this application seeks to provide an implementation that is cost free to the user and involves sophisticated and clever use of digital signal processing to ensure and produce a high quality, and pleasant user experience. In addition to the prior mentioned this application represents an application of the passion and education that I, Jonah Clinard, have received while attending California Polytechnic University in San Luis Obispo, California.

II. Background

Digital Signal Processing

Digital signal processing involves the operation of using mathematical techniques to process or alter a digital signal. In general digital signal processing can perform the same functionality of analog circuitry, but is even more expansive in its ability to finely tune signals with a great level of control. This iOS application will seek to take various concepts of digital signal processing and apply them in order to achieve its desired functionality. In particular the concepts of digital signal manipulation to produce musical effects will be used in this application, therefore, this project in no way attempts to describe or define all the abilities that one may accomplish though DSP.

Mobile Based Computing

Mobile based computing entails the use of devices that are inherently more portable to perform similar functionality to that of a desktop computer. Such devices are commonly seen in the form of tablets, and cellular phones. One of the main goals of this project is provide a musical effects solution available via a mobile computing device. Figure II.i below shows a comparison of the total market share occupied by mobile platforms. Three platforms will be discussed: iOS, Android, and Windows Phone OS. The other shareholding platforms: Java ME, Symbian, Blackberry, and other devices will not be discussed in terms of this project due to their low share holdings.

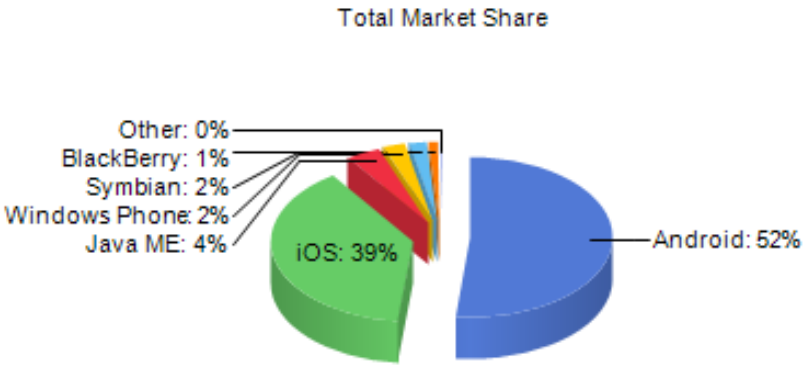

Figure II.i - Total Market Share of Mobile based Operating Systems Source: https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1

Available Mobile Development Platforms

iOS by Apple

iOS is a distributed mobile operating system developed by the technology company Apple. This operating system runs exclusively on Apple's line of i-based devices. Such devices include the iPhone, iPod, and iPad. Currently iOS is on version 8.3 of its release. With this release Apple improved the general overall performance of the operating system as well as a large number of social and technology based Application Programmer Interfaces (APIs). Though other mobile based

Android by Google

Android is a distributed mobile operating system developed by the technology company Google. This operating system runs on a wide range of devices, employing the "Be different, not the same" design philosophy. Android is an entirely open source platform, meaning its source code is shared freely to the public. Currently Android is on its 5.1 Lollipop release. Lollipop represents a complete overhaul in terms of Android's user interface, as well as several general performance improvements.

Windows Phone OS by Microsoft

Windows Mobile is a distributed mobile operating system developed by the technology company Microsoft. This operating system is currently on version 8.1 of its release. The Windows Phone OS runs exclusively on windows phone devices and tablets.

Digital Effects

Digital Delay

The concept behind a digital delay is fairly straightforward. At the most basic level the effect combines the current output signal with time delayed versions of the same signal. The following diagram shows a basic implementation of the Digital Delay effect. We can see that the filter used for implementation is an FIR filter with order equal to number of delays.

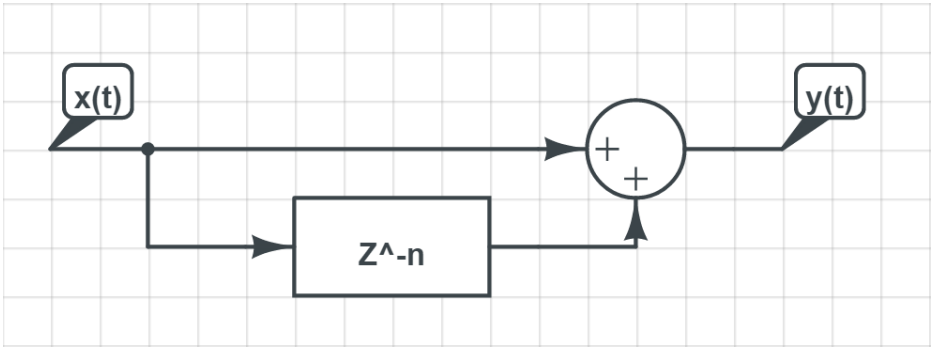

Figure II.ii - Digital Delay Realization

Flanger

Flangers work off of the principles of the Doppler effect. By combining the current output signal with varying short delayed versions of itself (0 – 10 milliseconds) the perceived signal appears to have a slight pitch modulation. The following diagram shows the basic implementation of the flanger

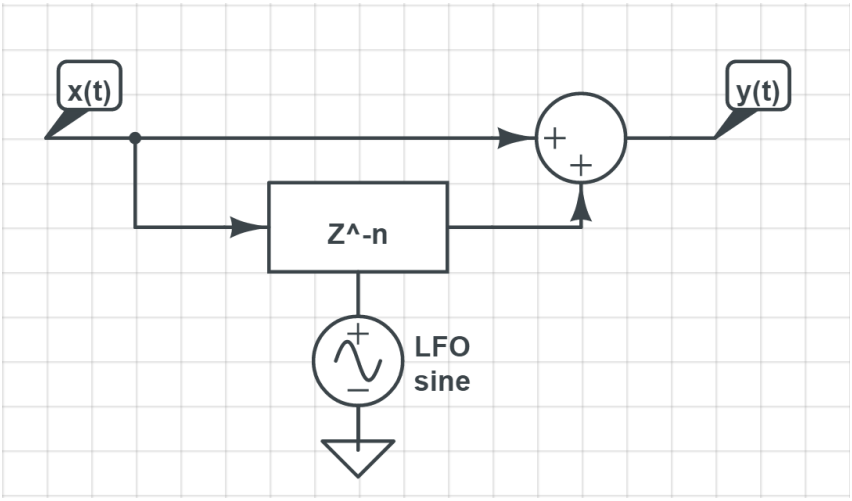

Figure II.iii - Digital Flanger Realization

Tremolo

The tremolo effect works using the principles of amplitude modulation. By modulating the current signal with an oscillator we are able to produce a “shuttering” effect characteristic of the tremolo.

