Senior Project Report

Delay-Based Digital Audio Effects Module for DJs and Musicians

Submitted to Dr. David Goodman

Engineering Technology Department

By

Alexander Perr

12/1/2018

Abstract

This module can manipulate audio signals, in real time, by modulating them in the time domain. This device is nicknamed “The Time Machine”. This device is meant for DJs and musicians who wish to be able to change characteristics about the music that they are playing during a live performance. This device allows the user to be able to change the playback speed of a song, such as slowing the song down and speeding it up. It allows the user to reverse the song. This device allows the user to perform momentary loops at various lengths for a stutter effect. This device can even change up the rhythm of a song by rearranging parts within a sequence. This device also lets the user perform vinyl effects, like what DJ’s do. This device lets the user be able to remix music on the fly and in real time.

The device has a touch screen user interface and a set of 12 hardware push buttons. The user is able program each of the buttons using a touch screen interface. The user selects which button they want to program with the touch screen interface. They can then select from a list of 25 different delay-based audio effects to program to the button. This gives the user full customizability of the layout on the control board. The user can load all 12 buttons with different effects in any arrangement they choose. The user plugs their audio device into the input of the module with a 3.5 mm jack. They then plug in a speaker or headphones. There is a Tap Tempo on the GUI used to synchronize the effects to the tempo of any song. After the song is synchronized, the user can then perform combinations of delay-based audio effects to remix any song.

The origins of this device came from a project for the Multi-Disciplinary Undergraduate Research Institution at IUPUI. I led a team of 3 other researchers where we were tasked with developing a digital audio effects module. During the research, we evaluated several different types of digital signal processors and devices to house our touch screen graphical user interface which was used in this project. I expanded on this project to add more functionality and customizability to the device.

The inspiration for this project came from a VST (Virtual Studio Technology) plugin called Gross Beat, from Image Line.

TABLE OF CONTENT

Abstract…………………………………………………………………2
Table of Contents……………………………………………………….3
Revision History………………………………………………………...4
Introduction……………………………………………………………….5
Referenced Documents…………………………………………………6
System Wide Design Decisions…………………………………………..7
System Architectural Design……………………………………………..9
Conclusions……………………………………………………………16
Appendix………………………………………………………………17

REVISION HISTORY

Version	Date	Revised by	Description
1.0	11/6/2018	Alexander Perr	Initial version
2.0	11/15/18	Alexander Perr	Add text
3.0	11/20/18	Alexander Perr	Add Conclusion
4.0	11/28/18	Alexander Perr	Add Appendix
5.0	12/1/18	Alexander Perr	Final

Introduction

Problem Statement

DJs and musicians are becoming more and more reliant on technology in modern day musical performances. Artists need a device that will allow them to be able to manipulate audio signals in new and creative ways. Most people turn to hardware controllers that are required to be used in combination with software on a laptop. The problem with this is that personal computers can be unreliable. Laptop computers have so many other programs running in the background that can hinder the performance of the application. Personal computers are also very prone to bugs and viruses that can completely disrupt a musical performance. Sometimes the only thing a person can do when this happens is to try to restart their computer. When they do this, computers can take up to 5 to 10 minutes in order to fully boot up again. When you have a whole crowd of people wondering where the music went, 5 minutes is now an eternity. They need something that is reliable and easy to use that can allow them to have powerful creative control over their music. I have developed a standalone device that can be used by DJs and musicians who want to be able manipulate their music without the need for a laptop and allows the user to have powerful creative control by modulating audio signals in the time-domain.

System Overview

This device allows the user to be able to change the playback speed of a song, such as slowing the song down and speeding it up. It allows the user to reverse the song. This device allows the user to perform momentary loops at various lengths for a stutter effect. This device can change up the rhythm of a song by rearranging parts within a sequence. This device also lets the user perform vinyl effects, like what DJ’s do on turntables.

The device has a touch screen user interface and a set of 12 hardware push buttons. The user is able program each of the buttons using a touch screen interface. The user selects which button they want to program with the touch screen interface. They can then select from a list of 25 different delay-based audio effects to program to the button. This gives the user full customizability of the layout on the control board. The user can load all 12 buttons with different effects in any arrangement they choose. The user plugs their audio device into the input of the module with a 3.5 mm jack. They then plug in a speaker or headphones. There is a Tap Tempo on the GUI used to synchronize the effects to the tempo of any song. After the song is synchronized, the user can then perform combinations of delay-based audio effects to remix any song.

