Music Synthesizer Senior Project: Danalog

Report by: Vikrant Marathe

Other Group Members: Bryan Bellin, Evan Lew, Jordan Wong

Advisor: Dr. Wayne Pilkington

Spring 2017

Cal Poly Electrical Engineering

Table of Contents

List of Tables and Figures

Abstract

The Danalog is a 25 key portable digital music synthesizer that uses multiple synthesis methods and effects to generate sounds. Sound varieties included three synthesis methods including FM, subtractive, and sample-based, with up to eight adjustable parameters, at least four effects, including reverb, chorus, and flange, with five adjustable parameters, and at least two note polyphony, and a five band equalizer. The user would be able to adjust these effects using digital encoders and potentiometers and view the settings on two LCD screens.

The finals project was unable to meet the original design requirements. The FM synthesis method was primarily working in the end product. The synthesizer was built to produce two note polyphony. The LCD screens displayed the information about the synthesis method as the user plays.

I. Introduction

The purpose of this project was to create a portable, inexpensive digital music synthesizer for amateur musicians. The intended customer base consists of young, amateur musicians who don't have a big budget for a more expensive music synthesizer.

The market requirements for this product are as follows:

- The Danalog Synthesizer will be inexpensive at less than $200
- The design will be sleek and lightweight to promote portability
- Up to eight adjustable synthesis parameters
- Up to five adjustable effects parameters
- Five band equalizer

Our intended customer is an amateur musician seeking an inexpensive digital synthesizer to create a wide array of user-defined sounds.

Several other companies have their own digital synthesizers equipped with numerous features. The Danalog's main competitors would be the Yamaha Reface, Korg Minilogue, Roland Boutique, and Arturia MicroBrute. The lowest price of these is the Arturia MicrBrute at $299 - which the Danalog has beat by $100. The Danalog digital synthesizer is also smaller than the other competitor's options.

II. Product Design Engineering Requirements

<u>Functional and Feature Requirements</u>

- The Danalog Synthesizer will produce notes over a 2 octave range via Frequency Modulation Synthesis, with two note polyphony.

- The chassis will be made from lightweight plastic that is easy to carry and hold.
- All components, peripherals, and circuit boards are industry standard and well supported.
- The encoders, potentiometers, and switches will be strategically placed in a manner that follows the logical path of the signal from generation, to equalizing, to modulating.
- The processing will be split among two IC's: The ATmega2560 for peripheral information, and the TMS320C5535 for Digital Signal Processing.

Performance Specifications

- Low latency (<3ms delay) production of notes
- Instant visual feedback (<3ms) on pressed note
- Internal rechargeable battery of 5 hour life
- Able to run on 5V 500mA USB power
- Low noise audio outputs. 90dBc S/N with +4dBu max output

Level 0 Blackbox Diagram

User Input: All of the user inputs are translated into 8-bit signals, handled by the ATMega2560 microcontroller. This includes MIDI protocol, potentiometer positions, encoder rotation direction, switch and button positions (via mux), for a total of 30 bytes. All of them are traced to an Arduino Mega development board on a PCB where the ATMega chip resides.

5V Power Supply: The synthesizer is powered via 5V 500mA USB power or a 5 volt battery. There is a level shifter as well, because the Arduino Mega board runs on 5V while the TMS320C5535 runs on 3.3V.

MIDI Input: An optional external MIDI input is available too, which will override the bytes sent by the in-built keyboard.

User Interface

The user can control the type of synthesis (FM, Subtractive, Sample), and shift octaves on the keys using the rotary switches. With the encoders, the user can control the ADSR envelope of the audio wave, select the digital effects that will be utilized, and control their parameters. For example, for the reverb effect, the user may be able to control the delay time (10-200 samples) between each reverberation as well as the attenuation constant (0.1-0.99). The potentiometers are used to control the audio equalizer, by setting the gains (-12dB - 12dB) at specific frequencies to attenuate and boost certain frequency ranges. There are also potentiometers used to modulate a user-defined parameter, bend the master pitch, and control the master volume.

The two LCD screens provide visual feedback for the user. The left one lets the user know the type of synthesis and the ADSR envelope settings, and the gains of the equalizer. The right screen displays the type of audio effect in use along with its respective parameters and their settings.

IV. System Design - Functional Decomposition (Level 1)

The system can be broken down as shown in figure 4.1. The operation of the system can be generalized as:

1. The user interacts with the device
    a. Presses a key on keyboard
    b. Sends a MIDI event
    c. Changes a parameter on the front panel
2. The ATmega reads in information from the user
3. The C5535 requests an update on the status of the system
4. The ATmega responds with the latest information on key presses, parameter changes and MIDI information
5. The C5535 generates a waveform based on the state of the system
6. The sound is enjoyed by the listener