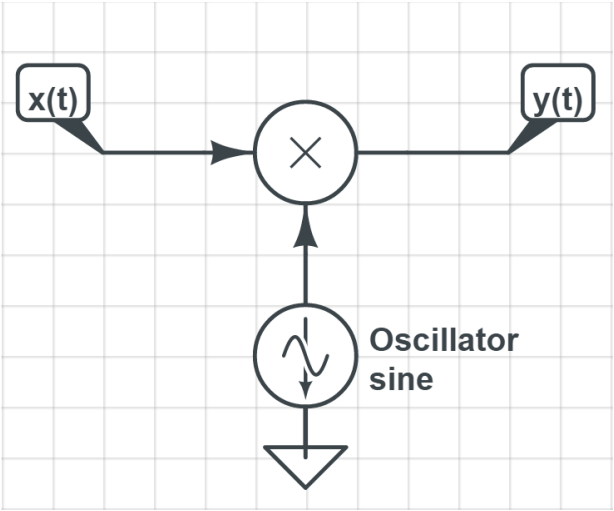

Figure II.iv - Digital Tremolo Realization

III. Product Design Engineering Requirements

Functional and Feature Requirements

This application obtains input in the form of a musical analog signal and processes this signal with the user selected effects all in “real time.” This means that the effects processing is performed in an efficient manner that will prevent the user from noticing any discernable latencies between the input signal and the dry output signal, unless intended by the effect, for example a digital delay. The features of the application include:

- Adjustable effects parameters
    - Intensity of the effect
    - Variable actions associated with the effect
    - Output volume of the signal
- Interactive User interface that provides visual feedback to the user’s actions
- Provided Effects
    - Delay
    - Reverb
    - Flanger
    - Tremolo

Performance Specifications

The following are a list of specific metrics that will be used to evaluate the proper functionality of the application.

- Overall output latency of no greater than 13ms vs. the detected input signal.
- Minimum 12bit sampling resolution - This speciation is in terms of the input signal being sampled.
- Minimum 22kHz sampling frequency – Following the nyquist criteria this should provide us with a bandwidth of around 11kHz, which should be wide enough to capture most of the produced harmonic content from an instrument such as an electric guitar.
- Maximum 10 second initial application load time.

Blackbox Diagram (Level 0 Diagram)

Figure III.i shows the level 0 blackbox diagram of the system. The flow of the system will be as follows:

1. Signal input from the Input Device/Signal Generation component to the iOS device
2. iOS Device receives the input signal which it then processes as determined by the application. Then iOS device takes the processed signal and outputs it.
3. The signal is output from the iOS device which can be observed both via a connected oscilloscope, and heard by connecting speakers/headphones to the device.

Figure III.i - Level 0 Blackbox Diagram

Description of User Interface

The following are a listing of the user controlled variables that the application works in partner with.

- Effect volume level – A relative ratio of the blend of the processed output signal vs. the clean input signal.
- Effect feedback level – Where necessary (effect based) the feedback level of the effect that will route from the processed output back to the input to the effects processor.
- Digital sliders that will be displayed on the devices screen will allow over the control of these effects/variables.
- Effect selection – Provides the user with the ability to process the signal using more than one effect (this may be limited to the effects being used).
- Output volume control – The iOS device will have hardware buttons that will control the level of the output signal from the application.

The following image shows the Applications User Interface. We can see the provided controls and parameters for each effect. This is the only user interface element of the application, therefore the following represents the main and only view users will interact with.

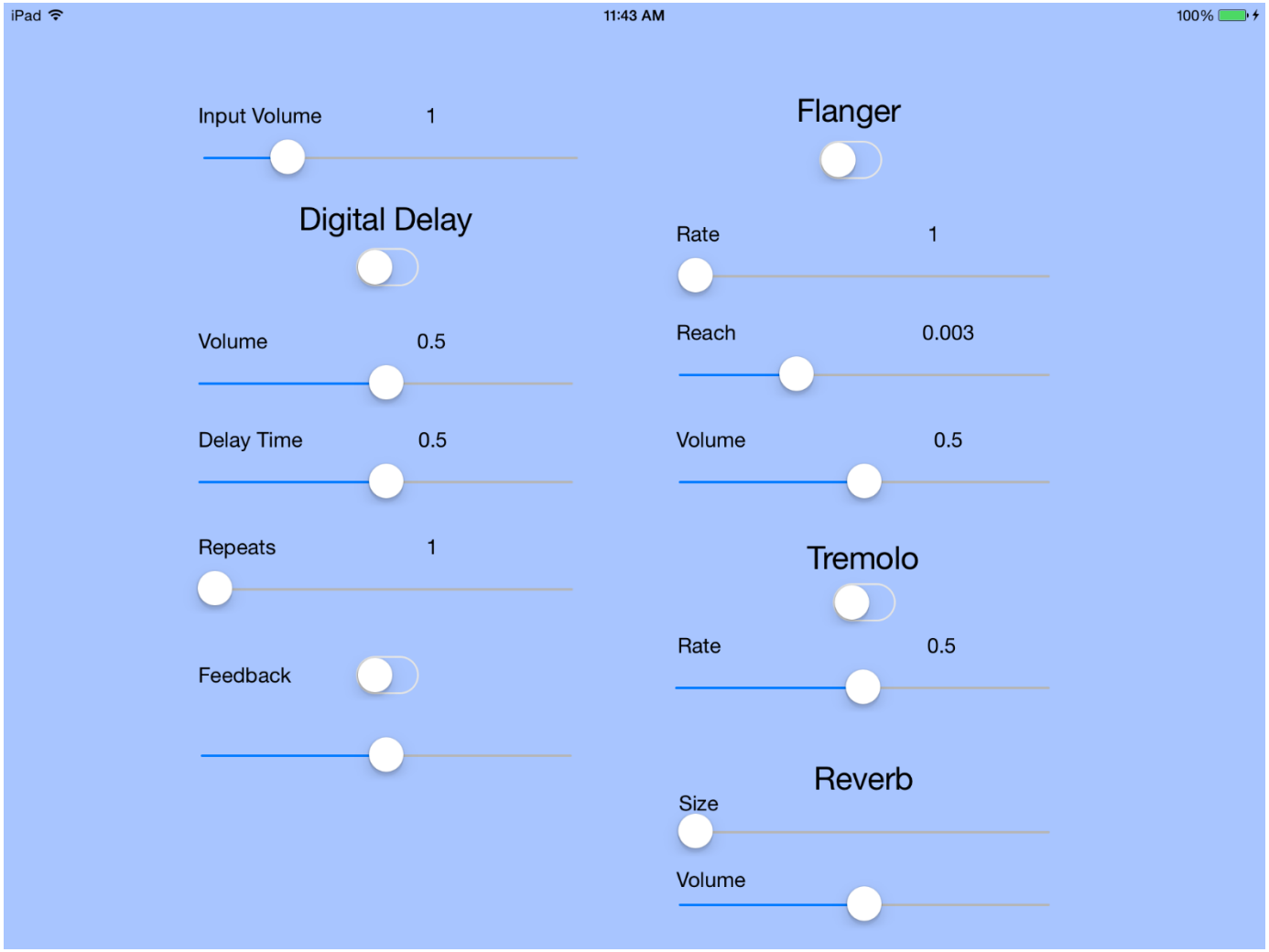

Figure III.ii - Application User Interface

IV. System Design

The following figures shown in the Input Device and iOS device sub-sections show the signal flow paths for the processed signal.