Referenced Documents

Title
Real Time Digital Signal Processing from MATLAB to C with the TMS320C6x DSPs, Third Edition: Welch, Wright, and Morrow 2017
http://users.ece.utexas.edu/~bevans/courses/realtime/index.html - University of Texas, **Prof. Brian L. Evans**
OMAP-L130 Datasheet from Texas Instruments
TM320C6748 Datasheet from Texas Instruments
CS470xx Datasheet from Cirrus Logic
ADSP-BF706 Datasheet from Analog Devices
ADAU1761 Datasheet from Analog Devices

System-Wide Design Decisions

The hardware selection process was done with the team that I worked with during the MURI project. This effect module utilizes a rolling buffer that is allocated in RAM memory. Most delay-based audio effects need to have some form of RAM into order to access recorded signals. It was important in hardware selection process that our DSP chip would be able to be supplied with enough RAM to give us the desired delay effect. In order to get audio signal information into the buffer, an audio codec was needed. The audio codec converts analog signals from the music source into digital signals and digital signals back to analog signals. The audio codec also gives us the platform for our 3.5mm audio in and out jacks. Our audio codec needed to have an AD/DA conversion resolution of at least 16-bits to give us the desired audio quality. Once we figured out our desired specifications, we selected several DSP evaluation kits from various manufacturers. We chose kits from Texas Instruments, Analog Devices, and Cirrus Logic. We tried to develop on each of the boards. We had trouble trying to find helpful information on how to develop on most of the boards. We also selected several micro-controller boards in order to develop our user interface on. We chose to evaluate the Arduino Mega, Freescale FRDM-KL125Z, and a Raspberry Pi 3 Model B. We needed a micro-controller that would allow us to establish serial connection to our DSP chip in order to control the DSP program and to provide us with the environment to make out touch screen GUI. We listed off all of critical specifications in order to make a decision on hardware by utilizing a decision matrix for both the DSP and the micro-controller.

	Analog Devices-ADAU1761	**Analog Devices - ADSP-BF706**	**TI - TMS320C6748**	**TI-OMAPL138**		
RAM	41 kB	1.136 MB SRAM	256KB Unified Mapped RAM and External 16-Bit SDRAM With 128-MB Address Space	256KB Unified Mapped RAM and External 16-Bit SDRAM With 128-MB Address Space + 8kb from ARM		
Power	1.8v - 3.3v	5v	5v	5v		
AD/DA Resolution	24 bit	16 - 32 bit	8 - 32 bit	8-32 bit		
Processor	Sigma DSP	Blackfin+, ADAU1761 DSP	TI C6748	TI - C6748 + ARM9 Core		
Processing Speed	12 MHz	400 MHz	375 - 465 MHz	375 to 465 Mhz		
GPIO/Serial	4 pins	Yes and Arduino support	9 banks of 16 pins	9 banks of 16 pins		
Audio IN/OUT Device	3.5 mm Jack	3.5 mm Jack	3.5 mm Jack	3.5 mm jack		
Programming Environment	Sigma Studio,Drag and Drop	CrossCore,C + Audio Weaver, Drag and Drop	Code Composer, C	Code Composer, C		
Measured Latency	40 samples			39 samples		
Maximum Possible Buffer Size	4096 samples (~0.1 seconds at 48kHz sampling rate)			16,777,216 samples (~5.8 minutes at 48kHz sampling rate		

Digital Signal Processor Selection Matrix

	Freescale-FRDM-kl25z	Arduino Mega	Raspberry Pi 3 Model B
RAM	128KB flash, 16KB SRAM	8KB SRAM	1GB, LPDDR2
Power	3.3-5v	5V	5V
Processor	KL25Z128VLK4-Cortex-M0+ MCU	Atmel ATmega V-2560	4× ARM Cortex-A53
Processing Speed	48MHz	16 MHz	1.2GHz
GPIO/Serial	Yes	54 I/0s + 16 analog in	USB UART
Programming Environment	Kinetis, Mbed	Arduino Sketch	Python

