

Audio DSP Amplifier

by

Will Saba
Nicholas Barany

Senior Project
Electrical Engineering Department
California Polytechnic University
San Luis Obispo
June, 2015

Table of Contents

<u>Section:</u>	<u>Page:</u>
Abstract:	7
Chapter 1: Introduction	8
Chapter 2: Requirements and Specifications	11
Chapter 3: Project Plan.....	15
Chapter 4: System Design.....	22
Chapter 5: Testing.....	36
Chapter 6: Conclusions and Discussion.....	47
References:	51
<u>Appendices:</u>	
A. Senior Project Analysis.....	55
B. Software Design.....	60
C. Market Analysis.....	120

List of Figures and Tables

Tables:

2.1) Requirements and Specifications for DSP Amplifier.....	12
2.2) Engineering/Marketing Trade Off Metric for DSP Audio Amplifier.....	13
3.1) DSP Audio Amplifier Top Level Functionality.....	15
3.2) Microcontroller Level 1 Functionality.....	16
3.3) DSP Board Level 1 Functionality.....	17
3.4) Testing and Verification Plan.....	18
3.5) Audio-DSP Amplifier Deliverables.....	19
4.1) LCD to Arduino Connections.....	25
4.2) Rotary Encoder to Arduino Connections.....	26
4.3) ADAU1702 DSP Eval Board to Arduino Connections.....	26
4.4) Planned Menu State Chart.....	29
5.1) Hardware Test Output Voltage Summary.....	43
6.1) Final Requirements Conclusions.....	49
A.1) Audio DSP Amplifier Estimated Cost.....	56
C.1) Audio-DSP Amplifier Development Costs.....	123
C.2) Competing Product Solutions Comparison.....	126

Figures:

1.1) Room EQ Wizard Interface.....	8
1.2) U.S Census Infographic on Computers in the Home.....	9
3.1) Level 0 Black Box Diagram.....	15
3.2) Level 1 Black Box Diagram.....	16
3.3) Gantt Chart Legend For Figures 3.4-3.6.....	20
3.4) Gantt Chart Fall 2014.....	20
3.5) Gantt Chart Winter 2015.....	21
3.6) Gantt Chart Spring 2015.....	21
4.1) Arduino Rotary Encoder and LCD Black Box Diagram.....	23
4.2) Hardware Debouncing of a Button Switch.....	24
4.3) Audio amplification and DSP Interfacing.....	26
4.4) Sigma Studio Signal Flow Diagram, Senior Project Expo.....	27
4.5) Final Sigma Studio Signal Flow Diagram.....	28
4.6) Menu Architecture, Full System.....	28
4.7) Menu Architecture, Simplified.....	29
4.8) Cardboard Mock-up for Enclosure Design.....	33

4.9) Front View Enclosure Model.....	34
4.10) Power Cord Recess Dimensions.....	34
4.11) Enclosure Back Panel.....	34
4.12) Painted Amplifier Enclosure.....	35
5.1) I ² C Bus Verification.....	36
5.2) Sigma Studio Basic Audio Adjustment GUI Interfacing.....	36
5.3) First Attempt at I ² C Transmission.....	37
5.4) Second Attempt at I ² C Transmission.....	38
5.5) Program Data Serial.print Transmission Test.....	39
5.6) Param Data Serial.print Transmission Test.....	39
5.7) File Parsing Signal Flow Chart.....	40
5.8) String Parse Test.....	41
5.9) Filter String Import With Parameter Decoding.....	42
5.10) Parametric Coefficient Calculation Verification.....	42
5.11) System Integration and Connection Testing.....	43
5.12) 125 Hz High Pass Illustrating a 6 dB/Octave 1st Order Slope.....	44
5.13) 4 dB Low Shelf with Fc of 180 Hz.....	45

5.14) Notch Filter, Fc of 300 Hz, Q=4.....	45
5.15) Lowpass at 500 Hz Compounded with -6 dB High Shelf	46
A.1) OSHA Sound Level Safety Chart.....	59
B.1) Menu Flow Diagram.....	60
B.2) Example REW Filter File.....	61
C.1) DSP Board Cost Comparison.....	128
C.2) Estimation of Design Method Ease of Implementation.....	129

Abstract:

The key concept of this project is to create a microcontroller system that serves as an interface between a DSP board and a total of 4 amplifier channels. The fully integrated system will provide a fully inclusive audio DSP amplifier for use in 2.1 or bi-amplified stereo speaker setups. The project will focus on developing an intuitive interface that is operable from the device or a computer that programs the DSP board for various speaker applications. The finished design will provide a custom computer sound amplifier in one package, eliminating the need for multiple components by interfacing two stereo amplifiers, a DSP unit, and an LCD menu using a microcontroller. This solution will provide a more affordable alternative to the current market solution for creating a DSP enabled, 2.1 sound system. The system will provide higher quality audio with more customization options than current competing market solutions.

Chapter 1: Introduction

Overview

The 2 input 4 output.1 DSP audio amplifier project is an approximately 2 rack unit solution that utilizes a microcontroller to program a DSP unit and drive a graphical interface controlled by sensors and knobs which interfaces with a 4 channel (bi-amplified stereo) amplifier that has the ability configure as a stereo plus subwoofer system. This device is entirely self contained in that the user sends it an audio signal over RCA or Coaxial S/PDIF and it outputs to passive speakers. Current solutions to a bi-amplified speaker with subwoofer (heretofore referred to as 2.1) system require you either to buy several components, use a specific set of speaker hardware, learn how to integrate an often archaic signal processing unit, or don't allow the end user full flexibility over inputs and outputs. The goal of this project is to fill the niche market of a single unit DSP 2.1 amplifier that is both easy to use and achieves high flexibility with modest cost. To give the reader a reference for what DSP software looks like from the user's perspective, see below capture of Room EQ Wizard V5, the most widely used and most powerful free DSP software available often used for home theater applications.