Input Device

Figure IV.i - Level 1 Input Device Diagram

iOS Device

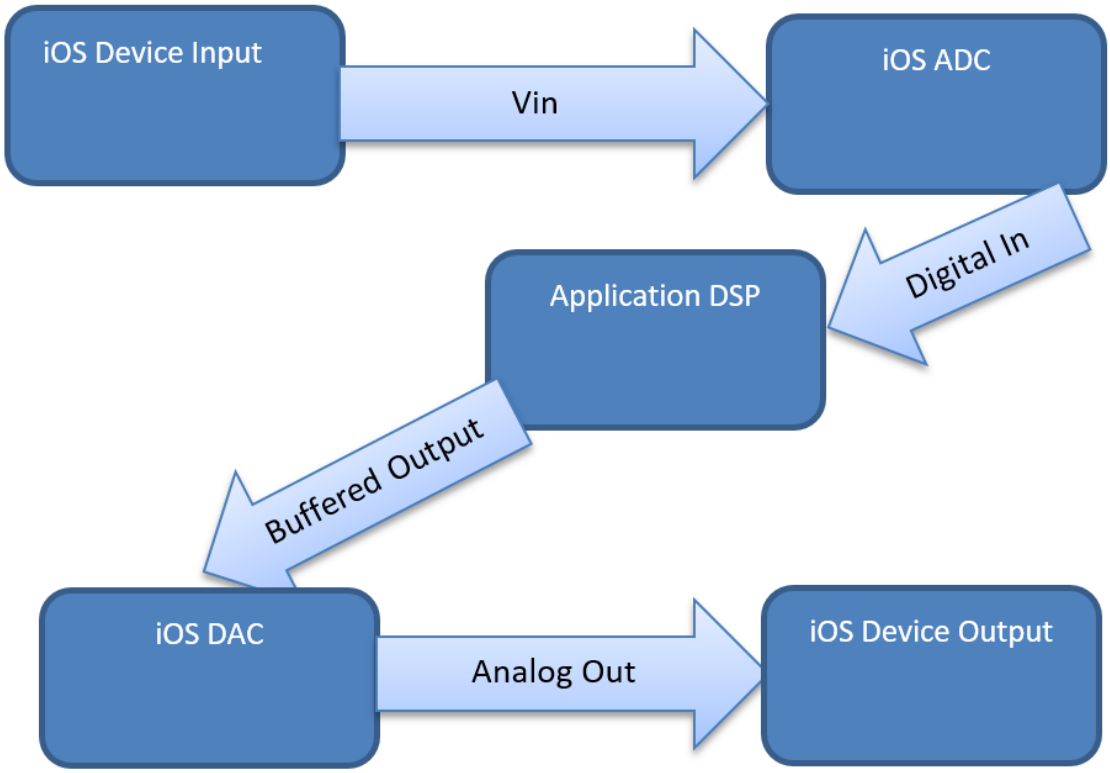

Figure IV.ii - Level 1 iOS Device signal flow

Application Software Flow Chart

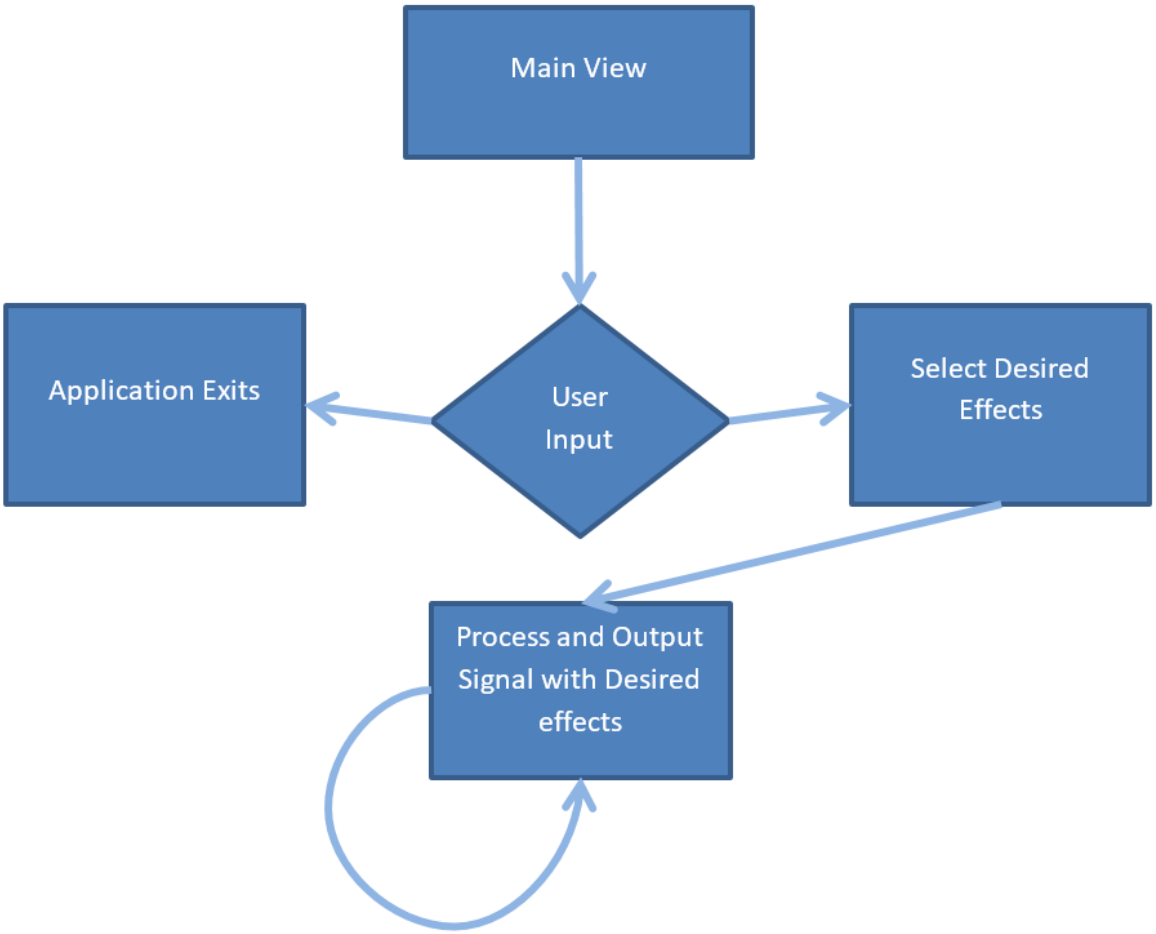

Figure IV.iii - Software Flow Chart

In general the software flows as follows

1. The user is presented with the main application screen
2. The user provides feedback to the application via the touchscreen
    a. If the user signals to the application that they would like to exit, the application will exit.
    b. If the user provides the application with what effect they would like to use the application will obtain the digital signal and process the signal with the desired effects.
3. Unless the user provides instructions to the application to exit or modify the effects being applied to the signal, the application will repeat the “Process and Output Signal with Desired Effects” block.