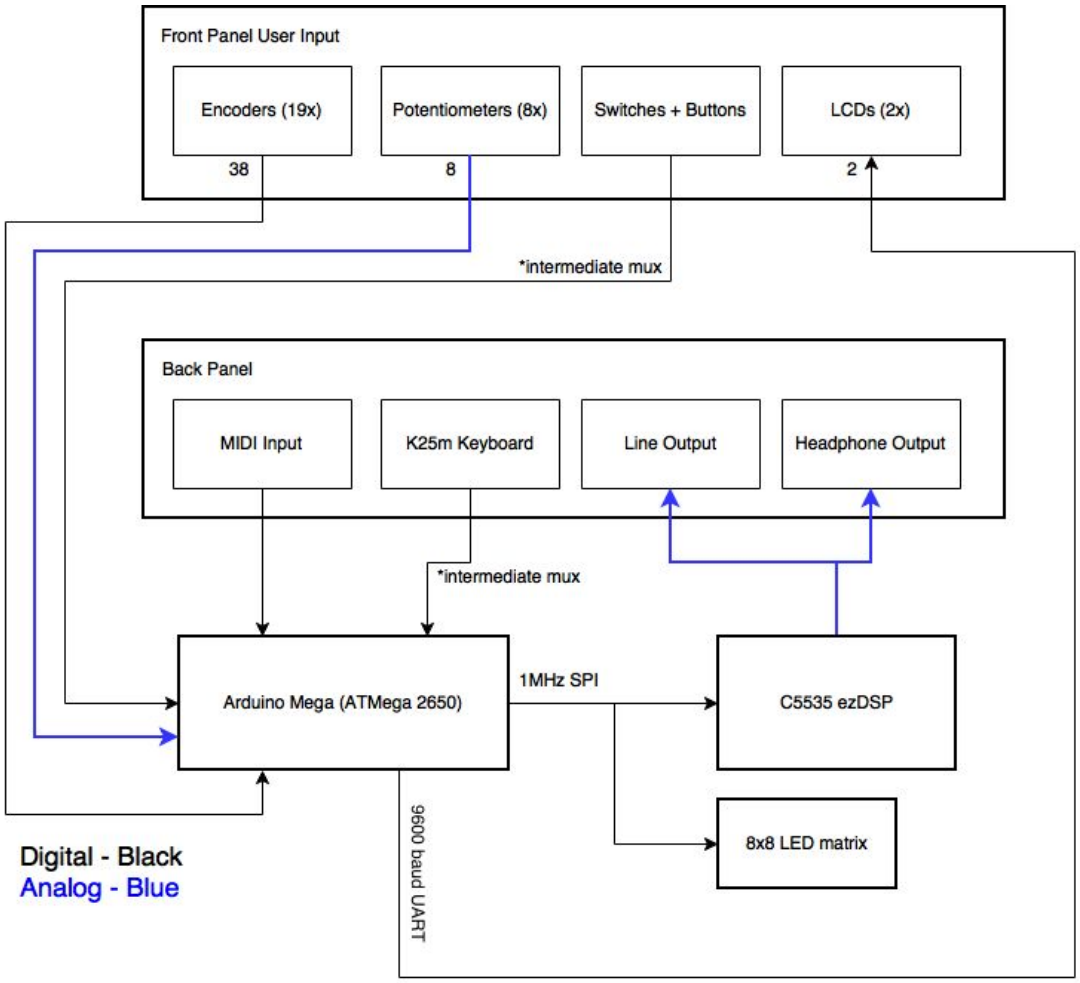

Figure 4.1: Level 1 block diagram of system

System Design Specification Partitioning or Performance Budgets (Vikrant)

16-bit Mux, Rotary Switches, Buttons

The 16 bit mux is used to keep track of the positions of all the switches and buttons, and store that information in a single byte. This includes the synthesis type switch, the octave select switch, and the preset selector buttons. The output pin of the mux is connected to a GPIO pin on the ATMega, which looks for an active low, and then the program will check which input was being selected on the input select pins to determine which switch it's looking at and what position that switch is in. The status was checked once every 10ms.

Potentiometers and ADC

The 8 potentiometers in the system are used to control 5 passbands in the equalizer, the master volume, the master pitch, and a user defined parameter. Each potentiometer was connected to the ATMega ADC and stored as an 8 bit value. These were checked around every 5 ms.

Serial Peripheral Interface (ATMega Side)

The SPI bus on the ATMega Side is configured in Slave Mode, and it sends the DSP chip information about all the peripherals (potentiometers, encoders, keys, etc.) in an interrupt service routine. To minimize latency, the DSP chip looks for certain information more often than other. For example, the status of the keys needs to be known almost at all times, so it is checked significantly more often than the switches for example.

V. Technology Choices and Design Approach Alternatives

16-bit Mux, Switches, and Buttons

Initially, we were considering sending information about the switch/button positions individually through the SPI. However, after finding a way to encode the information into a single byte, the data transfer became more efficient.

First, the mux was tested for basic functionality on a breadboard, by applying voltages to the input selectors, and seeing if the appropriate signal would appear at the mux output. This was verified with an oscilloscope.

Potentiometers and ADC

The potentiometers were fairly straightforward; only one design choice was selected, with the positions read by the ADCs on the ATMega chip.

The components themselves were tested for functionality using a simple ADC read program that read the position of the potentiometer and translated the value into a number ranging from 0 to 255.

Serial Peripheral Interface (ATMega)

We considered using a polling scheme for slave side of the SPI, but that would have required too much attention from the DSP chip and the ATMega, so we decided to enable interrupts on the SPI, so the ATMega could quickly check the status of the peripherals when requested, and send them to the DSP.

The SPI was tested using a logic analyzer. Dummy data was sent from the DSP chip and the SPI data register on the ATMega was checked to see if it received the same data. Given a particular piece of data from the DSP, the ATMega was programmed to send a particular byte, or array of bytes referencing some peripheral on the Danalog.

VI. Project Design Description

16-bit Mux, Switches, and Buttons

This subsystem is better described as the synth selector. Its purpose is to select the type of synthesis (FM, Subtractive, Sample), select the octave on the keyboard, and set and select a preset condition.

The mux selected was the CD74HC4067, a high speed CMOS logic 16 channel analog multiplexer. The schematic is shown on the next page in Figure 7.1, with the mux connected to 2 rotary switches, 3 buttons, and the 5 GPIO pins on the Arduino ATMega development board. The 3 position switch SW202 is used to select the type of synthesis, the 5 position switch SW201 is used to select the octaves, and the buttons set and select preset parameters. The 4 synth_mux pins are driven by the ATMega to select the mux input, which is sent to a different pin on the ATMega via the synth_mux_out pin.