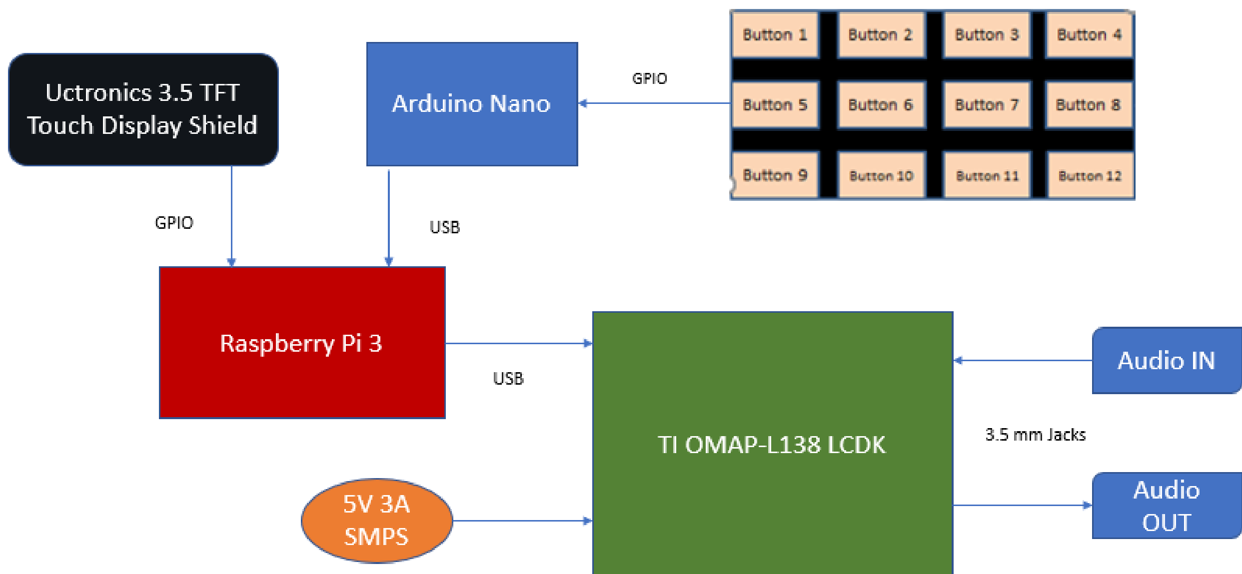
Micro-Controller Selection Matrix

The DSP chip that we went with is the TI OMAP-L138. This chip is an ARM9 processor + C6748 DSP. The ARM allows for more flexible control of the C6748 DSP, such as using a Linux operating system. We developed on the OMAP-L138 Low Cost Development Kit (LCDK). The LCDK comes equipped with all of the necessary hard ware connectors that we need for our device, such as 3.5mm audio jacks, USB UART connectors, and serial connectors. The LCDK comes equipped with a 128 MB of SDRAM, which allows us to have quite a large buffer size. We were able to create a max buffer size of 16,777,216 samples, which gave us a delay of 350 seconds. The LCDK is equipped with an audio codec that has a variable AD/DA conversion resolution of 8-32 bits. Code was developed in C programming language on Texas Instrument's Code Composer IDE.

The micro-controller that we went with to use as our user interface is the Raspberry Pi 3 Model B. A touch screen shield for the Pi is used for the user interface. The GUI was developed on the Raspberry Pi using Python programming. The Raspberry pi is equipped with HDMI ports, USB Hosts ports, serial connectors, and GPIO. We used the USB host port on the Raspberry Pi to communicate and send commands to the OMAP-L138 LCDK. We originally developed the GUI on the Arduino Mega; however, we were not able to successfully establish a serial connection to the OMAP. We also learned that the Arduino would not be able to process the commands to the DSP very fast. It would not have been ideal for our audio application as the system needs to be fast responding for good operation. The Raspberry Pi has a processor speed of 1.2 GHz compared to the Arduino Mega, which has a 16 MHz processing speed. In my design, I decided to use an Arduino Nano to handle the hardware push button user interface.