V. Technology Choices and Design Approach Alternatives Considered

The main selection choice came in regards to which mobile operating platform would be used. The choices were between the following:

- iOS by Apple
- Android by Google
- Windows Mobile OS by Microsoft

In order to determine which platform was to be used the platforms were ranked across several categories, which were:

- Accessibility – How user friendly is the device
- Audio Latency – How quickly can audio be processed and output
- Developer Toolkit (APIs) – How much documentation and functionality is provided to the developer?
- Overall Operating System Performance – In terms of overall performance how efficiently does the operating system perform?

The effects of the above rankings were considered and assigned associated weightings. Using the associated weightings the following decision matrix was generated:

Table V.i - Operating System Decision Matrix

		iOS	Android	Windows Phone OS
Accessibility	.15	10	9	8
Audio Latency	.45	9	5	5
Developer Toolkit	.25	7	9	6
Overall Operating System Performance	.15	7	7	4
Score		8.35	6.9	5.55

We can see that the greatest weighting is placed on the audio latency of the system. This is necessary because the intended functionality of the application is to operate in a real time fashion. The greater the latency corresponds to an undesired and noticeable delay, which will break the intended functionality and overall experience of the application. The developer toolkit takes the second highest ranking to account for the ease and friendliness of the platform. The accessibility and overall operating system performance result in equal weighting, in that while both are important considerations, they don't affect the design considerations as much as the others.

We can see that iOS ultimately wins, this is due to the fact that research showed that the throughput of the device can be seen in as little as 3ms, whereas Android results in 15ms in the most optimized fashions, and Windows Phone OS comes in around 25ms. Since the highest weighting was associated

with audio latency, iOS results in the overall winner. In addition it provides a high level of accessibility as well as exceptional developer provided APIs with good overall system performance.

VI. Project Design Description

The following figure is a level 2 software flow diagram. This diagram shows the flow of the application from audio obtainment to audio output. In general the process is fairly simple in terms of individual components. The process to follow will be in the form of making a call to obtain audio from the ADC via a helper method provided from the Novocaine audio SDK. Once this call is completed we will make a call to place the obtained audio into a circular buffer. After this buffer is populated, the application checks which effects the user wishes to apply to the signal, after which processing of the audio can then take place. Once the audio is processed it is then written out using the Novocaine audio SDK helper method.

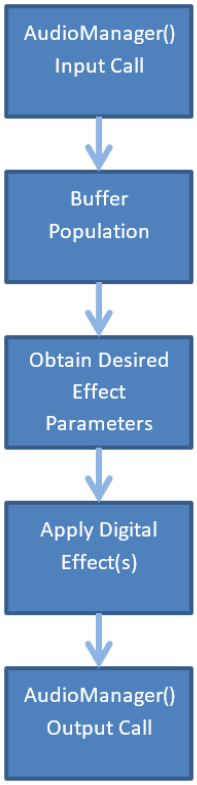

Figure VI.i - Level 2 Software flow diagram

Since audio may be input to the iOS device in a multitude of ways we will take some time to examine why we are simply using analog input, when we could be using Bluetooth, or USB. In terms of quality the analog signal obtained via the headphone port on the iOS device will not be better and potentially will be worse than a digital signal transmitted over USB via the lightning port on the iOS device. Bluetooth will be prone to cross channel interference and results in an over higher complexity in terms of implementation, so it was ruled out. USB audio also results in higher complexity, though in terms of quality it provides the greatest. Thought is currently being put into whether or not USB audio support

will be implemented. Due to the timeframe of this project, for now audio will be handled via the headphone port on the iOS device.

VII. Physical Construction and Integration

The overall physical implementation of this project will focus primarily with working with the provided ADC and DAC provided from the headphone input and output jack onboard the iOS Device. In general the only external hardware that may be needed (but does not affect the functionality of the application) will be in the form of pre-amplification of the input signal and post amplification of the output signal. This is necessary to provide the application with a signal that operates at line level as opposed to instrument level.

VIII. Integrated System Tests

In order to provide the best overall end experience to users of the application vigorous testing plans will need to be implemented in order to ensure that the application functions with no bugs, and as quickly as possible. The following are a proposed list of planned test procedures that will help to ensure the correct functionality of the application

Audio Effect Throughput Test

In order to test the real time processing effectiveness of the application multiple trials will be ran with each individual audio effect applied as well as with a combination of each in every scenario. The following table will be used to document and record the overall audio throughput of the device. In order to measure the audio latency software timestamps will be logged dictating the overall time an effect block takes to complete. Using this information we can determine the throughput of the audio via the effects path.

Table VIII.i - Effect Throughput Testing

Run/Effect	Digital Delay (seconds)	Tremolo (seconds)	Flanger (seconds)	Total Time (seconds)
1	Time run 1	Time run 1	Time run 1	Time run 1
2	Time run 2	Time run 2	Time run 2	Time run 2
3	Time run 3	Time run 3	Time run 3	Time run 3
Average (seconds)	Average time	Average time	Average time	Average time

Buffer Length Test

Each effect that is processed will result in more overhead, which results in more overall usage of the processor. Because of this the audio will have to be buffered before being output. A larger buffer size allows the application to perform more processing between writes to the DAC. This means that larger buffer sizes will provide the best quality output, but will also add latency to the system. Most of this is provided behind the scenes from the kernel, but as a developer you still have levels of control over the buffer length. Finding the correct length results in having a long enough buffer to allow your application

to breathe, but also short enough that the output writes of the application are not noticeable (less latent). In order to test this, the buffer size will be adjusted until the optimal level is found. This will need to be tested with every individual effect as well as combination of them all. In order to ensure that the audio does not encounter buffer underruns the audio will be monitored to ensure that no pops or clicks are present when processing the audio.

User Experience QA

User Experience quality assurance testing will be performed in order to ensure a pleasant overall user experience. This testing is directed at the average level user whom might not know all the ins and outs of the application upon usage. The following test plan and questionnaire will be implemented and tested with multiple individuals.

1. Launch the application
    a. Are you presented with a welcome message/direction on how to proceed with the application?
    b. Is the visual layout aesthetically pleasing to you? Is it easy to see what you are doing?
2. Apply an effect
    a. Can you hear a noticeable difference?
    b. Is there anything unpleasant about the audible difference?
    c. Are you easily able to disable the applied effect?
3. Apply an additional effect
    a. Can you hear a noticeable difference?
    b. Is there anything unpleasant about the audible difference?
    c. Are you easily able to disable the applied effect?
4. Exit the application
    a. Can you confirm that the audio is no longer playing?

Summary of Test Results

Audio Throughput Results

The following table shows the results taken from 3 separate trials in which the effects algorithms were profiled to determine the amount of runtime which each effect, as well as a combinations of all effects took.

Table VIII.ii - Effect Throughput Results

Run/Effect	Digital Delay (seconds)	Tremolo (seconds)	Flanger (seconds)	Total Time (seconds)
1	0.001	0.001	0.006	0.008
2	0.002	0.001	0.005	0.008
3	0.001	0.001	0.006	0.008
Average (seconds)	0.0013	0.001	0.0057	0.008

Buffer Length Results

Due to how the Novocaine SDK hands off raw sample data, the buffer length is fixed at 2048 samples of stereo audio, or 1024 samples for each channel per callback. This results in 23.22 milliseconds of audio being delivered in each callback. However, due to the need for the circular buffer, it was found that storing 10 seconds worth of samples at a sampling rate of 44100 Hz (441000 floating point samples) resulted in providing the application with a sufficient memory component that didn't result in any noticeable increase in latency.