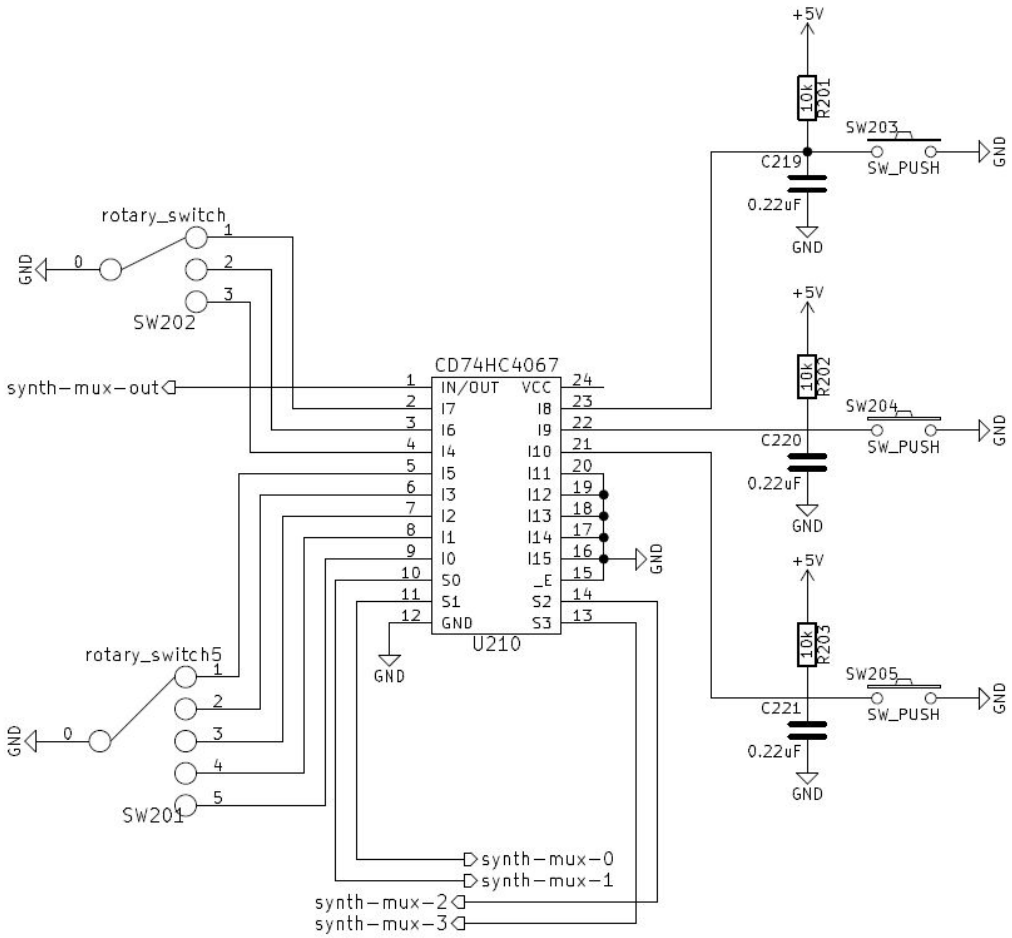

Figure 6.1 - 16 bit Mux, Rotary Switches, and Push Buttons Schematic

In software, a loop is generated to cycle through inputs 0 through 10 on the mux; inputs 11 through 15 are unused. Since the synth_mux pins are connected to various different ports on the ATMega, a clever bit masking and shifting technique is used to easily iterate through the input selections. As shown in the circuit, any input that is actually picked by the user will be grounded, which means this circuit is active low. The ATMega GPIO pin connected to synth_mux_out is internally connected to a pull-up resistor, waiting to be driven low. Once it is driven low, the software will check to see which input is currently being selected, and store that information accordingly. The information is stored in a single byte after all the inputs have been iterated through. The format is XXXYYZZZ. The Xs are for the SW01 position, the Y's are for the SW02 position, and the Z's are for the buttons, SW03, SW04, SW05.

Potentiometers and ADC

This subsystem is simply 8 potentiometers: 7 rotary potentiometers and 1 linear, connected to the ATMega ADC. These are meant to control the gains of various bandwidths in the equalizer, as well as the volume and pitch of the audio. The schematic is shown below for one of them.

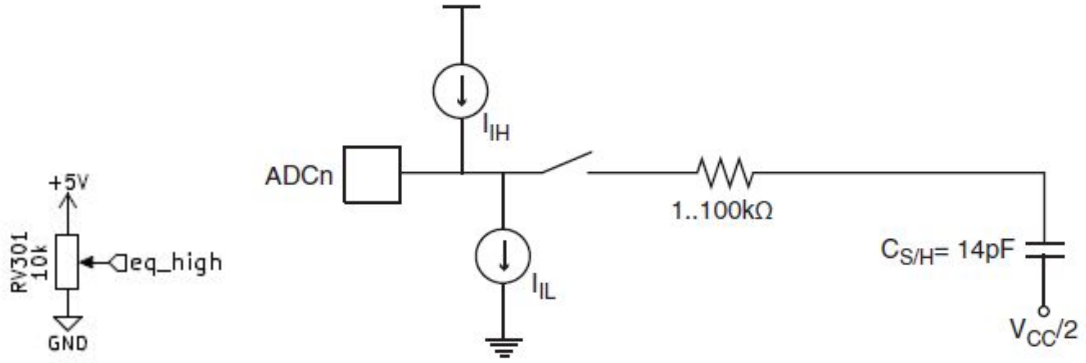

Figure 6.2 - ADC and Potentiometer Schematic

The potentiometers, as expected, are essentially variable resistors, with a maximum resistance of 10kΩ. It controls the input voltage to the ADC pin with values ranging from 0-5V, which is stored in the 14pF capacitor via sample and hold. These values are then encoded as 8-bit numbers for each potentiometer, for a total of 8 bytes. The sampling rate was made as quick as possible at ½ of the crystal oscillator frequency, 8 MHz, in order to minimize the latency of changing equalizer gains in real time.

Serial Peripheral Interface Slave Side (ATMega)

The SPI bus acts as the communication link between the ATMega and the TMS320C5535 DSP. The ATMega side is configured as a slave, and awaits instruction from the DSP to send information about the keys, switches, potentiometers, and encoders. The Level 2 block diagram is shown on the next page.

As shown in Figure 7.3, data from the DSP chip is clocked in from the MOSI line to a receive buffer on the ATMega, one bit per cycle. Once the shift cycles are complete, the data is written to the 8 bit shift register. This data is interpreted in software as an instruction from the master, and the shift register content is replaced with a byte containing information about a peripheral, which is finally sent back to the DSP via the MISO line. In some cases, multiple bytes of information need to be sent after receiving a single instruction.

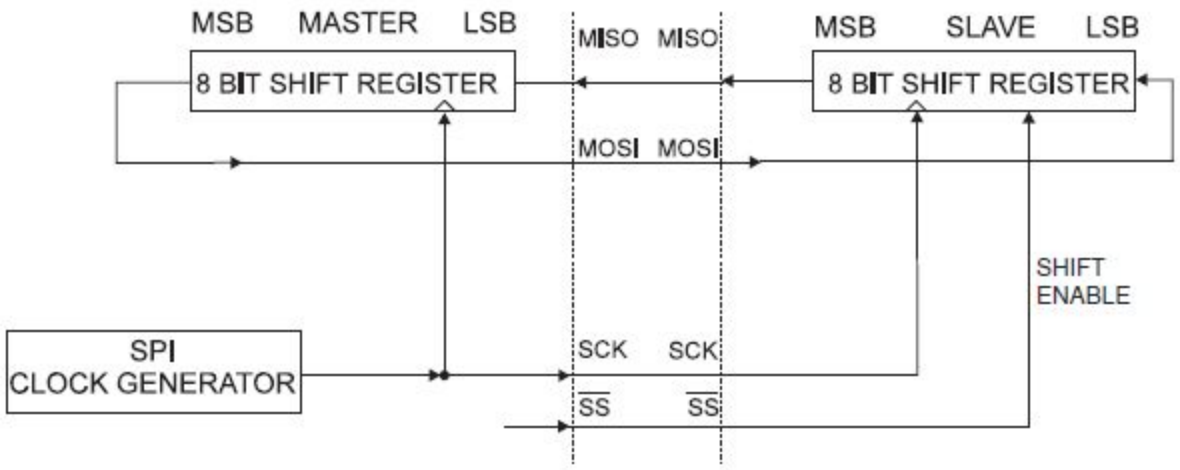

Figure 6.3 - Serial Peripheral Interface Level 2 Block Diagram

The SPI is initialized by setting it in slave mode, and enabling interrupts. Once a command is sent from the master, an interrupt service routine is initiated. If information about the switches is requested, this is easily sent back in a single byte. However, if any other information is needed, a counter needs to be set for the number of bytes to be sent. For the encoders, it is 19, and for the potentiometers it is 8. The counter is decremented after each byte is sent. Once the count reaches 0, the ISR is exited.

VII. Physical Construction and Integration

Physically the Danalog synthesizer consists of

1. Main PCB: The PCB connects all the devices together and functions mechanically to hold all the components neatly in place inside the enclosure.

2. Arduino Mega: The Arudino Mega functions to interface with all the user input controls. It communicates all the fundamental information to the C5535 via a SPI communication bus

3. TI ezDSP C5535: This device is responsible for interpreting the information sent by the Arduino and generating sound.

4. 3D printed chassis: Encloses all components

The PCB functions as the harness for all the front panel interface controls, which consist of rotary quadrature encoders, rotary potentiometers, linear potentiometers, and rotary switches. The organization of the user interface was decided by the team during the initial planning phase. All interconnections on the PCB were made to accommodate the initial user interface design.