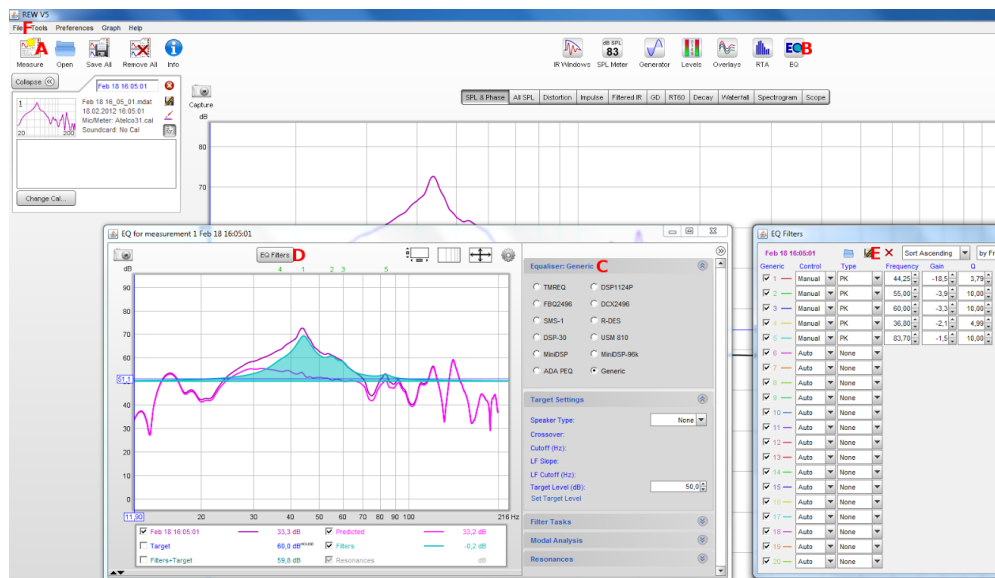


Figure 1.1: Room EQ Wizard Interface

Software such as this is extremely useful as an open source baseline for determining which filters will achieve a desired response and can be used in tandem with this product to achieve a more automated system. In short, this software adds microphone measurement and calibration capabilities and through room measurement would generate a list of parametric filters, that correct for room modes and speaker deficiencies, to be entered into the DSP system.

Why is Quality Audio Important?

Before diving into the bulk of this report, a small debriefing on what the 2.1 DSP amplifier will provide to the consumer is helpful to understand why this solution is important. Digitally transmitted media is becoming more and more common in modern homes. 99 percent of American households own at least one television [17]. Most households also have computers, and speakers.

Household Computer and Internet Use: 1984-2011

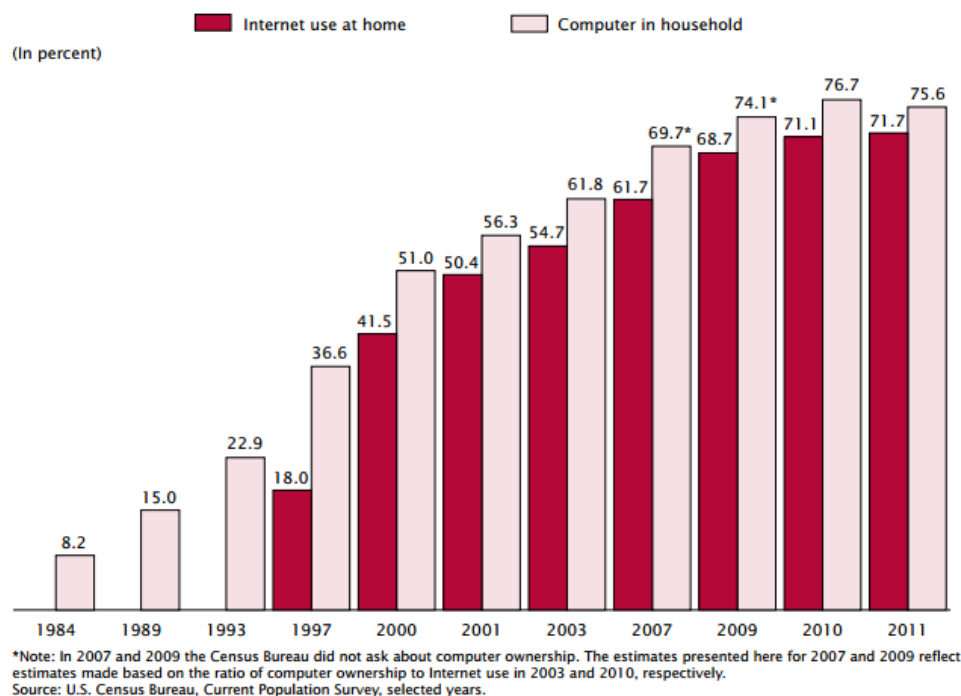


Figure 1.2: U.S Census Infographic on Computers in the Home[18]

All of the songs, movies, and shows people enjoy to everyday use audio signals. A high quality audio signal can fully immerse the listener, amplifying the emotions felt and creating an overall enhanced media experience. In order to obtain this experience in the home, a DSP home theater system is a necessity.

Why DSP?

DSP is the single most flexible and powerful tool that a loudspeaker designer can use to maximize the performance from a speaker and the fact that most if not all home audio is conducted inside where room modes dominate frequency response means even an acoustically flat speaker will not perform optimally in an untreated room[1]. DSP allows the end user to correct for room response, driver matching imperfections based off of manufacturing tolerances, transition between low mid and high frequency drivers with sharp transitioning phase response, and is utilized in all modern surround sound receivers as well as most all professional audio applications. DSP solutions take significant design and development in order to incorporate into products and as such very few inexpensive amplifiers utilize them. Graphical all-in-one products

come at an extreme markup because most are tailored towards professional, much more complex processing.

The biggest trend in audio innovation is the utilization of DSP to achieve otherwise impossible (or prohibitively difficult) audio performance [3]. Marginal audio performance improvement can be made by increasing amplifier performance compared to mass produced market available units and due to the team's desire to work on embedded systems rather than chip level design, we chose a system level approach instead of electronic design [2]. There is a surprising void of price competitive dsp amplifier solutions beyond the ubiquitous stereo DSP pro amplifier. In recognition of this emphasis on DSP systems and due to the fact that interfacing existing components including: amplifiers, dsp board, and a microcontroller with a display and physical controls to achieve a high performance system is the industry standard practice, the focus of this project will be integrating commercially available components into a user experience oriented, aesthetically pleasing device.

Chapter 2: Requirements and Specifications

The following section describes the process of how the engineering requirements and specifications for the Audio DSP Amplifier are derived. To begin the process, the customer desired attributes are established to determine basic product requirements. From the customer desired attributes, the engineering requirements and specifications are created. The following subsections highlight the process in more detail.

Customer Desired Attributes

- Production price must not exceed competing solution options, <~\$500.
- Enclosure design must be aesthetically pleasing (minimal buttons, simplified I/O panel, no front vents).
- Size must not exceed 3u standard rack mount dimensions for potential use (via adaptor) in audio racks: 18.19" inch wide by 5.256" tall.
- Quiet, <35 dBA, operation under standard operating load.
- Amplifier must be capable of handling loads as low as 4 ohms on all channels at rated power without overheating.
- Device must have at least 6 parametric filters for equalization, with separate high pass and low pass filters of 1st, 2nd, 4th, and 8th, orders for each output.
- Device must have user programmable limiting with variable release times for driver protection.
- Device must not pull more than 8 amps using a source voltage of 120 V during heavy amplifier clipping scenario (1/3 duty cycle, constant clipping program material)[9].