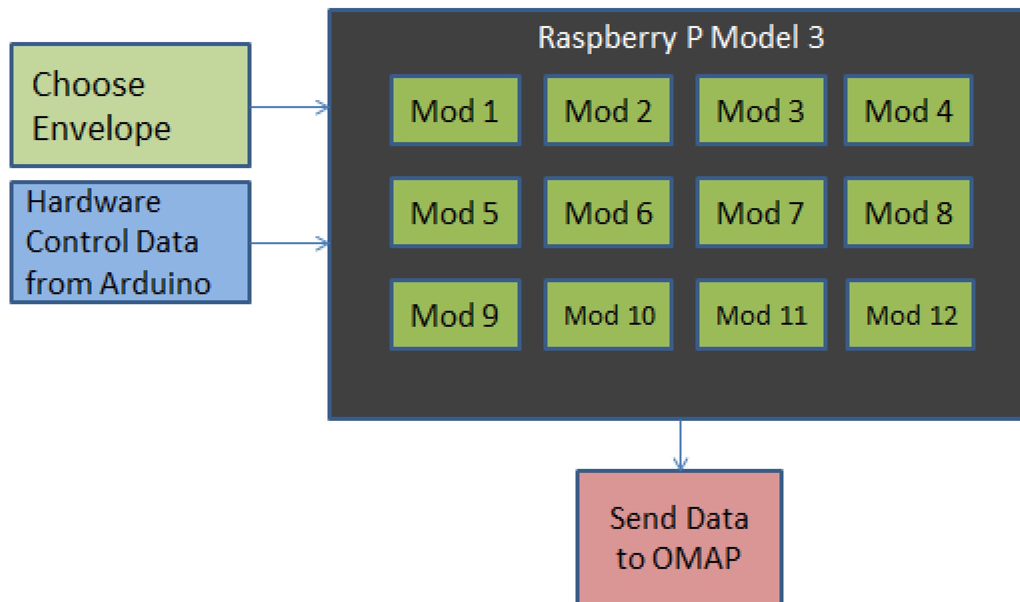
SYSTEM ARCHITECTURAL DESIGN

SYSTEM COMPONENTS



A Raspberry Pi 3 served as a central hub for the devices to communicate as it has 4 available USB Host drives. The Raspberry Pi 3, Arduino Nano, and the TI OMAP-L138 LCDK communicate over serial with each other using UART USB protocol. An Uctronics 3.5 Inch TFT Touch Screen shield was placed on top of the Raspberry Pi 3 and used for the touch screen GUI, connected via GPIO and HDMI. The Arduino Nano takes care of all the GPIO and logic for the physical buttons. The Arduino Nano sends data over to the Raspberry PI 3 when a push-button is pressed, over USB. Once the Raspberry Pi 3 reads the data from Arduino, it sends commands to the TI OMAP-L138 LCDK over USB. The TI OMAP-L138 LCDK contains a Digital Signal Processor, SDRAM, and an audio codec all in one package. The OMAP-L138 LCDK is powered by a 5V 3A power supply. The OMAP-L138 LCDK powers the Raspberry Pi 3 via USB host connect from the LCDK to the Raspberry Pi 3 with a USB micro.

CONCEPT OF EXECUTION



The functional architecture is shown above. The user uses the touch screen to program the 12 hardware push buttons. There are 12 buttons on the touch screen GUI that correspond with the 12 hardware push buttons in order. The user selects a button they want to program on the touch screen, where it will then prompt the user to select an effect. The user selects an effect, such as ½ speed, reverse, 2X speed, and many others. The selected effect is then programmed to the selected push-button.

The touch screen GUI code was created using object-oriented programming techniques, written in Python on the Raspberry Pi 3. This program reads incoming data from the Arduino Nano, and relays data to the OMAP-L138. The Arduino has 12 buttons attached to it, as well as pull up resistors for each of the buttons. The Arduino is programmed to send over different ASCII characters via serial. Each pushbutton sends a different ASCII character when pressed. The Raspberry Pi program has a set of 12 different functions that get called based on the ASCII characters sent from the Arduino. Each function represents a pushbutton. These functions contain a unique global variable that can be changed at any time that correspond with a pushbutton. By default, the variables are empty strings. These values are changed when the user selects a button they want to program and makes an effect selection. When the user makes an effect selection on the touch screen GUI, it changes the value of the global variable inside a button function to a particular ASCII character. Each effect corresponds with a unique ASCII character, such as “#”, “$”, or letters. When a user presses a button, the ASCII character programmed to a global variable are sent over to the OMAP-l138. The OMAP will then read this character which activates the function within the DSP module to carry out the process that creates the desired audio effect.