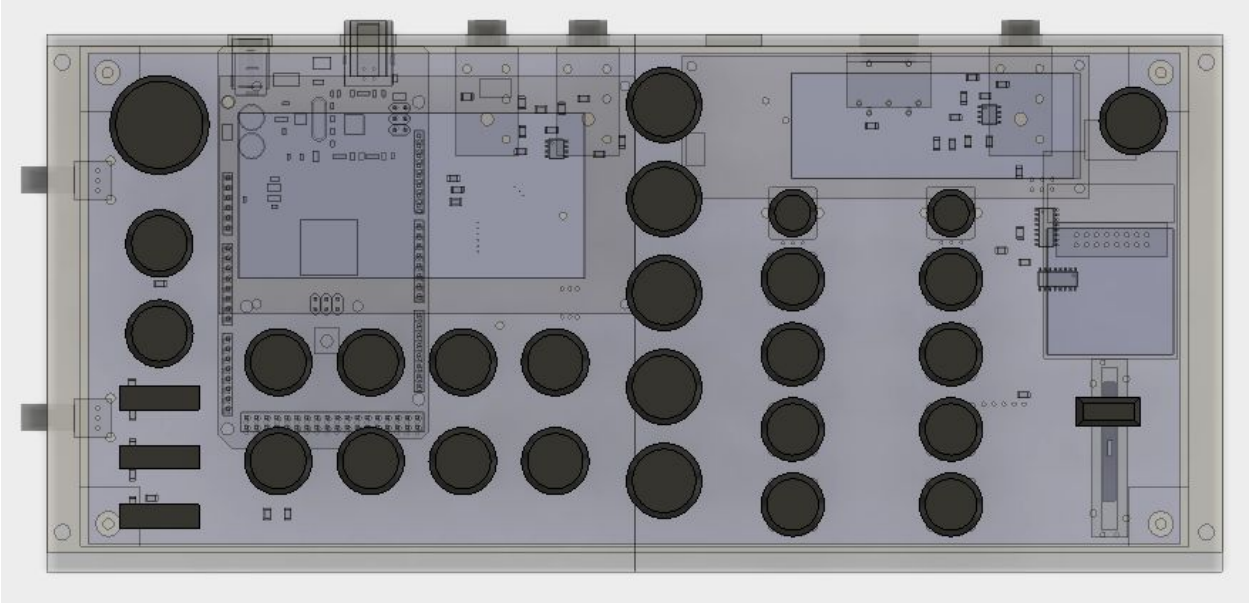

Figure 7.1: Overview of internal device construction and organization

Each device is routed to pins on the Arduino Mega board. Since the amount of IO needed was slightly more than the Arduino Mega provided we used multiplexers between the diode connected matrix keyboard (figure 7.2a) and between the rotary switches and front panel buttons (figure 7.2b)

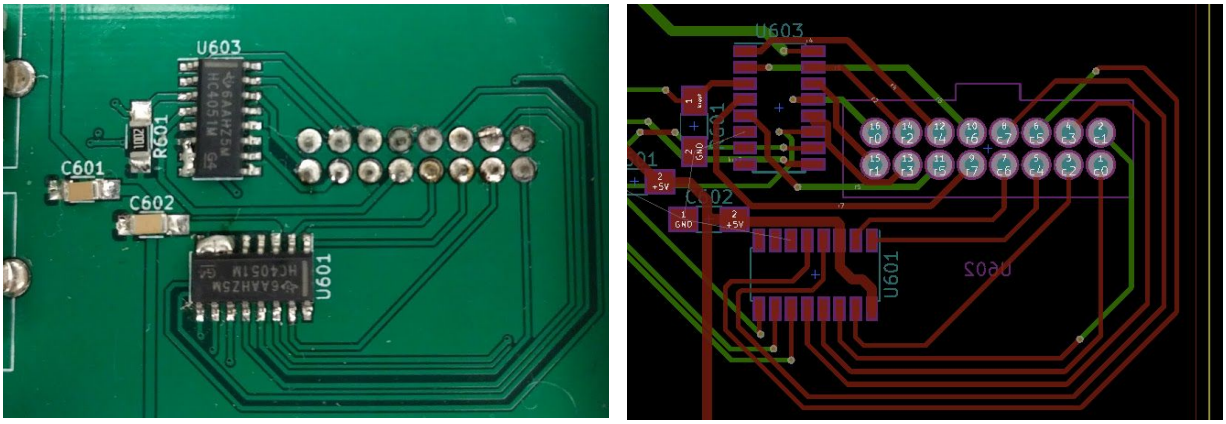

Figure 7.2a: Diode connected keyboard multiplexer layout

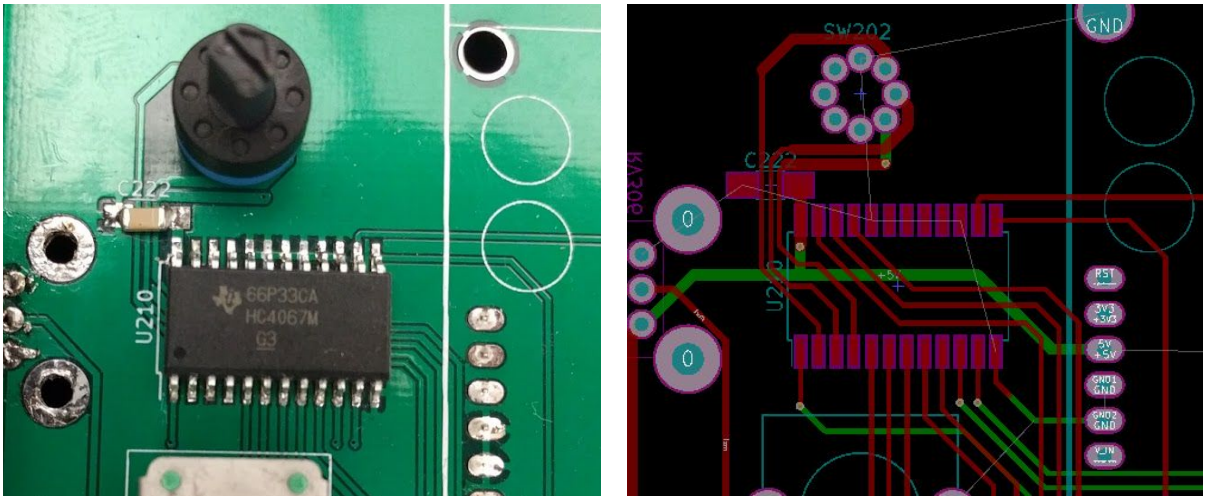

Figure 7.2b: Rotary switches and buttons multiplexer layout

The device also has two displays for outputting information about the state of the synthesis engine and the state of the effects processor. The displays were purchased from sparkfun as separate units not soldered to the the main circuit board. These displays were used because of their simple serial interface which allowed us to use a hardware UART to communicate with the display