The unit will be housed in a custom fabricated enclosure with an interfacing comprising of the following:

- LCD display
- Array of rotary encoders to operate audio controls such as volume
- Speaker outputs
- USB input for direct access to the DSP

The amplifier should be capable of outputting 50 watts per channel (assuming a nominal 4 Ω load) for each of 4 main speaker outputs: low-left, low-right, high-left, high-right, and 50 watts for a mono subwoofer out (that replaces the low outputs). This design will require a DSP board, four channel amplifier, mono amplifier, and microcontroller to be purchased as the goal of the project is to implement the system rather than design each component individually.

Requirements and Specifications

Table 2.1 Below lists the engineering requirements and specifications for this project. The marketing requirements for this product are listed below Table 2.1.

Table 2.1 Requirements and Specifications for DSP Amplifier

Marketing Requirements	Engineering Requirements	Justification
2,4,6,7	1. The <i>total harmonic distortion</i> at rated output power should be <0.1 %	Based off competitor performance specs and class D amplifier topology, this THD should be obtainable. *Money Spec[2]
1,2,4,6,7	2. Should be able to sustain an average <i>output power</i> > 50 watts to each main output assuming 4 ohm loads for each channel.	This power range should be sufficient for all reasonable home theater and stereo listening scenarios that maintains a reasonable power supply footprint. This amount of power exceeds all cheap alternatives and thus puts it in its own class. *Money Spec
1,2,4,6,7	3. Should have an efficiency ($\eta > 80\%$)	Maintaining a high standard for efficiency is in line with the expected class D topology and ensures device longevity by reducing heat generation.
2,3,5	4. Average initial setup time should not exceed 30 minutes.	This length of time should be sufficient to connect speaker cables and go through initial filter setup. This is important for the desired low learning curve.
1-7	5. The dimensions should not exceed 18.19" wide by 5.256" tall by 14" deep	Target size is less than 3 rack units in size such that it can be mounted via adaptor in a rack case.
2-6	6. The dsp unit should have at least 6 parametric filters, custom High/Low pass filters with multiple orders with varying topologies, and delay/phase for each output.	Individual HP/LP filters for each output is key to providing maximum flexibility and 6 parametric filters is the average found in most professional DSP's.
1,2,5,6,7	7. Device should generate less than 35 dBa of sound under	Critical listening applications require silent or near silent

	average load.	amplifiers.
1-7	8. Production price must not exceed \$500	Beyond this price it becomes feasible to buy alternative components and match if not exceed performance, below this price point the system carries an edge.
6,7	9. Device must not draw more than 5 amps on a 120 V circuit under a worst case scenario.	At an efficiency of 80%, maximum power draw assuming sinusoidal output should conservatively attain this spec. For home purposes excessive current draw can lead to tripping a breaker or device damage which do not review well (marketing) negatively impact device reliability.

*Money Spec indicates a specification that customers are willing to spend extra in return for a specific performance metric.

Marketing Requirements:

1. The system should be quiet for use in a home environment.
2. The system should have excellent perceived sound quality.
3. The system should be easy to use and have a minimal learning curve.
4. The system shall be competitively priced (< \$600) to attract the DIY end user.
5. The system shall be user configurable from the device and a PC.
6. The system shall fit in a standard audio rack mount space .
7. The system amplifier stage shall maintain greater than 80 percent power efficiency for loads over 10% rated power

Table 2.2 Engineering-Marketing Trade Off Metric for the DSP Audio Amplifier

		THD	Output Power	Amount of DSP	Learning Curve	Cost	Size
		-	+	+	-	-	-
1)Device Noise	-	↑	↓↓			↓	↓
2)Sound Quality	+	↑↑	↓	↑↑		↓↓	↓↓
3)Easy to	+		↓	↓↓	↓↓	↓	↑

Use/Install							
4)Cost	-	↓↓	↓↓	↓	↓	↑↑	↓
5)Flexibility	+		↑↑	↑↑	↓↓	↓↓	↑↑
6)Size	-	↓	↓↓	↓	↑	↓↓	↑↑
7) Efficiency	+	↓	↓			↓↓	↓↓

Table 2.2 above shows the correlation between marketing specifications and the engineering design goals in a grid formation by taking each row (marketing spec) and comparing it to each column (engineering spec). Two arrows up indicates a high level of correlation, meaning that improving the marketing specification will correspond with a strong improvement in the corresponding engineering specification whereas two arrows down indicates that the marketing spec strongly conflicts with the engineering goal. Single arrows down or up apply a similar logic with a less drastic degree whereas blank cells represent little to no relation between the marketing and engineering spec.

Chapter 3: Project Plan

The following section details the overall project plan, focusing on top level functionality of system design. Level 0 and Level 1 black box diagrams are used to showcase the system inputs/outputs (Level 0) and subsystem interconnections (Level 1). Basic testing and verification plans are established alongside Gantt charts displaying the overall project work schedule for both partners. A final overview of division of labor for each partner rounds out the section.

High Level Overview:

Figure 3.1 and Table 3.1 below showcase the Level 0 Block Diagram of the Audio DSP Amplifier as well as its respective input/output descriptions.

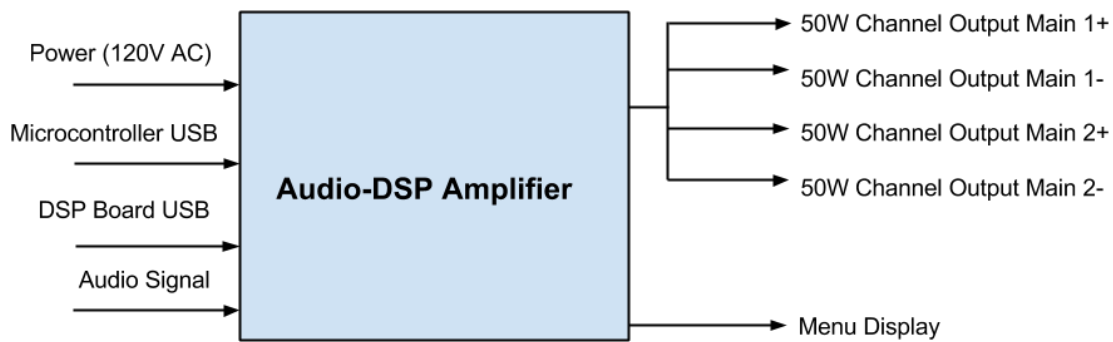


Figure 3.1: Level 0 Black Box Diagram

Table 3.1: DSP Audio Amplifier Top Level Functionality

Input/Output	Type	Description
Input	Power	Main power supply input power: US standard 120 V/60 Hz AC
Input	Microcontroller-USB	USB connection for reprogramming menu behavior and active development.
Input	DSP-USB	Connection for computer based filter programming
Input	Audio Signal	'One pair of RCA phono panel-mount jacks.

Output	Speaker Level Audio	4 Independent binding post speaker outputs.
Output	LCD Display	This display serves as the graphical interface for programming or modifying the DSP filters and setting input-output routing.

Figure 3.2 and Tables 3.2 and 3.3 below showcase the Level 1 Block Diagram and function description for the Microcontroller and DSP board.