User Experience Results

In general, user acceptance testing resulted in generally positive comments with respect to the overall application experience. One users experience was as follows:

1. Launch the application
    a. Are you presented with a welcome message/direction on how to proceed with the application? ***No, there is no welcome message – the application is shown immediately***
    b. Is the visual layout aesthetically pleasing to you? Is it easy to see what you are doing? ***Yes – visually the application is easy to look at and understand***
2. Apply an effect
    a. Can you hear a noticeable difference? ***Yes***
    b. Is there anything unpleasant about the audible difference? ***None that I can note***
    c. Are you easily able to disable the applied effect? ***Yes***
3. Apply an additional effect
    a. Can you hear a noticeable difference? ***Yes***
    b. Is there anything unpleasant about the audible difference? ***None that I can note***
    c. Are you easily able to disable the applied effect? ***Yes***
4. Exit the application
    a. Can you confirm that the audio is no longer playing? ***Yes***

IX. Conclusion

The end result of the developed application met the desired specifications. The following offers a summary of the end project results vs. those of the initial project specifications.

Table IX.i - Summary of Project Specifications

Initial Specification	Final Project Implementation
Output Latency of no greater than 13ms	Output Latency of 8ms
Minimum 12bit Sampling Resolution	16 bit resolution provided by the devices ADC
Minimum 22kHz Sampling Frequency	44.1kHz Sampling Frequency
Maximum 10 second load time	0.5 second load time

While the project specifications were met, one of the initial proposals of this project was to implement the digital Chorus effect. Due to the time restrictions and development issues this effect was not implemented in the final version of the application.

Proposed Alternative Approaches, Future Improvements, Project Reflection

This application could greatly benefit from a User Interface redesign, due to time development difficulties the user interface was left in a wireframe stage. The current User Interface of the application provides functional levels of feedback to the user as to what they are controlling in the application. This project also could greatly benefit from effect algorithm optimization, some of the calls made in the code are extraneous and could be finessed in a manner that provides less latency to the overall audio throughput.

Future proposed improvements from the applications current state that would offer more value to the user are:

- User Interface Redesign
- Additional Effects
    - Particularly Pitch Shifting, Compression, Distortion/Overdrive, Equalization
- Ability to arrange effects signal flow
    - Ability to choose between parallel and series signal chains
- Effect Algorithm Optimization

This project proved to be quite challenging in a number of ways. First, it offered me the opportunity to develop my first mobile application, which meant it carried along the lessons and challenges of learning how to develop for a mobile platform. This project also tested my ability to take a design plan and fully see it through to the end. In the past, my experience with software engineering has been limited to a structured way that was enforced by an instructor. In the development of this project I had to take all the previous design practices I had been educated on and implement them in a way that allowed for the successful completion of this project. This project was an incredibly humbling experience and I am greatly looking forward to working on further revisions of this project.

X. Bibliography

[1] comScore, Inc, 'March 2015 U.S. Smartphone Subscriber Market Share', 2015. [Online]. Available: https://www.comscore.com/Insights/Market-Rankings/comScore-Reports-March-2015-US-Smartphone-Subscriber-Market-Share. [Accessed: 16- Apr- 2015].

[2] Acm.org, 'Software Engineering Code of Ethics and Professional Practice — Association for Computing Machinery', 2015. [Online]. Available: http://www.acm.org/about/se-code. [Accessed: 12- Nov- 2015].

[3] Apple, 'iOS - Apple', 2015. [Online]. Available: http://www.apple.com/ios/. [Accessed: 18- Apr- 2015].

[4] GitHub, 'Novocaine Audio', 2014. [Online]. Available: https://github.com/alexbw/novocaine/blob/master/README.md. [Accessed: 27- Sep- 2015].

[5] Developer.apple.com, 'Xcode - Apple Developer', 2015. [Online]. Available: https://developer.apple.com/xcode/. [Accessed: 28- Sep- 2015].

XI. Appendices

A. Analysis of Senior Project Design

Summary of Functional Requirements

The current version of this application allows the user to process a given input signal with several digital based effects. The following is a summary of the provided effects as well as their respective adjustable parameters:

- Digital Delay
  - Volume of the effect
  - Delay Time (up to 1 second)
  - Number of Repeats (up to 6 repeats)
  - Feedback
- Tremolo
  - Rate of the Oscillator (0.5 – 100 Hz)
- Flanger
  - Rate of the Oscillator (0.1 – 2 Hz)
  - Reach of the Flanger (0 – 10 ms)
  - Volume of the effect
- Reverb
  - Size of the room
  - Volume of the effect

All the effects mentioned above appear to function in a manner that would be considered real-time - that is to say there is very little to no noticeable latency between the input signal and processed output signal.

Primary Constraints

This project proved to be significant, both in terms of its implementation and in the experience and knowledge it provided me with. One of the notable challenges encountered was in the implementation and use of the circular buffer used to store the previous output/input samples. As it exists right now, the buffer is simply an array of floats that is written to and read from using an iterator that uses the modulo operator to “circulate” around the array. While this seems like a simple solution and a basic circular buffer, getting to this point proved to be quite the challenge. This was a scenario in which my design went through several iterations on the best way to handle the buffer, it ultimately resulted in the use of a fundamental data structure (the array) and the very useful modulo operator. I definitely was fighting the issue of trying to do too way more than was necessary to implement the buffer. This buffer was absolutely crucial in my implementation of the delay, flanger, and reverb effects.

Another notable difficulty encountered was in the implementation of the oscillator for the flanger effect. During my initial attempt at implementation I kept obtaining an aliased version of my input signal and a sinusoidal output! I realized my issue was that I was performing the calculations correctly for the

oscillator, however I wasn't accounting for the fact that I was running on a priority thread in the operating system, which meant that my callback functions were actually getting called at a rate of around 43Hz which was actually generating the sinusoid in my output! Once I accounted for this rate I was able to make the flanger work as desired!

Overall the most difficult aspect of this project, which turned out to be the most rewarding, was my lack of knowledge with the iOS development framework. This was really the first mobile application I've ever developed, and my first time programming in Objective-C . There was definitely a fairly steep learning curve to the application development framework. Ultimately this resulted in me having to truly learn by doing, which proved to be incredibly rewarding in the end.

Economic

The following is a summary of the economic factors incurred by this development of this project, included is the project's estimated cost, as well as the final bill of materials.

Table XI.i - Original Estimated Cost

	Item	Cost
1	**iOS Device (iPad Air 2)**	$450.00
2	**IK Multimedia iRig2**	$39.99
3	**Instrument Cable**	$29.99
Total Cost (including CA sales tax)	----------	$561.58

Table XI.ii - Final Bill of Materials

	Item	Cost
1	**iOS Device (iPad Air 2)**	$450.00
2	**MacBook (Development Device)**	$799.99
3	**Combined Microphone/Headset Cable**	$9.99
Total Cost (including CA sales tax)	----------	$1360.79

In terms of development time I initially estimated to have the project fully completed by the end of my second quarter. However, I ran into several issues during the alpha stage of development, which resulted in little beta stage and release stage development time. The following Gantt charts provide both the initial and actual time estimates for this project.