The OMAP-L138 interprets the ASCII characters as a decimal value. The program checks to see if a particular decimal value is sent over. When a value that corresponds with a particular effect is sent, it carries out the function that particular effect. The functions on the OMAP tell the system to play back data stored in RAM memory, much like how delay effects work, but at different locations as time moves forward. By using difference equations, the system can perform these effects by modulating the numerical value of the playback index used to access the data in the rolling buffer in RAM sequentially. The Tap Tempo is used to change the size of the rolling buffer and to change the variable “oneBeat”. The Tap Tempo function is essentially a sample counter that starts and stops a counter after the Tap Tempo button is pressed on the touch screen GUI. This allows the user to synchronize the module with any song. The Tempo Shift Up and Down buttons simply shifts the sample size in “oneBeat” up or down for added precision. Unit Step functions are used to perform momentary loops and to switch up the beat. Ramp functions were used to speed up, slow down, and reverse the song. Exponential functions are used to create vinyl effects. Examples of the C code are shown on the following pages.

```
137 //Time Machine Functions
138 //*************************
139 int Step(int passTime)
140 {
141     if(passTime < 0)
142     {
143         return 0;
144     }
145     else if(passTime >= 0)
146     {
147         return oneBeat;
148     }
149 }
150
```

Step Function

```
151 int Ramp(int passTime, int passTime2, double passSlope)
152 {
153     int slopeProd = passSlope*passTime2;
154
155     if(passTime >= 0)
156     {
157         return slopeProd;
158     }
159     else if(passTime < 0)
160     {
161         return 0;
162     }
163 }
```

Ramp Function

```
165 int Spin(int passTime, int passTime2, int passAmp, double passDecay)
166 {
167
168     int spinProd = passAmp*exp(passDecay*passTime2);
169
170
171     //printf("%d\n", spinProd);
172
173     if(passTime >= 0)
174     {
175         return spinProd;
176     }
177     else if(passTime < 0)
178     {
179         return 0;
180     }
181 }
```

Exponential Function

The functions shown are used in a global variable called “goBack”, which is used to change the value of the index used to access the rolling buffer in SDRAM sequentially. The different effects are created from different variations of this variable. Example of this code is shown below.

```
if(serialStore == 49)                                           //1 BEAT REPEAT
{
    goBack = 0-Step((time-oneBeat))-Step((time-2*oneBeat))-Step((time-3*oneBeat));
    //printf("%d\n", time);

}

else if(serialPass == 51)                                    //1/3 Beat REPEAT
{
    goBack = 0-(0.3333)*Step((time-(0.3333)*oneBeat))-(0.3333)*Step((time-(0.6666)*oneBeat))-(0.3333)*Step((time-(1)*oneBeat))
            -(0.3333)*Step((time-(1.3333)*oneBeat))-(0.3333)*Step((time-(1.6666)*oneBeat))-(0.3333)*Step((time-(2)*oneBeat))
            -(0.3333)*Step((time-(2.3333)*oneBeat))-(0.3333)*Step((time-(2.6666)*oneBeat))-(0.3333)*Step((time-(3)*oneBeat))
            -(0.3333)*Step((time-(3.3333)*oneBeat))-(0.3333)*Step((time-(3.6666)*oneBeat))-(0.3333)*Step((time-(4)*oneBeat));
}

else if(serialPass == 53)                                       //Reverse
{
    goBack = Ramp(time, time, -2);
}

else if(serialPass == 54)                                       //1/2 Speed
{
    goBack = Ramp(time, time, -0.5);
}

else if(serialPass == 55)                                       //2X Speed
{
    goBack = Ramp(time, (time-4*oneBeat), 1);
}
```