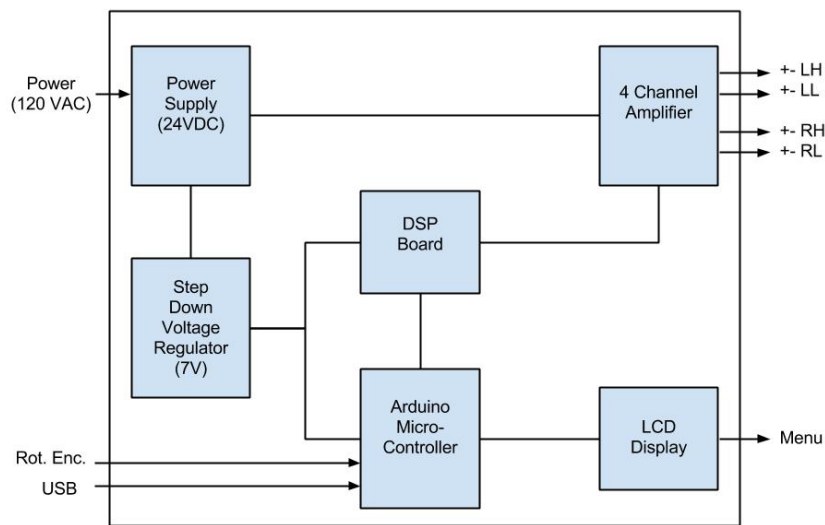


Figure 3.2: Level 1 Black Box Diagram

Table 3.2: Microcontroller Level 1 Functionality

Input/Output	Type	Description
Input	USB	Computer to arduino with USB Micro-b termination [4]
Input	Rotary Encoder	Push type knob for setting DSP filters and limiters for LCD interface.
Input	BTN	Navigating categories for LCD interface
Input	Power	7V regulated input voltage
Input/Output	I ² C Digital Data	Interface between DSP board and microcontroller for

		reading and writing DSP configuration files. [5]
Output	LCD Display	Graphical interface for programming or modifying the DSP filters and setting input-output routing. [4]

Table 3.3: DSP Board Level 1 Functionality

Input/Output	Type	Description
Input	Audio signal input	RCA stereo signal coming from panel mount connectors.
Input	Power	7V regulated input voltage
Input/Output	I ² C Digital Data	Interface between DSP board and microcontroller for reading and writing DSP configuration files. [5]
Output	4 channels of RCA audio signal	4 main channel outputs with subwoofer output derived from summing woofer outputs.

Testing/Verification Plan:

The projected testing and verification plan of the engineering requirements and specifications is shown using the methods in Table 3.4 below.

Table 3.4: Testing and Verification Plan

Engineering Requirement/Specification	Plan of Verification
I2C Communication	Oscilloscope capture of I2C data transfer
Filter Frequency Response Accuracy	Use an oscilloscope to measure the frequency response of the implemented filters expected value within 1dB of their respective cutoff frequencies using its FFT function
Filter Coefficient Generation Accuracy	Test one of each filter type as calculated by microcontroller and compare the converted biquad coefficients to those generated by Sigma Studio (within 5%).
Self-noise Level	Noise level should not disturb the user nor exceed 40 dBa as measured with an SPL meter 1 meter from the enclosure (lid on).

Testing of the design itself will incorporate many pass fail checkpoints to guide debugging including the following:

- Does the dsp boot up stand alone?
- Does the DSP accept analog signals?
 - If not then modify source selection hardware/firmware and check jumpers
- Does the output voltage of the DSP match the input sensitivity of the amplifiers?
 - If out of range (clipping the output or too small of an input), adjust input sensitivity fixed gain settings for the amplifiers such that full scale output of DSP corresponds with maximum power output of the amplifier.
- Does the main menu behave as expected, transitioning from menus to submenus smoothly?
 - If not, check microcontroller code, verify LCD driver is behaving properly, and check for memory leaks and finite state logic errors.
- Does changing DSP settings make an immediate impact on sound?
 - If not, verify that microcontroller is forcing a DSP board reset and that it is uploading the new file with the configuration profile.

Project Schedule:

Nicholas Barany:

My focus on this project is primarily creating software to interface the microcontroller with the physical hardware of the system. The rotary encoder, volume knob, menu control buttons and the LCD Display communicate with the microcontroller directly without the need for a serial protocol interface[4]. Since this portion is not as difficult the I²C interfacing of the microcontroller and the DSP board[5], I am also working alongside my partner to finish the DSP interfacing after the physical hardware interfacing is complete. Both us are testing the system once fully assembled, however my testing focus once again lies with the physical hardware interfacing that I coded. See the Gantt charts in Figures 3.5-3.8 below for a more detailed division of labor.

Will Saba:

The majority of my contribution to the project is focused on interfacing between the microcontroller and the DSP unit. Specifically, I am in charge of writing the configuration code for reading, modifying, and exporting DSP settings files such that the microcontroller LCD interface can be used to program the DSP. I am also the design lead for the construction of the system's physical enclosure as I have more enclosure design experience. Furthermore, the audio testing and filter verification sections remain in my focus. Both my partner and I are splitting the documentation effort and LCD menu design. Near the end of the development process,(specifically in the testing and bug testing software phase) the task allocation will become more team centric due to the nature of the problems.

Table 3.5 below showcases the deliverables for this senior project. These are key project deadlines that are included in the Gantt charts of Figures 3.3-3.6 below.

Table 3.5: Audio-DSP Amplifier Deliverables

Delivery Date	Delivery Description
02/20/2015	EE 460 Report
02/20/2015	Design Review
03/10/2015	EE 463 Prototype Demonstration
04/20/2015	EE 464 Report
05/11/2015	EE 464 Demonstration
05/11/2014	ABET Senior Project Analysis
05/31/2014	Senior Project Expo

Figures 3.4-3.6 on the next two pages showcase Gantt Charts that display a more precise division of labor and projected completion goals and dates. This a tentative schedule that may be adjusted as the project progresses.