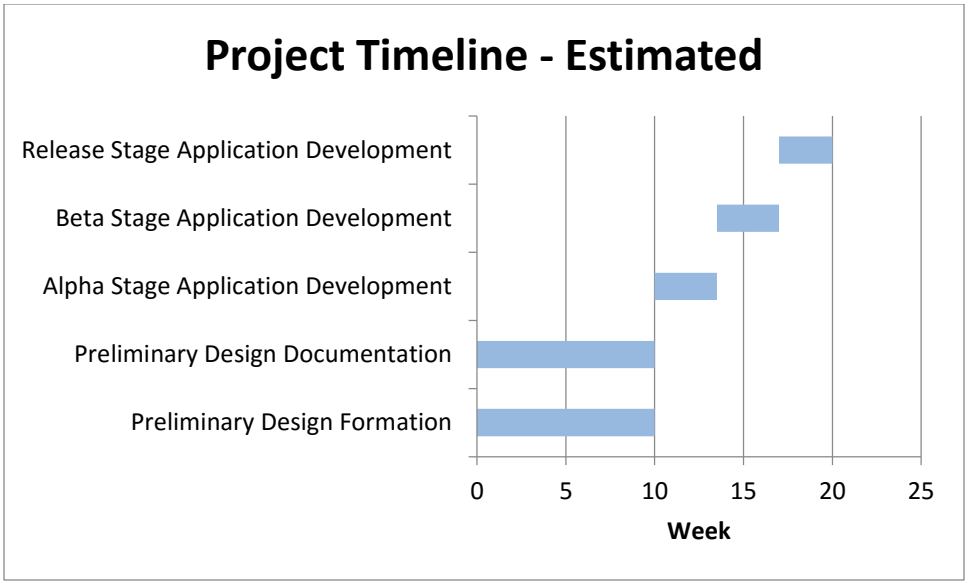

Figure XI.i - Estimated Project Timeline

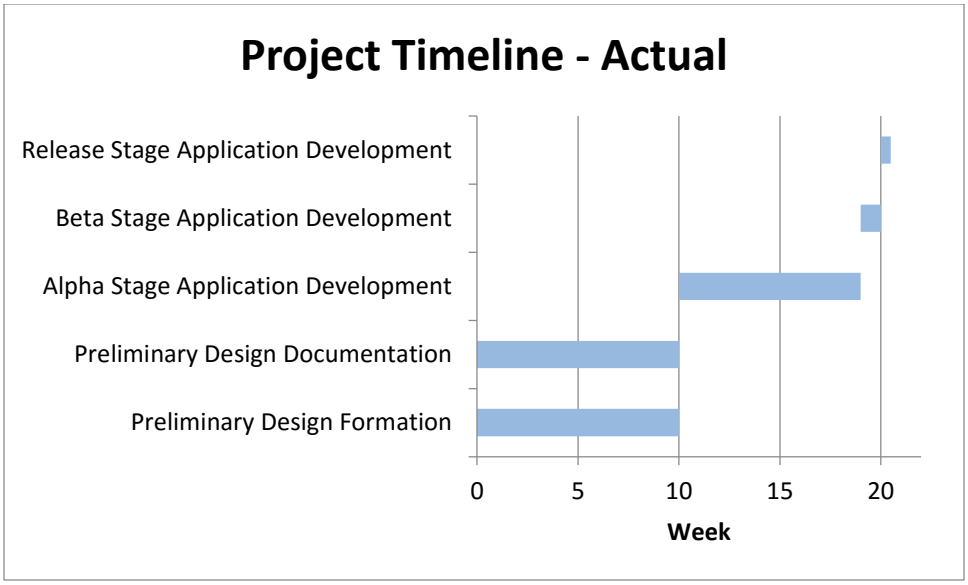

Figure XI.ii - Actual Project Timeline

Environmental

Environmental impacts from the use of this application exist via the energy use of the device as well as any materials used in conjunction with the application, such as instrument cables. The main environmental concern is in regards to the energy use of the device, which is actually fairly minimal.

Manufacturability

Since this project dealt with the development of a software application and had no produced hardware related components, there are no notable mentions in terms of the manufacturability of this project.

Sustainability

The main challenge for maintaining the application includes adhering to future iOS SDK standards, this will be necessary to ensure that the application is able to run on future iOS devices. In addition to adhering to future standards the application could also obtain performance gains in terms of the new hardware that will be released alongside future versions of the SDK. Improvement to this application can come in regards to adding additional effects, as well as optimizing the effects algorithms implemented.

Ethical

In terms of ethical implications, the design of this product adheres to the design standards outlined at: http://www.acm.org/about/se-code [2]

Additional note is given to credit that this software uses helper methods as produced by the Novocaine Audio Framework by Alex Wiltschko. Further information is available at: https://github.com/alexbw/novocaine/blob/master/README.md [4]

Developer documentation for Apple's iOS Development framework is available at: https://developer.apple.com/xcode/ [5]

Health and Safety

Any user of this application should ensure that the iOS devices output signal is at a safe level to prevent any potential damage to the user's ears.

Social and Political

The development of this application does not contain any political concerns or factors. In terms of social use, this application offers a means for users to create and share their musical ideas with one another. This application provides a means for users to further interact with music and to express themselves.

Development

Development of this application introduced me to multiple new tools and development techniques that I have never in the past encountered. Once of the greatest things I learned throughout the development of this project was in regards to the iOS development framework. This was the first mobile application I've ever developed which meant learning the development process as well as the development tools for iOS applications. I now have experience with XCode, which is the IDE used for developing iOS and OSX applications. In order to develop for iOS I had to become familiar with the Objective-C++ programming language, which before this application I had never programmed in. Overall I'm incredibly happy with the knowledge I've gained from creating this application.

B. Parts List and Costs

The following is a list of the items used for the development of this project. All of the following items were necessary items to ensure the successful development of this project.

Table XI.iii - Final Cost of Project

	Item	Cost
1	**iOS Device (iPad Air 2)**	$450.00
2	**MacBook (Development Device)**	$799.99
3	**Combined Microphone/Headset Cable**	$9.99
Total Cost (including CA sales tax)	----------	$1360.79

C. Project Schedule

Development of this application was implemented into the following 4 core stages:

- Preliminary Design - This stage dealt with preliminary and prerequisite research into the project – ie. Available technology, educational resources, user interface design etc.
- Alpha Stage Application Development – This stage dealt with the fundamental proceedings of the application development. This is the stage that was responsible for taking the application from the preliminary design phase into the beginning phases of software development.
- Beta Stage Application Development – This stage focused primarily on the refinement of the aesthetic elements of the application, as well as vigorous testing to thwart out any potential bugs.
- Release Stage Application Development - This stage aims to provide a fully functional feature rich and release ready application with all known bugs removed.

Milestones

In order to stay on track for project completion within the two quarter long span the following were the proposed milestone dates.