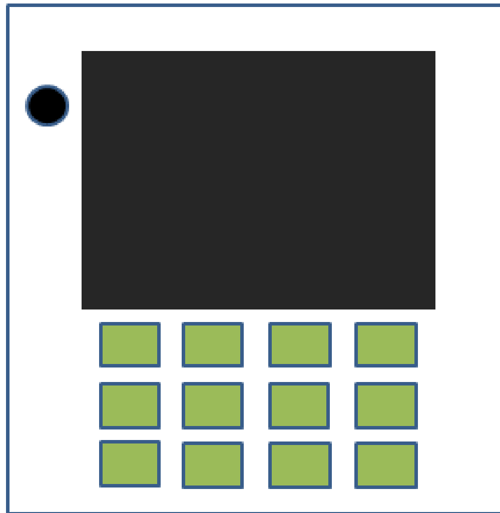
```
else if(serialPass == 59)                                   //Stutter 1
{
   goBack = Ramp((time-1.5*oneBeat),(time-1.5*oneBeat),-2) + Ramp((time-2*oneBeat),(time-1.5*oneBeat),2)-(0.5)*Step((time-2.5*oneBeat))
            +(0.5)*Step((time-3*oneBeat))-(0.5)*Step((time-3.5*oneBeat));
}

else if(serialPass == 68)                                   //Complex
{
    goBack = -(0.5)*Step((time-0.5*OB))+(0.5)*Step((time-0.75*OB))-(0.75)*Step((time-1.25*OB))-(1.25)*Step((time-2*OB))
             +(0.5)*Step((time-2.25*OB))+Step((time-2.5*OB))-(2.25)*Step((time-2.75*OB))+(0.75)*Step((time-3*OB))
             -(1.5)*Step((time-3.5*OB));
}

else if(serialPass == 57)                                   //BackSpin Start
{
   goBack = -4*oneBeat + Spin(time, time, 4*oneBeat, -0.00005);
}

else if(serialPass == 58)                                   //Vinyl Off
{
   goBack = oneBeat + Spin(time, time, -oneBeat, 0.00001);

}
```

The variable “goBack” changes over time using these functions. The variable is then used in the heart of the program, that plays back the data stored in the rolling buffer. The code is shown below

```
322 //Time Machine
323 //************************
324 void timeMachine()
325 {
326     if(goBack == 0)
327     {
328         dryPlayback();
329     }
330     else if(goBack < 0)
331     {
332     //output buffer audio
333         fluxCapacitor = recIndex + goBack;
334
335         if(fluxCapacitor < 0)
336         {
337             fluxCapacitor = fluxCapacitor + bufferEnd;
338             // printf("funk you");
339         }
340
341         CodecDataOut.Channel[LEFT] = buffer[LEFT][fluxCapacitor];
342         CodecDataOut.Channel[RIGHT] = buffer[RIGHT][fluxCapacitor];
343      }
344
345 }//end Time Machine
```

The variable called “fluxCapacitor” is the index used to access the rolling buffer, for a good reason. It drives the “Time Travel”. The data stored in the buffer is then sent to the audio codec to be processed into an analog signal.