Key	Color
Work In Progress Will	Blue
Work In Progress Nick	Light Green
Work in Progress Mixed	Dark Teal
Complete by	Black
Due Dates	Red
Advisor Contact/Feedback	Yellow
Break	Light Grey
Major Segments	Dark Olive Green

Figure 3.3: Gantt Chart Legend for Figures 3.4-3.6

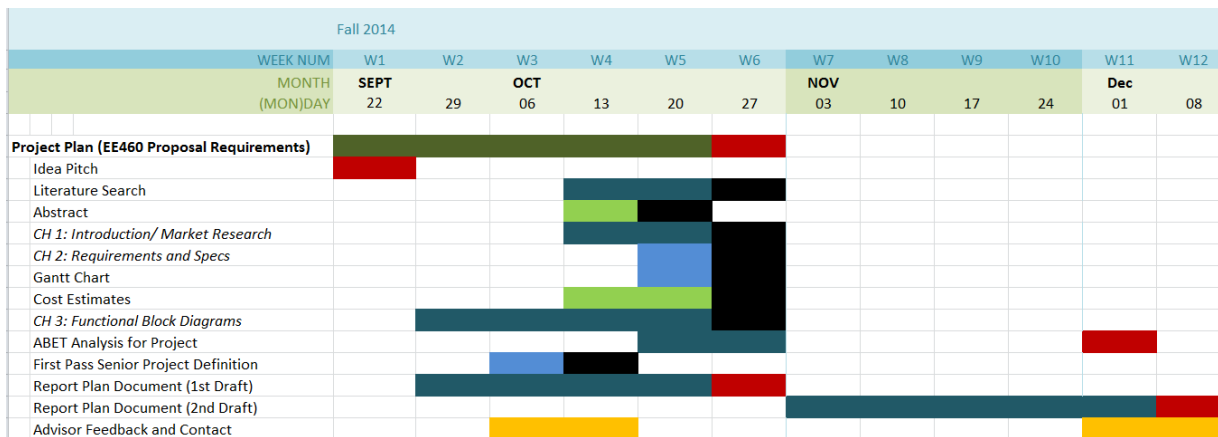


Figure 3.4 Gantt Chart Fall 2014

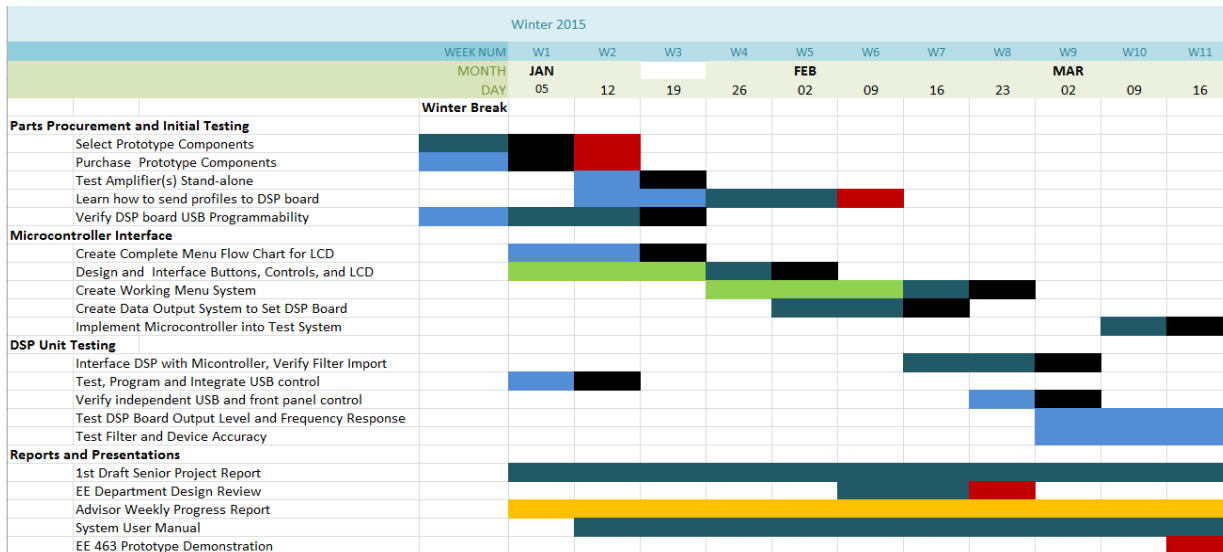


Figure 3.5: Gantt Chart Winter 2015

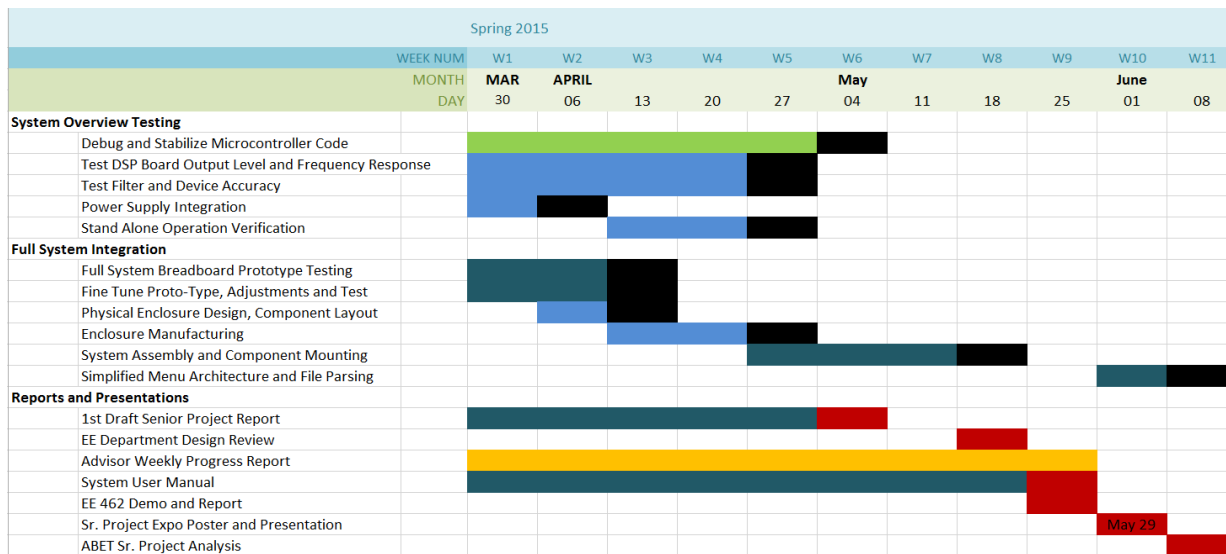


Figure 3.6: Gantt Chart Spring 2015

Chapter 4: System Design

The goal of this chapter is to fully describe how the system is configured and operates from both a physical connection and software logic point of view. As such, this section includes details such as wiring diagrams for the display and SolidWorks models for the enclosure. Further, this section specifies hardware and software flow both from a menu and SigmaStudio perspective. Finally, this section also addresses strategies employed to combat memory limitations as it pertains to coding structure.

Hardware:

The following hardware components below are used to interface and operate the DSP amplifier system:

- Analog Devices ADAU1702 Eval Board
- Arduino ATmega328p
- HDM16216H-B Hantronix LCD display
- 24VDC → 7VDC Drok Voltage Regulator
- 1 COM-09117 Rotary Encoder
- Resistors: 9.1kΩ, 4.7kΩ, 4.7kΩ, 2.2kΩ, 2.2kΩ, 2.2kΩ
- Capacitor: 1uF
- 20 male-male leads
- 3 female-male leads
- 2 2' stereo RCA cables (DSP to Amps)
- 4 2' 16 gauge speaker wire segments (power, gnd)
- 4 6" 16 gauge speaker wire segments (speaker out)
- 4 Dual banana terminals
- 1 RCA pair of terminals
- 1 AC power cord

Figure 4.1 below showcases the black box diagram for the front end interfacing between the rotary encoder, the ATmega328p, the ADAU1701, and LCD display.