- Preliminary Design – Completed by end of Week 11 of the first quarter of development.
- Alpha Stage Completion – Completed by end of Week 4 of the second quarter of development.
- Beta Stage Completion – Completed by the end of Week 7 of the second quarter of development.
- Senior Project Report Rough Draft – Completed by the end of Week 8 of the second quarter of development.
- Release Stage Completion – Completed by the end of Week 11 of the second quarter of development.
- Senior Project Report Final Draft – Completed by the end of Week 11 of the second quarter of development.

For actual project timelines please see figures XI.i and XI.ii in the Analysis of Senior Project Design subsection of the appendices

D. Program Listing

The primary program used for the development of this application was Apple's IDE: Xcode 7. For further information on Xcode please visit the development website at: www.apple.com/xcode

E. Source Code

The following shows both the ViewController.h and ViewController.mm implementation files. Together these files define the logic and behavior of the entire application.

```
/* ViewController.h Senior Project Implementation
*/

#import <UIKit/UIKit.h>
#import "Novocaine.h"
#import "RingBuffer.h"

@interface ViewController : UIViewController

@property (nonatomic, strong) Novocaine *audioManager;

@end

/* ViewController.mm Senior Project Implementation
Student: Jonah Clinard
Advisor: Dr. Wayne Pilkington
Fall 2015

Version 1.0
*/

#import "ViewController.h"

@interface ViewController ()

@property (nonatomic, assign) RingBuffer *ringBuffer;
@property (weak, nonatomic) IBOutlet UISlider *volumeSlider;
@property (weak, nonatomic) IBOutlet UISlider *delayTime;
@property (weak, nonatomic) IBOutlet UILabel *volumeLabel;
@property (weak, nonatomic) IBOutlet UILabel *delayTimeLabel;
@property (weak, nonatomic) IBOutlet UISwitch *delayToggle;
@property (weak, nonatomic) IBOutlet UISwitch *tremoloToggle;
@property (weak, nonatomic) IBOutlet UISlider *repeatSlider;
@property (weak, nonatomic) IBOutlet UISlider *tremoloSlider;
@property (weak, nonatomic) IBOutlet UILabel *repeatLabel;
@property (weak, nonatomic) IBOutlet UILabel *tremoloRateLabel;
@property (weak, nonatomic) IBOutlet UILabel *gainLabel;
@property (weak, nonatomic) IBOutlet UISlider *gainSlider;
@property (weak, nonatomic) IBOutlet UISlider *reverbSlider;
@property (weak, nonatomic) IBOutlet UISlider *reverbVolumeSlider;
@property (weak, nonatomic) IBOutlet UISwitch *feedbackToggle;
@property (weak, nonatomic) IBOutlet UISlider *mixFeedbackSlider;
@property (weak, nonatomic) IBOutlet UISlider *flangerReach;
@property (weak, nonatomic) IBOutlet UISlider *flangerRate;
@property (weak, nonatomic) IBOutlet UILabel *flangerRateLabel;
@property (weak, nonatomic) IBOutlet UILabel *flangerReachLabel;
@property (weak, nonatomic) IBOutlet UISwitch *flangerToggle;
@property (weak, nonatomic) IBOutlet UISlider *flangerVolumeSlider;
@property (weak, nonatomic) IBOutlet UILabel *flangerVolumeLabel;

@end

@implementation ViewController

float gain = 1;
float repeatVolume = 1;
float reverbVolume = 1;
```

```
float feedbackVolume = 0;
float echoDelay = 22050;
float tremoloRate = 500;
bool delayOn = false;
bool tremoloOn = false;
bool reverbOn = false;
bool feedbackOn = false;
bool flangerOn = false;
int delayRepeat = 1;
int reverbSize = 0;
UInt32 flangerCount = 0;
UInt32 place = 0;
UInt32 i = 0;


- (void)dealloc
{
   delete self.ringBuffer;
}

- (void)viewDidLoad
{
   [super viewDidLoad];
}

- (void)viewDidUnload
{
   [super viewDidUnload];
}

// Handle UI Elements
- (IBAction)repeatChanged:(id)sender {
   delayRepeat = self.repeatSlider.value;
   self.repeatLabel.text = [NSString stringWithFormat:@"%i", delayRepeat];
}

- (IBAction)volumeChanged:(id)sender {
   repeatVolume = self.volumeSlider.value;
   self.volumeLabel.text = [NSString stringWithFormat:@"%.1f",
self.volumeSlider.value];
}
- (IBAction)delayTimeChanged:(id)sender {
   echoDelay = self.delayTime.value*44100;
   self.delayTimeLabel.text = [NSString stringWithFormat:@"%f", self.delayTime.value];
}
- (IBAction)delayToggleTapped:(id)sender {
   i = 0;
   self.ringBuffer->Clear();

   if (self.delayToggle.on) {
       delayOn = 1;
   }
   else{
       delayOn = 0;
   }
}
- (IBAction)tremoloToggleTapped:(id)sender {
   if(self.tremoloToggle.on){
       tremoloOn = true;
   }
   else {
       tremoloOn = false;
   }
```

```
}
- (IBAction)tremoloSliderChanged:(id)sender {
    tremoloRate = self.tremoloSlider.value*1000;
    self.tremoloRateLabel.text = [NSString
stringWithFormat:@"%f",self.tremoloSlider.value];
}
- (IBAction)gainSliderChanged:(id)sender {
    gain = self.gainSlider.value;
    self.gainLabel.text = [NSString stringWithFormat:@"%f",gain];
}
- (IBAction)reverSliderChanged:(id)sender {
    reverbSize = round(self.reverbSlider.value);
}
- (IBAction)feedbackToggleTapped:(id)sender {
    if (self.feedbackToggle.on){
        feedbackOn = true;
    }
    else{
        feedbackOn = false;
    }
}
- (IBAction)mixFeedbackSliderChanged:(id)sender {
    feedbackVolume = self.mixFeedbackSlider.value;
}
- (IBAction)flangerRateSliderChanged:(id)sender {
    self.flangerRateLabel.text = [NSString
stringWithFormat:@"%f",self.flangerRate.value];
}
- (IBAction)flangerToggleTapped:(id)sender {
    if (self.flangerToggle.on){
        flangerOn = true;
    }
    else{
        flangerOn = false;
    }
}
- (IBAction)flangerReachSliderChanged:(id)sender {
    self.flangerReachLabel.text = [NSString
stringWithFormat:@"%f",self.flangerReach.value];
}
- (IBAction)flangerVolumeSliderChanged:(id)sender {
    self.flangerVolumeLabel.text = [NSString
stringWithFormat:@"%f",self.flangerVolumeSlider.value];
}

- (void)viewWillAppear:(BOOL)animated
{
    [super viewWillAppear:animated];
    __weak ViewController * wself = self;
    self.audioManager = [Novocaine audioManager];
    self.ringBuffer = new RingBuffer(44100*100, 2);
    float *holdingBuffer = (float *)calloc(44100*20, sizeof(float));
    float *holder = (float *)calloc(44100*100, sizeof(float));
    float *processedData = (float *)calloc(44100*30, sizeof(float));
    __block float phase = 0.0;
    __block float samplingRate = wself.audioManager.samplingRate;
    __block float tremoloOsc = 0.0;
    __block float reachBack = 0;
    __block float flangerOsc = 0;
    __block int bufferLength = 0;
    __block UInt32 lastPlace = 0;
    __block UInt32 delayFetch = 0;
    __block UInt32 flangerBack = 0;
```

```
    __block UInt32 numSamples = 0;

    [self.audioManager setInputBlock:^(float *data, UInt32 numFrames, UInt32
numChannels) { // Callback to receive audio
        for (int i = 0; i<numChannels*numFrames;i+=numChannels){ //copy samples over to
send mono out