INTERFACE DESIGN



Tempo Select

Pad 1

Figure 1: Hardware Interface Design

Figure 2: Actual Interface

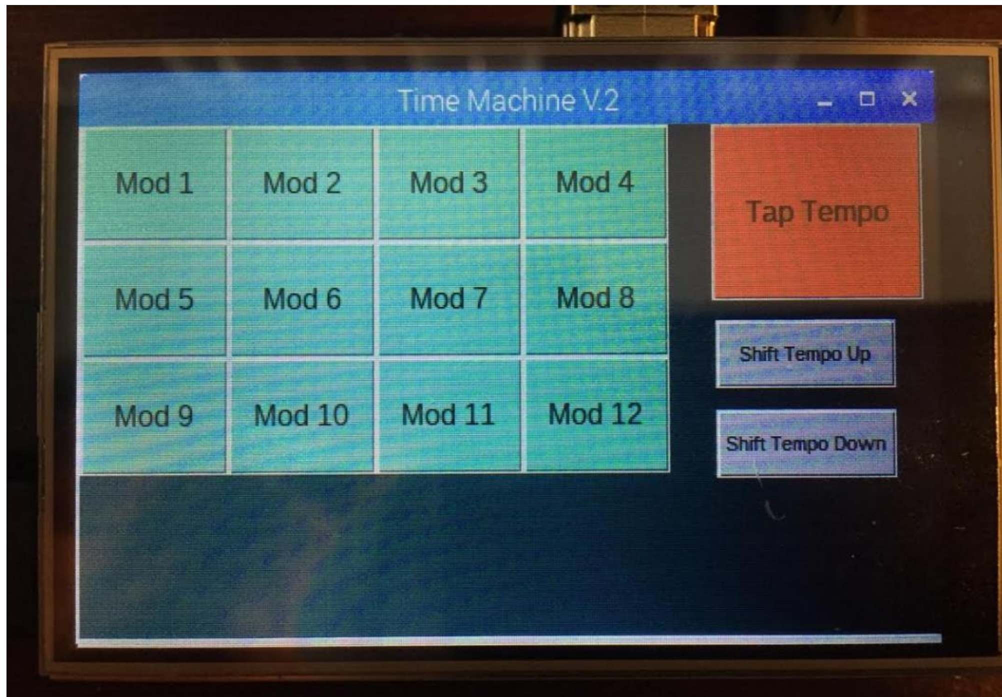


Figure 3: Main Menu

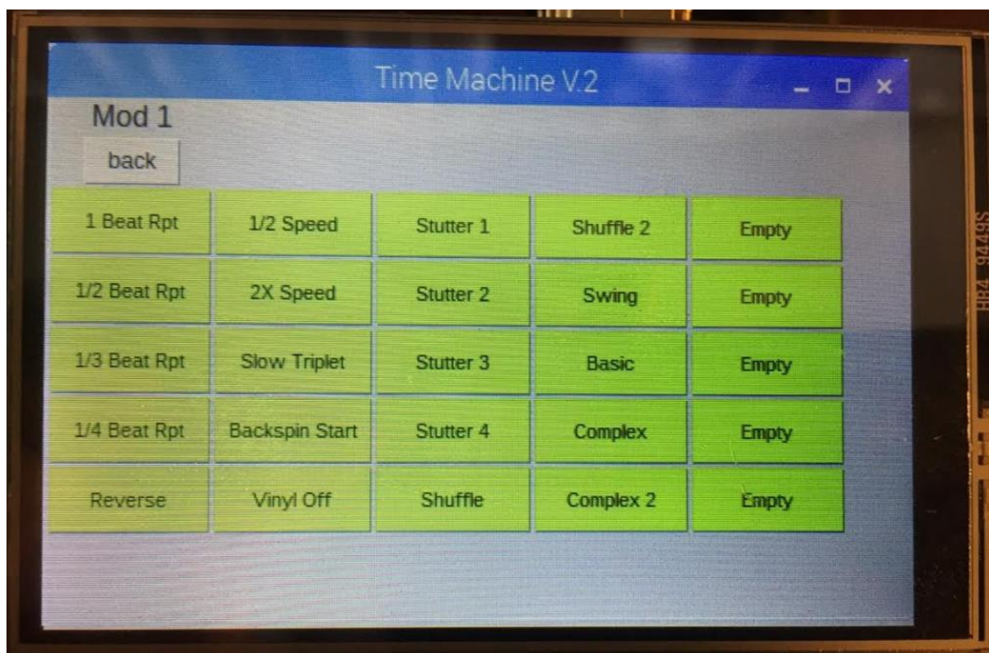


Figure 4: Effect Selection Menu for MOD 1

The Raspberry PI 3 houses the touch screen, receives information from the Arduino Nano, and sends data to the TI OMAP-L138 LCDK. There is a set of 12 physical hardware buttons, as shown in Figure 2. On the touch screen there are 12 touch buttons, labeled Mod 1, Mod 2, etc. Each of these “Mods” correspond with a physical hardware button. There user selects which button they want to program by clicking the corresponding mod. When they do

this, it brings them to the effect selection menu, shown in Figure 4. The user then selects an effect. When they select an effect, the button gets programmed to carry out that function. This also brings them back to the main menu. I have replaced the tempo knob idea with a simple tempo shift up or tempo shift down. This is way more effective and quicker than using a tempo know. This helps the user get synchronized a lot quicker.

USER SETUP AND OPERATION

There are several steps in order to operate this machine as of right now.

- Plug 5 Volt 3A power supply into the TI OMAP-L138 LCDK
- Plug USB Type B Micro from the OMAP LCDK HOST USB port to the Raspberry Pi 3 Power
- Plug USB Micro from bottom left HOST USB on Raspberry Pi to Arduino Nano
- Plug USB Micro from Bottom Right HOST USB on Raspberry PI to OMAP LCDK
- Plug in music source with AUX cord to input
- Plug in speaker to output
- Play Song
- Synchronize Time Machine to BPM of the song using Tap Tempo
- Select button to program on the touch screen GUI
- Select Effect
- Push and hold down push button to activate effect

Conclusion/Recommendations

This report has talked about the scope of my senior project and what I have achieved during the project. This paper talked about the problem statement and what my plan was to fix this problem. This paper talked about the functionality of the Time Machine. This paper talked about the System Wide Design Decisions, the System Architectural Design, as well as the Interface Design of my project. This paper talked about set up procedure of the project.