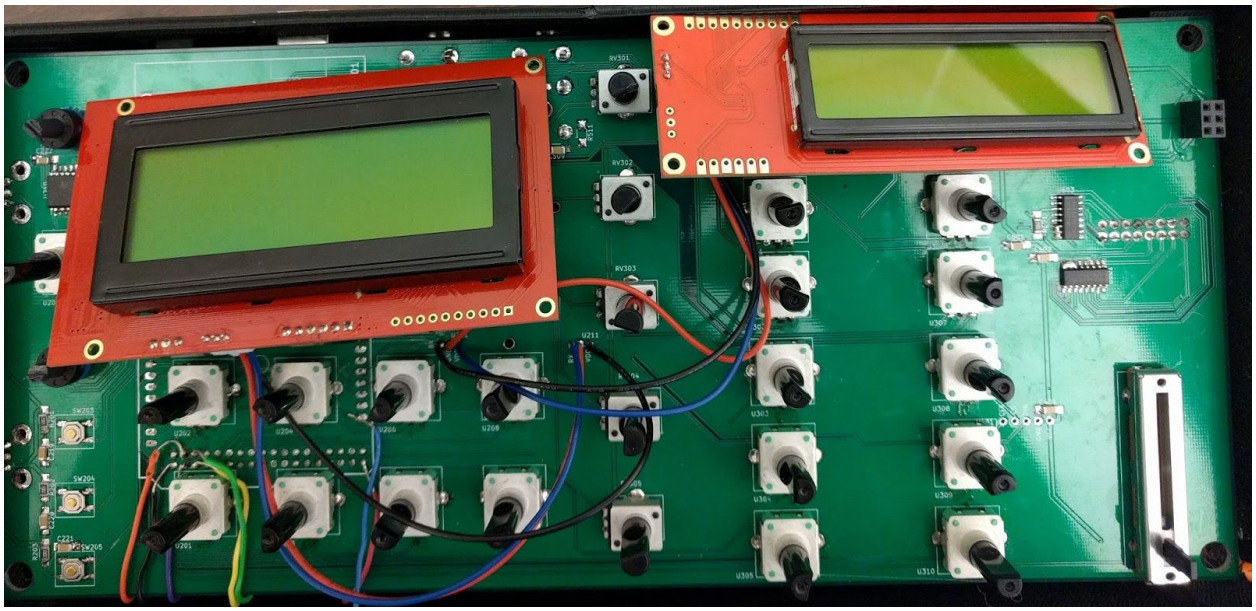

Figure 7.3: Both LCD displays connected to the main PCB via wires

Since the both the Arduino Mega and TI ezDSP both can be driven by 5 volts USB power there was no need to design any sort of power system. Additionally since the devices are low power, as USB devices usually are, there is no need for any form of heat sinking inside of the enclosure.

The chassis was 3D printed on Evan Lew's home 3D printer. Due to sizing constraints the chassis was printed in two halves and then glued together to form the final chassis. Figure 7.4 shows the 3D model of the chassis and figure 7.5 shows the real life chassis supported by the keyboard.

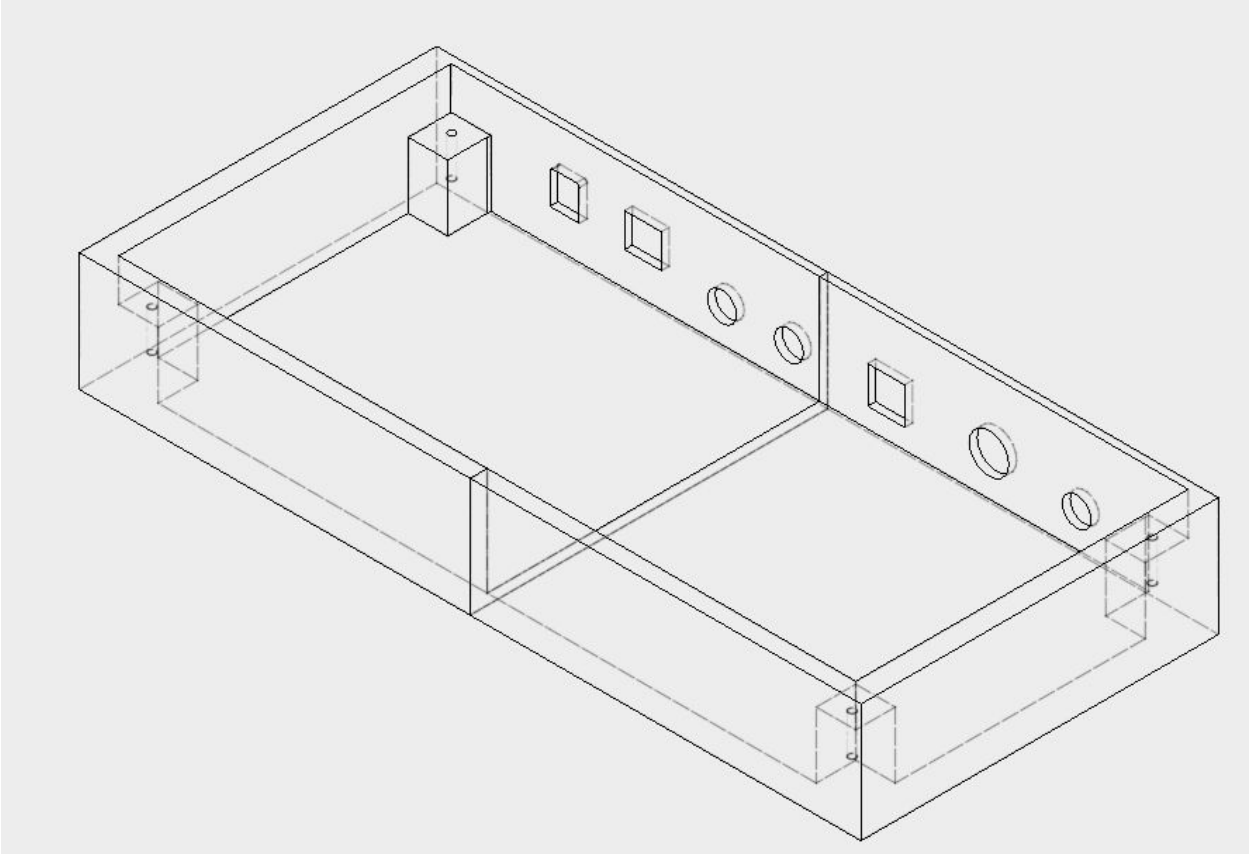

Figure 7.4: 3D model of the chassis

Figure 7.5: Chassis with internal hardware

VIII. Integrated System Tests and Results Bryan

Due to the ambitious and complex nature of the project we were not able to achieve all of our goals. However, we were able to have a product at the end that was on the path to achievement. At the end of our spring quarter, we had a functioning piece of hardware and functional synthesis engine.

The FM synthesis works. The latency is tested by having the arduino send a pin high when a key is pressed and measuring the delay between that transition and the start of the note being played.