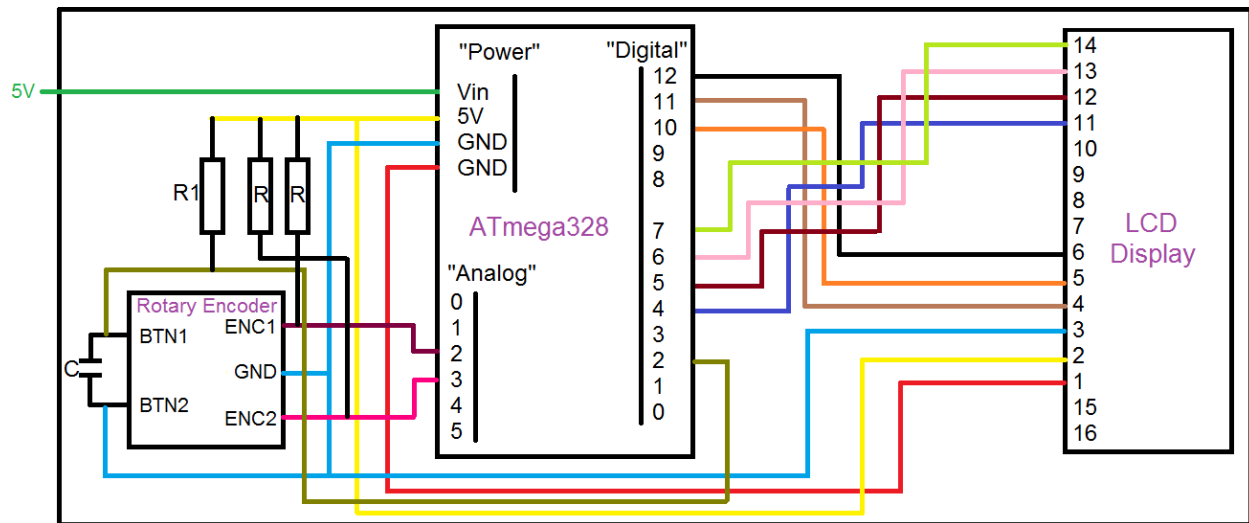


Figure 4.1: Arduino Rotary Encoder and LCD Black Box Diagram

The component values R1 and C are set to debounce the button. Both R values are pull-up resistors that can range from $\sim 1\text{k}$ to $\sim 5\text{k}$. More detail on chosen component values is shown in the debouncing and pull-up sections below.

Pull-Up Resistors

Pull-up resistors are resistors that are tied from the signal conductor to the positive rail to ensure that digital logic levels are met if high impedance or communication device disconnection is introduced to the circuit. Primary uses for these resistors in this project are for establishing solid I²C communication bus voltage levels and stable logic levels for the rotary encoder signals. Pull-up resistors can have any value between $1\text{k}\Omega$ and $5\text{k}\Omega$ to be effective. Choosing a resistor value between this range ensures enough current is drawn for the device to recognize the transition.

Rotary Encoder Debouncing

Debouncing is necessary in order to prevent the arduino from triggering multiple interrupts upon pressing of the rotary encoder. While software debouncing is possible, it is taxing on the microcontroller's processing power. Hardware debouncing using an RC circuit is a more effective approach to solving this issue. Figure 4.2 below shows a basic schematic of debouncing a switch (such as a button).

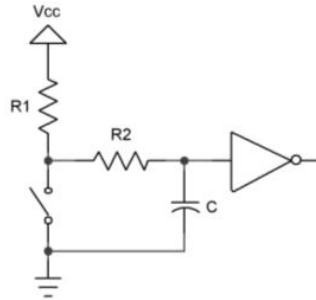


Figure 4.2: Hardware Debouncing of a Button Switch

By placing a capacitor in parallel over the switch and a pullup resistor from the output to the positive rail, an RC circuit is formed. While the switch is open (i.e unpressed button), the capacitor charges from 0 to 5V over time. The logic level will reach from low to high at time $\tau = R_1 C_1$. When the mechanical switch is closed via a button press, the capacitor discharges slowly through R_2 to ground. R_2 prevents any bouncing spikes that may occur the switch is closed, while R_1 prevents any bouncing as the switch is opened. The general recommended time constant value for button debouncing is about 0.5msec. The following calculations are used to determine the resistor values needed to debounce a circuit at 0.5msec with a 1uF capacitor:

$$V_{\text{cap}} = V_{\text{initial}} e^{(-t/RC)}$$

$$R = R_1 + R_2$$

V_{th} = The worst-case transition point for a high-going signal on the arduino (3V)

V_{initial} = Rail of the device (5V from the arduino's voltage regulator)

$t = 0.5\text{msec}$ (desired debounce time)

$C = 1\mu\text{F}$ (arbitrarily chosen)

$$3 = 5 \exp(-0.0005/R(1 \times 10^{-6}))$$

$$\ln(0.6) = -0.0005/R(1 \times 10^{-6})$$

$$R = -0.0005/[\ln(0.6) * 1 \times 10^{-6}]$$

$$R = \sim 978\Omega \rightarrow 1\text{k}\Omega \text{ standard value with 5\% tolerance}$$

Note : R_1 as the pullup should be fairly larger than R_2 . A ratio of around 4~1 was used.

$R_1 = 810\Omega$ standard value with 5% tolerance

$R_2 = 220\Omega$ standard value with 5% tolerance

However, the button debouncing time proved to be not sufficient, so the debounce time was raised to 5ms. Leaving the capacitor at 1uF, R_1 and R_2 were approximately resized by a factor of 10.

Final Hardware Values:

$R_1 = 9.1\text{k}\Omega$ standard value with 5% tolerance

$R_2 = 2.2\text{k}\Omega$ standard value with 5% tolerance

$C = 1\mu\text{F}$

Note: Even with this hardware debounce extension, a software delay during the button press ISR of 50ms was necessary to ensure proper debouncing.

Wire Connections

Tables 4.1-4.3 below list the pin connections for each interfacing component.

Table 4.1: LCD to Arduino Connections

LCD Pin	LCD Name	Arduino Pin	Arduino Port Name
1	V_{SS}	POWER:GND	GND
2	V_{DD}	POWER: 5V	V_{CC}
3	V_O	GND	GND
4	RS	DIGITAL:11	PB3
5	R/W	DIGITAL:10	PB2
6	E	DIGITAL:12	PB4
7	DB0		
8	DB1		
9	DB2		
10	DB3		
11	DB4	DIGITAL:4	PD4
12	DB5	DIGITAL:5	PD5
13	DB6	DIGITAL:6	PD6
14	DB7	DIGITAL:7	PD7
15	LED+		
16	LED-		

Note: The blank spaces in Table 4.1 above are left blank because these connections were not used for this portion of interfacing.