I think that this project could be improved greatly. I needed to figure flashed on the Raspberry Pi and on the OMAP-L138 LCDK that would automatically load the program as soon as the device was turned on. In its current state, I have to debug the program from my laptop onto OMAP every time I want to use it. However, I can unplug the debugger after it has been loaded on, making it a standalone device, without a laptop. Another point of improvement would be to change the labels on the touch screen GUI button modules to let the user know which effects are loaded on each of the push buttons.

I originally had much bigger ideas for this project. The hardware used for this project has a lot of potential. The amount of RAM on the OMAP LCDK could store a lot of sound. I wanted to add a sound sampler and sequencer capability to this device. I wanted the user to be able to make songs using sequenced samples using the push buttons, as well as have the Time Machine Functionality. There is a lot of room for exploration and research with the hardware selected.

Appendix

Gross Beat Functionality

Image Lines Gross Beat is the inspiration behind this device. I incorporated ideas from Gross Beat into the device, that made the Time Machine a powerful real time standalone device. Gross Beat is a delay-based audio effect virtual studio technology (VST) plugin that allows the user to temporarily change the rhythm, pitch, and playback speed of a song in real time by manipulating the signal within the time domain. It also lets the user add on custom volume gates as well. The ideas behind Gross Beat can be complicated. In this section, I am going to explain how Gross Beat works and how I incorporated the time manipulation idea of Gross Beat into the Time Machine. The VST GUI is shown below.

Figure 1

Gross beat works within a 2-bar rolling buffer. The x-axis represents time (going from left to right). The y-axis represents how much time goes backwards (going from top to bottom). The maximum length of the x-axis is 1 bar. There are 4 beats in each bar. The green vertical line that is parallel to the y-axis is the **playback locator**. The playback locator will scroll from left to right and will loop around to the beginning when it reaches the end. The playback locator essentially represents "Real-Time". The line that looks like green stairs is the **mapping envelope**. The playback locator will read the location of the mapping envelope in order to determine how far back in the 2-bar rolling buffer the track will go. The image shown above is a simple 1 beat repeat. It repeats the first beat of song data that goes into Gross Beat for 4 beats, as you can see graphically. The stairs are riding on a dark sloped line in Figure 1. This is called the "safety line". Placing the mapping envelope at any point on the safety line will access the location in the buffer where the function first started. This is the start of the 1 bar. The stairs

cause repeats, because it's going from -1, to -2, to -3. This line ends at -4 beats on the y-axis, the same length as the x-axis. The image shown in is Gross Beat performing a 2x speed increase. This makes the song seem like it's going 2x fast. This is working by starting at 1 bar behind what is played in real time and essentially "playing catch up" to music played in real time. However, this effect generates a 1/2 bar delay in 2x speed because the program is accessing information previously stored in the buffer and can only go fast until it hits the safety line.

Figure 2

Figure 3

Figure 4

We can change the pitch of the incoming signal with Gross Beat as well. It is just like the 2x speed but at smaller intervals. Gross Beat can switch up the rhythm and beat of a song by following an envelope like the one shown in Figure 3. Gross Beat has mapping envelopes in order to control the volume as well, as shown in Figure 4. This can give the user the ability to remove parts of a song or add volume-based effects such as side chaining. In my device, the user inputs BPM into the effect. This will judge how fast the position indicator will scroll from left to right as well as determine how large the buffer will be. This will sync the functionality to the speed of the song that is being played. The only downside of this is that the user will have to know the BPM at which the song is played. A tap tempo will be used in order to give the user an idea as to what BPM the song is at. In a real-time environment, I can make this a one shot or a press and hold effect when the user chooses an envelope. The user will press the desired effect to trigger the playback locator (starting from the left) for 4 beats only one time, instead of looping back. The instant the user presses the button; they can call upon data from up to 2 bars. As time goes on, this information gets replaced. Think of the 2-bar rolling buffer as always following behind the playback locator as it goes from left to right, with data continuously getting added and erased. If this function is activated instantaneously, the user can call on the buffer at any point in time.

The envelopes for both the Time and Volume are essentially difference functions over a period. I created difference functions in order to create each of these timing envelopes and then converted the step functions into C code. With the functions I was be able to create the steps, slopes, and curves that are shown in the effect. For Time, these functions are used to

access the different locations inside the memory buffer to play back the audio data stored there for a 4 bar period. The user uses buttons in order to select the mapping envelope of their choosing. They are essentially playing time.