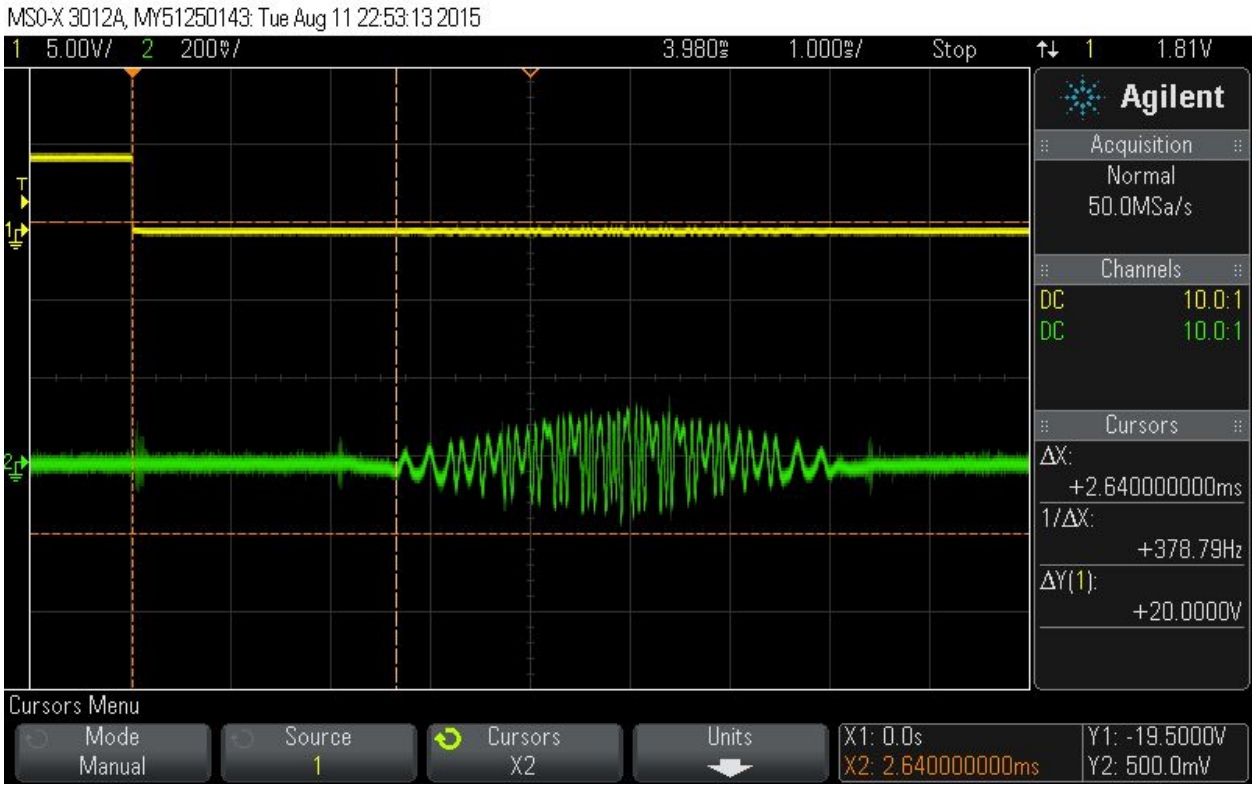

Figure 8.1: FM synthesis minimum latency test.

It should be noted that latency varied. We found the minimum latency to be 2.64 milliseconds while maximum extended to 5.36 milliseconds. This meets our specifications as it is not noticed

by the human ear.

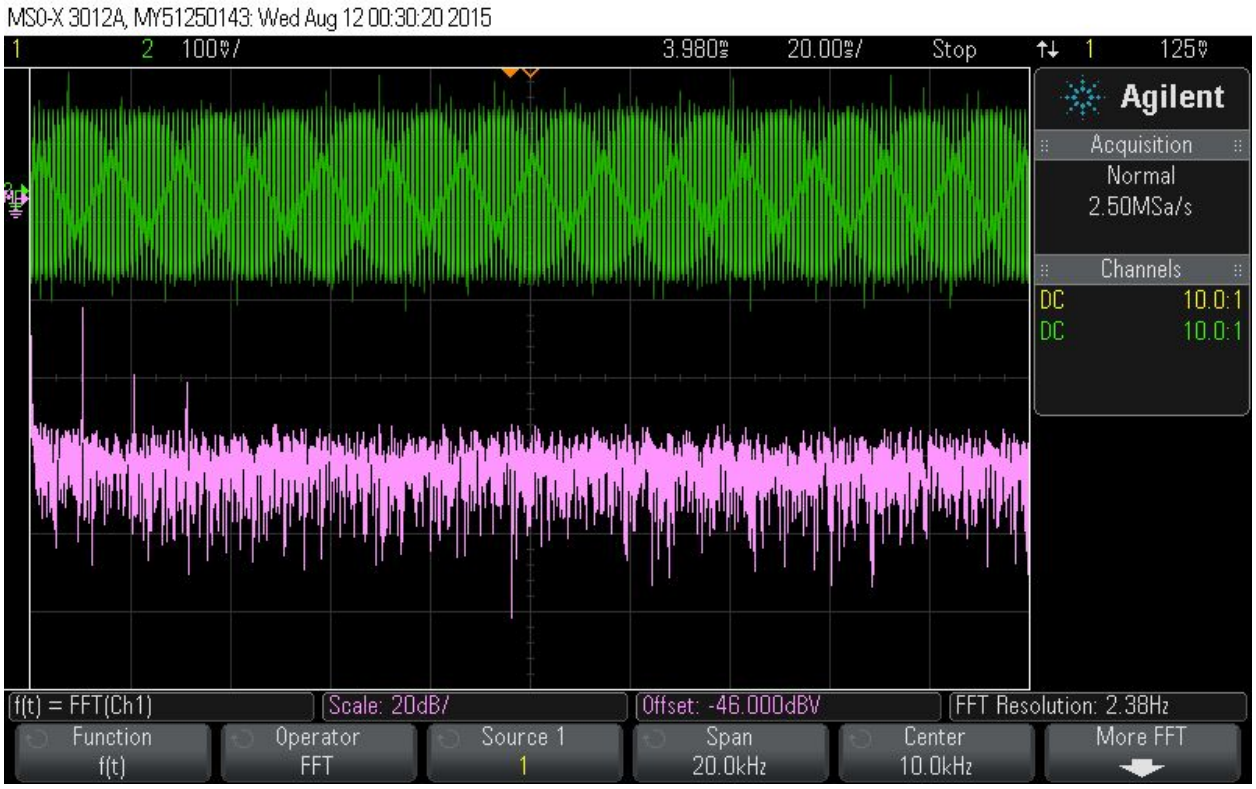

Figure 8.2. Signal to Noise Ratio Test.

The signal to noise ratio can be determined by having a singular tone play and performing an FFT on the signal. As shown above a tone peak appears but along with unintended harmonics. Using cursors the difference between the tone's peak and the noise level is around 45 dBV. This did not meet our original specification as we aimed to have 90 dBV. The output is admittedly a little noisy to the human ear but this could be due to the probes. While connecting the probes the noise became much more apparent with increased volume.

	Minimum	Maximum
Signal to Noise Ratio	44 dBV	44 dBV
Latency	2.64 milliseconds	5.36 milliseconds

Summary: The FM synthesis has relatively met our predefined specifications. The output is a bit noisier when probing but sounds fine without. Latency is low enough for the synthesis to be considered in real time.