Table 4.2: Rotary Encoder to Arduino Connections

Rotary Encoder Pin	Arduino Pin	Arduino Port Name
ENC1	ANALOG: 2	PC2
GND	GND	GND
ENC2	ANALOG: 3	PC3
BTN1	DIGITAL: 2	PD2
BTN2	GND	GND

Note: A capacitor should be placed in parallel over BTN1 and BTN2 and a pullup resistor should be placed from BTN1 to the rail to debounce the switch.

Table 4.3: ADAU1702 DSP Eval Board to Arduino Connections

Jumper Description	Jumper	Jumper Pin Number	Arduino Pin	Arduino Port Name
External SPI/I ² C	J8	1	ANALOG: 4	PC4
External SPI/I ² C	J8	3	ANALOG 5:	PC5
External SPI/I ² C	J8	10	GND	GND

Note: Pull-up resistors are needed to pull the I²C bus up to **3.3V**. Using the 3.3V voltage regulator on the ATmega328p, this is easily achievable. The DSP board must also be powered from J14 with a DC supply between 6V and 9V.

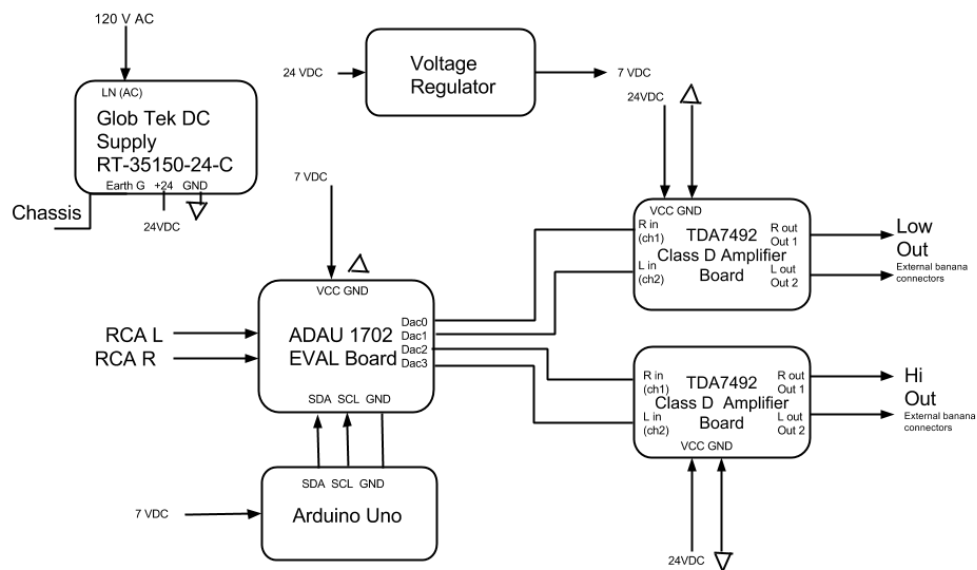


Figure 4.3 : Audio Amplification and DSP Interfacing

Figure 4.3 depicts the wiring diagram for the audio signal chain and power source. Stereo audio is received from the front panel of the enclosure via RCA panel mount connectors using a bare wire to 3.5 mm interconnect. The entire system runs off of a 24V DC powersupply with the microcontroller and DSP receiving regulated power and the amplifiers directly connect in parallel. RCA cables are used to connect the outputs of the DSP to the inputs of the amplifiers, jumpers connect the Arduino to the DSP and regulator, and bare wire connects the amplifier power, ground, and output signal.

Software:

DSP Signal Flow - Senior Project Expo

Programming the ADAU1702 DSP EVAL board starts with creating and editing a Sigma Studio project file. This project file designates hardware configuration, register control (for editing input types, ADC and DAC muting, program length and interfacing), and a functional block implementation of each DSP function to implement with the signal flow order indicated with yellow wire. In line with the philosophy of simple is better, the following filter blocks in Figure 4.4 represent the most straightforward to operate, yet still useful, DSP code as implemented for the senior project expo.

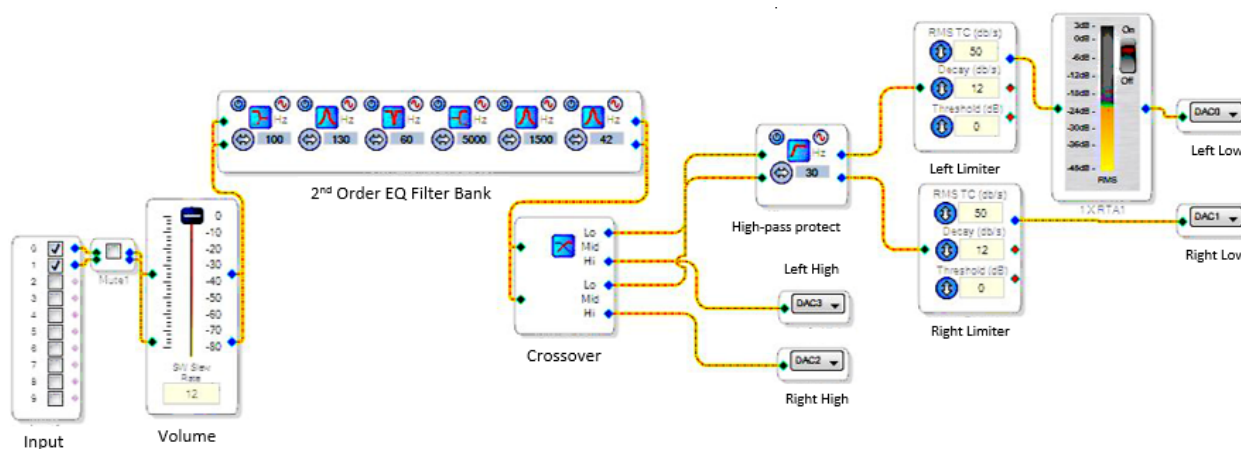


Figure 4.4: Sigma Studio Signal Flow Diagram, Senior Project Expo (May 29)

Key limitations of this version of the code (as constrained by microcontroller editing) include the high-pass protect and limiters had to be implemented in the boot sequence, the crossover filter type and slope is fixed to Linkwitz-Riley 24 dB/octave, and only the first filter within the EQ bank is menu editable. Most of these limitations result from limited microcontroller memory disallowing a more thorough menu structure with the exception being the crossover filter type which was chosen to be fixed for simplified user experience and because it is the correct type in excess of 90% of setup cases (Also, most users won't know or hear the difference between types). Following the expo, we chose to develop a filter parsing function within the microcontroller that allows the user to implement up to 10 filters and a mono subwoofer mode

using a Room Equalizer Wizard¹ (heretofore called REW) generated preset file (simulated using text strings stored in EEPROM). This preset file parse function expands the original capabilities of REW's auto-EQ by allowing the user to add notch, low-pass, and high-pass filters to the exported text file using the same format as the filter types implemented by the software. This preset file handling works based on the known order of words and thus the filter format must be followed exactly to behave correctly. See testing section for more information and an example text file displaying the expected format. Figure 4.5 below showcases the final signal flow diagram used for the DSP amplifier.