            for (int j=i; j<i+numChannels;++j){
                data[j] = data[i];
            }
        }
        wself.ringBuffer->AddNewInterleavedFloatData(data, numFrames, numChannels);
    }];

    [self.audioManager setOutputBlock:^(float *outData, UInt32 numFrames, UInt32
numChannels) { //Callback to output audio
        // Grab our playthrough audio
        wself.ringBuffer->FetchInterleavedData(outData, numFrames, numChannels);
        numSamples = numFrames*numChannels;
        bufferLength = numSamples*430; //Buffer to hold up to 10 seconds worth of prior
samples

        if (place == bufferLength) { // Handle circulation of buffer
            lastPlace = place;
            place = 0;
        }

        for (int j = 0; j<numSamples; j++) { //Copy current output samples into our
buffer
            outData[j] *= gain; //apply input volume gain
            holder[place+j] = outData[j];
        }

        place += numSamples; // adjust our offset for samples stored

        // Check if Our Delay is enabled

        if (delayOn) {
            i = 0;
            while (++i <= delayRepeat) {
                reachBack = (int)round(2*echoDelay*i);
                if ((int)reachBack%2)
                    reachBack -= 1;
                if (place >= (UInt32)reachBack+numSamples){
                    memcpy(holdingBuffer,holder+place -(UInt32)reachBack-
numSamples,sizeof(float)*numChannels*numFrames);
                }
                else { //Last written index is at the end of our buffer, or the first
stages of the buffer
                    if (place == numSamples){ //we started over, so grab from the end
of the buffer
                       memcpy(holdingBuffer,holder+lastPlace+place -(UInt32)reachBack-
numSamples,sizeof(float)*numSamples);
                    }
                    else {// we need to copy both data from the end of the buffer and
beginning
                        for (delayFetch = 0; delayFetch < numSamples;delayFetch++){
                            holdingBuffer[delayFetch] = holder[(lastPlace+place-
numSamples-(UInt32)reachBack+delayFetch)%bufferLength];

                        }
                    }
                }
                repeatVolume /= .8*i; // Decrease volume of each repeat
```

```
                // Multiply the samples and copy them to our current output
                vDSP_vsmul(holdingBuffer, 1, &repeatVolume, holdingBuffer, 1,
numSamples);
                vDSP_vadd(holdingBuffer, 1, outData, 1, outData, 1, numSamples);
            }
            repeatVolume = self.volumeSlider.value; // Reset volume
        }

        if (feedbackOn){ //Handle Delay Feedback logic
            vDSP_vsmul(outData,1,&feedbackVolume, holdingBuffer,1,numSamples);
            vDSP_vadd(holdingBuffer, 1, holder+place-numSamples, 1, holder+place-
numSamples, 1, numSamples);
        }

        if (tremoloOn) { //Handle tremolo logic
            for (int i=0; i < numSamples; i++)
            {
                tremoloOsc = phase * M_PI * 2 * tremoloRate/samplingRate; //Generate
iteration oscilation
                outData[i] *= sin(tremoloOsc) + 1; //Amplitude Modulate
                phase+=tremoloRate/samplingRate;
                if (tremoloOsc>512){ //Count to numSamples/4 to deal with callback
oscillation
                    phase = -512;
                }
            }
        }

        if (flangerOn){ //Handle Flanger Logic
            for (int i=0; i<numSamples; i++) {
                flangerOsc =
fabsf(sin(2*M_PI*(flangerCount++/self.flangerRate.value)/samplingRate)); // Generate
iteration oscillation
                flangerBack =
(UInt32)ceil(flangerOsc*self.flangerReach.value*samplingRate);

                if (((flangerBack)%2)){ //align - only grab valid data
                    flangerBack -=1;
                }

                if (place == numSamples){ //grab from the end of the holding buffer
                    outData[i] += self.flangerVolumeSlider.value*holder[(lastPlace + i
- flangerBack)%bufferLength];

                }
                else if ((int)(place -numSamples +i - flangerBack) >= 0){ //grab from
"front of the buffer"
                    outData[i] += self.flangerVolumeSlider.value*holder[place -
numSamples + i - flangerBack];
                }
                if (flangerCount/self.flangerRate.value >= samplingRate) { //Reset
flangerCount
                    flangerCount = 0;
                }
            }
        }

        //Copy samples over to both channels
        for (int i = 0; i<numSamples;i+=numChannels){
            for (int j=i; j<i+numChannels;++j){
                outData[j] = outData[i];
            }
```

```
        }

        //Copy current output samples into our processed buffer
        for (int j = 0; j<numSamples; j++) {
            processedData[place-numSamples+j] = outData[j];
        }

        if (1) { //Handle Reverb always enable
            i = 10;
            while (i <= reverbSize) {
                reachBack = 512*i;
                if ((int)reachBack%2) {
                    reachBack -= 1;
                }
                if (place >= (UInt32)reachBack+numSamples) {
                    memcpy(holdingBuffer,processedData+place -(UInt32)reachBack-
numSamples,sizeof(float)*numSamples);
                }
                else { //Last written index is at the end of our buffer, or the first
stages of the buffer
                    if (place == numSamples){ //we started over, so grab from the end
of the buffer
                        memcpy(holdingBuffer,processedData+lastPlace+place -
(UInt32)reachBack-numSamples,sizeof(float)*numSamples);
                    }
                    else { // we need to copy both data from the end of the buffer and
beginning
                        for (UInt32 reverbFetch = 0; reverbFetch <
numSamples;reverbFetch++){
                            holdingBuffer[reverbFetch] =
processedData[(lastPlace+place-numSamples-
(UInt32)reachBack+reverbFetch)%bufferLength];
                        }
                    }
                }
                reverbVolume /= (1/self.reverbVolumeSlider.value)*(i/8);
                vDSP_vsmul(holdingBuffer, 1, &reverbVolume, holdingBuffer, 1,
numSamples);
                vDSP_vadd(holdingBuffer, 1, outData, 1, outData, 1, numSamples);
                i += 10;
            }
            reverbVolume = self.reverbVolumeSlider.value;
        }
    }];

    //Begin audio Playback
    [self.audioManager play];
}

-
(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientat
ion
{
    if ([[UIDevice currentDevice] userInterfaceIdiom] == UIUserInterfaceIdiomPhone) {
        return (interfaceOrientation != UIInterfaceOrientationPortraitUpsideDown);
    } else {
        return YES;
    }
}

@end
```