IX. Conclusions

On the ATMega side, the switches and potentiometers were all able to send the correct data. This was tested and verified by first serial printing the bytes produced by each subsystem to the console. Next, using a logic analyzer, we were able to verify that the bytes were received by the DSP chip. Unfortunately, due to time constraints, we were unable to actually create subtractive and sample based synthesis in the DSP, and we also had difficulty implementing the equalizer due to problems with a dsp library that was supposed to carry out a Fast Fourier Transform, so it was abandoned.

Appendices

A. Analysis of Senior Project Design

Project Title: Danalog

Student's Names: Evan's Lew, Vikrant Marathe, Bryan Bellin, Jordan Wong

Advisor's Name: Wayne Pilkington **Advisor's Initials:** **Date:** 6/16/17

Summary of Functional Requirements:

The Danalog produces audio via FM synthesis with two note polyphony. It has a controllable ADSR envelope and phase between the carrier and modulating wave. There is also a digital equalizer to boost/attenuate certain frequency ranges, a master volume/pitch fader, and a modulation wheel that can affect a user defined parameter with ease. Finally, up to two digital effects (reverb, flange, chorus, etc) with adjustable parameters can be applied to the audio signal. All settings are displayed between two LCD screens.

Primary Constraints:

- Given a fixed point DSP chip, we were restricted to fixed point computations, greatly preventing accuracy in calculations which could have been achieved with a floating point processor.
- Using TI's dsplib for optimized fixed point processing created a large detour that unfortunately led to no results. The FFT function for our equalizer required a twiddle

factor table to multiply the signals with the factors, but we could not get the the table to be read properly in our program.

Economic:

Several hundred man-hours were put into product design, subcircuit building/testing, subcircuit integration, and programming the Danalog. A total of $792.70 was needed to make the project a reality. Several components and peripherals were needed, and a PCB had to be built and printed to connect the peripherals together. The chassis and keys were made from plastic, the PCB was made from fiberglass and copper, and many of the components as well as the development board were made of plastic, fiberglass, and various metals.

The vast majority of costs accrue in prototyping the product, researching and developing, and ordering all the necessary components. With an optimum design established, the cost to build a single Danalog will be significantly reduced, and we are confident we will be able to establish a strong customer base that will buy the product, which will compensate for the costs and eventually lead to a profit at the peak of its sales.

Originally, the project was estimated to cost $300. At the end, all the materials ended up costing $469.19. The bill of materials is shown as follows:

Price	Order
$106.67	K25M Keyboard from Amazon
$98.29	Sparkfun order (Buttons, LCD Screens, MIDI Connector, Jumper Adapter, Header
$119.26	Digikey order (Pots, Encoders, Rotary switches, Multiplexers, Diodes, Audio Jacks, Amplifiers)
$23.98	2 Arduino Megas
$12.96	DSP connector
$96.00	PCB
$3.61	USB adapter
$8.42	USB cable

The prototype included the purchase of many extra components in case of part failure or damage, therefore the cost would most likely be around $400, therefore selling at a price of around $450 would easily create a large profit for our company.

The products would emerge as soon as the first mass shipment of Danalog is complete, which would most likely occur about a year from the completion of the prototype. We expect a long shelf life of about 10 years with no maintenance costs.

Our original estimated development time is as follows:

Winter Quarter 2017										
Week	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
Month	JAN				FEB				MAR	
Day	9	16	23	30	6	13	20	27	6	13
Design										
Hardware Design										
Hardware Simulation										
Software Design										
Design Review										
Parts Research & Testing										
Select Components										
Research Cost-Effective Components										
Purchase Components										
Microcontroller										
Main Code										
DSP Synthesizer										

Winter project timeline

Spring Quarter 2017											
Week	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11
Month	APR				MAY					JUNE	
Day	3	10	17	24	1	8	15	22	29	5	12
Microcontroller											
Debug Code											
Ensure Proper Operation											
PCB Fabrication & Layout											
Design & Layout											
Assembly & Testing											
Full System Integration											
Full Breadboard Testing											
System Packaging											
Reports & Presentations											
Senior Project Report											
User Manual											
Demonstration							May 13				
Senior Project Expo										June 2	

Spring project timeline (estimated)

Our actual development time:

Spring Quarter 2017											
Week	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11

Month	APR				MAY					JUNE	
Day	3	10	17	24	1	8	15	22	29	5	12
Microcontroller											
Debug Code											
Ensure Proper Operation											
PCB Fabrication & Layout											
Design & Layout											
Assembly & Testing											
Full System Integration											
Full Breadboard Testing											
System Packaging											
Reports & Presentations											
Senior Project Report											
User Manual											
Demonstration											June 16
Senior Project Expo										June 2	

Once project ends, perhaps we will work to improve on the shortcomings of the prototype in order to meet the expectations of the beginning of the project.

Environmental:

Aside from the raw metal ore and plastics being manufactured to produce this product, there is no significant environmental impact from this product.

Manufacturability:

Since our PCB was printed by a third party company, it was important to verify the design is correct before sending out an order for the print. Also, the chassis had to be created one half at a time due to fact that we were using a group member's 3D printer.

Sustainability:

There are not really any issues associated with maintaining the synthesizer. One upgrade that could possibly help is using a floating point digital signal processor in order to use decimal numbers in the C code for the DSP chip, making it easier to program accurate filters for signals.

Ethical:

None.

Health and Safety:

One potential concern with safety is the possibility of ear damage due to long exposure to audio by our users, or from accidentally setting the volume too high.

Social and Political:

This product is intended to mainly impact the amateur music industry, providing music enthusiasts an opportunity to toy with different sounds and experience the Danalog synthesizer.

Development:
One important technique used for this project is the ping-pong buffer. This was necessary for real time signal generation. Essentially, while the ping buffer was being written to by the audio generator, the pong buffer was being read by the DMA, and vice versa. This prevented any loss of time in outputting the audio signals without losing samples.

C. Project Schedule - Time Estimates & Actuals
Our original estimated development time is as follows:

Winter Quarter 2017										
Week	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10
Month	JAN				FEB				MAR	
Day	9	16	23	30	6	13	20	27	6	13
Design										
Hardware Design										
Hardware Simulation										
Software Design										
Design Review										
Parts Research & Testing										
Select Components										
Research Cost-Effective Components										
Purchase Components										
Microcontroller										
Main Code										
DSP Synthesizer										

Winter project timeline

Spring Quarter 2017											
Week	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11
Month	APR				MAY					JUNE	
Day	3	10	17	24	1	8	15	22	29	5	12
Microcontroller											
Debug Code											
Ensure Proper Operation											
PCB Fabrication & Layout											
Design & Layout											
Assembly & Testing											
Full System Integration											
Full Breadboard Testing											
System Packaging											
Reports & Presentations											
Senior Project Report											
User Manual											
Demonstration							May 13				
Senior Project Expo										June 2	

Spring project timeline (estimated)

Our actual development time is on the next page

Actual Development Time

Spring Quarter 2017											

Week	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11
Month	APR				MAY					JUNE	
Day	3	10	17	24	1	8	15	22	29	5	12
Microcontroller											
Debug Code											
Ensure Proper Operation											
PCB Fabrication & Layout											
Design & Layout											
Assembly & Testing											
Full System Integration											
Full Breadboard Testing											
System Packaging											
Reports & Presentations											
Senior Project Report											
User Manual											
Demonstration											June 16
Senior Project Expo										June 2	

Once project ends, perhaps we will work to improve on the shortcomings of the prototype in order to meet the expectations of the beginning of the project.

E. Program Listings

Mux, Switches, Buttons

```
//Synth Mux C file
#include "SynthMux.h"
#include "serial_usb.h"
#include <avr/io.h>
#include <stdio.h>
#include <util/delay.h>

uint8_t switches;
```

```
void Synth_Mux_Init(void)
{
		PORTJ |= (1<<PJ0);   //Set Pullup on PJ0
		DDRG |= (1<<DDG5);   //Set PG5, PE5, PB7, and PD2 as outputs
		DDRE |= (1<<DDE5);
		DDRB |= (1<<DDB7);
		DDRD |= (1<<DDD2);
}

uint8_t Synth_Mux_Select(void)
{
	uint8_t cache_g = 0;
	uint8_t cache_e = 0;
	uint8_t cache_b = 0;
	uint8_t cache_d = 0;
	uint8_t i=0;
	uint8_t sw1=0, sw2=0, sw3=0, sw4=0, sw5=0;


	for(i=0; i<12; i++)
	{
		// Set S0
		cache_g = PORTG;
		cache_g &= ~(1<<PG5);
		cache_g |= (i&S0_MSK)<<S0_SFT;
		PORTG = cache_g;

		// Set S1
		cache_e = PORTE;
		cache_e &= ~(1<<PE5);
		cache_e |= (i&S1_MSK)<<S1_SFT;
		PORTE = cache_e;

		//Set S2
		cache_b = PORTB;
		cache_b &= ~(1<<PB7);
		cache_b |= (i&S2_MSK)<<S2_SFT;
		PORTB = cache_b;

		//Set S3
		cache_d = PORTD;
		cache_d &= ~(1<<PD2);
		cache_d |= (i&S3_MSK)>>S3_SFT;
		PORTD = cache_d;


		_delay_us(1);

		if ((PINJ&(1<<PINJ0)) == 0)
```

```
		{
			switch(i)
			{
				case 0 :
				sw1 = 5;
				break;
				case 1 :
				sw1 = 3;
				break;
				case 2 :
				sw1 = 4;
				break;
				case 3 :
				sw1 = 2;
				break;
				case 4 :
				sw1 = 1;
				break;
				case 5 :
				sw2 = 2;
				break;
				case 6 :
				sw2 = 3;
				break;
				case 7 :
				sw2 = 1;
				break;
				case 8 :
				sw3 = 1;
				break;
				case 9 :
				sw5 = 1;
				break;
				case 10 :
				sw4 = 1;
				break;
			}
		}
	}
	switches = (sw1<<5)|(sw2<<3)|(sw3<<2)|(sw4<<1)|sw5;
	//print_serial_usb("%u\n",switches);
	return switches;
}
```

Potentiometers, ADC

```
//potadc.c

#include "serial_usb.h"
```

```c
#include "potadc.h"
#include <avr/io.h>
#include <stdio.h>
#include <util/delay.h>

uint8_t adc_cache;
uint8_t adc_array[8];

void PotADC_Init(void)
{
	ADMUX = (1<<REFS0)|(1<<ADLAR);
	ADCSRA = (1<<ADEN);
}

void PotADC_Poll(void)
{
	int i, j;
	for(i=0; i<8; i++)
	{
		adc_cache = ADMUX;
		adc_cache &= ~(ADC_MSK);
		adc_cache |= i;
		ADMUX = adc_cache;
		ADCSRA |= (1<<ADSC);
		while(ADCSRA & (1<<ADSC));
		adc_array[i] = ADCH;


	}
	adc_array[5] = ~(adc_array[5]);
	//
	for(j=0; j<8; j++)
	{
		//print_serial_usb("%u\n", adc_array[j]);
	}

}
```

SPI

```c
//SPI.c

#include "potadc.h"
#include "encoder.h"
#include "midi.h"

#include "SPI.h"
#include <avr/io.h>
#include <avr/interrupt.h>
#include "serial_usb.h"
```

```
#include <util/delay.h>
uint8_t spidata, spistat = 0;
uint8_t switches;

/*ISR State Variables*/
uint8_t spi_tx_cnt = 0;
uint8_t spi_tx_type = 0;


// MIDI transfer state
#define MIDI_TX_STATUS_SENT 2
#define MIDI_TX_NOTE_SENT 1
MidiPacket current_packet;

void SPI_SlaveInit(void)
{
	/* Set MISO output, all others input */
	DDRB |= (1<<DDB3);
	/* Enable SPI */
	SPCR = (1<<SPE)|(1<<SPIE);
	sei();
}

ISR(SPI_STC_vect)
{
		if(spi_tx_cnt>0)
		{
			if(spi_tx_type == SPI_MIDI_CMD)
			{
		 switch (spi_tx_cnt) {
		   case MIDI_TX_STATUS_SENT:
		     SPDR = current_packet.note;
		     break;
		   case MIDI_TX_NOTE_SENT:
		     SPDR = current_packet.velocity;
		     break;
		 }
				spi_tx_cnt--;
			}
			else if(spi_tx_type == SPI_ENC_CMD)
			{
				SPDR = encoders[19-spi_tx_cnt].count;
				spi_tx_cnt--;
			}
			else if(spi_tx_type == SPI_POT_CMD)
			{
				SPDR = adc_array[8-spi_tx_cnt];
				spi_tx_cnt--;
			}
		}
```

```
                else {
                        spidata = SPDR;
                        if(spidata == SPI_MIDI_CMD)
                        {
                                if (!mbuffer_empty()) {
                                current_packet = mbuffer_read();
                                SPDR = current_packet.status;
                                spi_tx_cnt = MIDI_TX_STATUS_SENT;
                                spi_tx_type = SPI_MIDI_CMD;
                                }
        else {
          SPDR = 0;
        }
                        }
                        else if(spidata == SPI_ENC_CMD)
                        {
                                SPDR = encoders[0].count;
                                spi_tx_cnt = 18;
                                spi_tx_type = SPI_ENC_CMD;
                        }
                        else if(spidata == SPI_POT_CMD)
                        {
                                SPDR = adc_array[0];
                                spi_tx_cnt = 7;
                                spi_tx_type = SPI_POT_CMD;
                        }
                        else if(spidata == SPI_SWT_CMD)
                        {
                                SPDR = switches;
                        }
                }

}
```

Matlab Reverb Algorithm

```
%reverballpass.m
% This is a simple all-pass reverb filter with adjustable delay d and attenuation factor A. It
% creates a simple difference equation with a large amount of delays that decays over time
function [yout b a] = reverballpass(sample, d, A)

Fs = 48000;
[x,Fs] = audioread(sample);
num = -A*[1 zeros(1, d-1) -1/A];
den = [1 zeros(1, d-1) -A];
y = filter(num, den, x);
if(nargout == 1)
    yout = y;
else
    yout = y;
```

```
    b = num;
    a = den;
end
soundsc(yout,Fs);
```