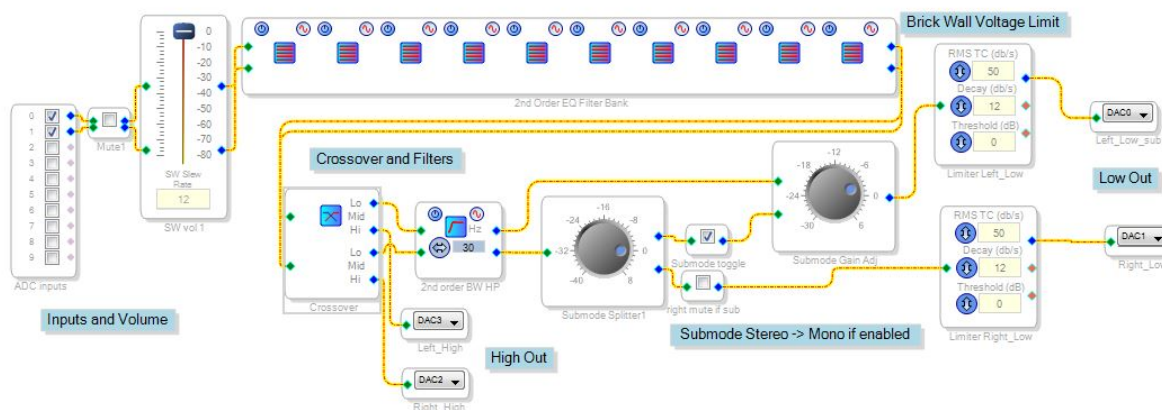


Figure 4.5: Final Sigma Studio Signal Flow Diagram

Menu - Architecture

The menu structure embodies the primary goal of the design UI: keep it simple. Minimal buttons and a simplistic memory structure keep the learning curve as intuitive as possible. The user adjusts the DSP parameters using a single knob on the front face of the enclosure, displayed using a 2-line seven segment display. The original menu layout (realizable with increased microcontroller memory) displayed parameters such as crossover point, filter type, EQ and speaker protection each have their respective sub-menu stemming from the top and are accessed with a single press of the menu button. Figure 4.6 below shows a high level description of this original menu flow in its entirety.

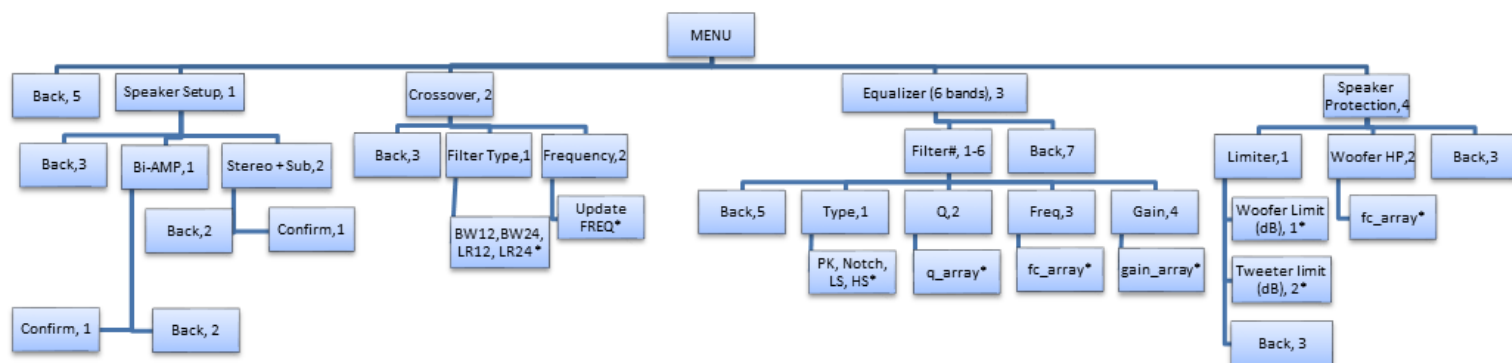


Figure 4.6: Menu Architecture, Full System (full size diagram in appendix B)

¹ <http://www.roomeqwizard.com/>

Note: each starred parameter in both menu architecture diagrams corresponds to a push turn command that edits the labeled value for the above menu architecture diagrams.

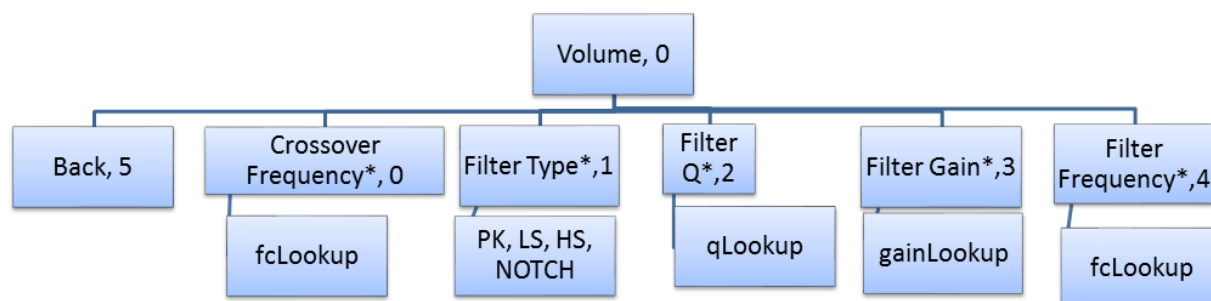


Figure 4.7: Menu Architecture, Simplified

Figure 4.7 above is the simplified menu state chart. It matches planned operation (Table 4.4) except the simplified menu follows a linear layout with a single parameter layer for decreased glitches and ease of use. As such there is not a separate state chart for the simplified layout.

Menu - Code Layout

The menu state behaves as a state machine (internally) and is maintained with an integer array and follows the below format: menu(depth, lvl 1 sub-branch, lvl 2 sub-branch, lvl 3 sub-branch, lvl 4 sub-branch). The default value for each of these variables is 0 and the menu variable itself is updated asynchronously on button presses and encoder rotations. Table 4.4 below displays a more in depth description of the menu state operation.

Table 4.4: Planned Menu State Chart

menu Value	Name of Item	Display	Update uC	Update DSP
[0,*,*,*]		Volume: [volume]	Turn right ++ Turn left --	setVolume(dBarray(volume), device address, target address, step address))
[1,1,0,0,0]	Speaker Setup	Speaker Setup	-- Sub Mode (mono bass directed to left output channel) implemented using filter readin version of code using command SUBMODE -- Default is stereo woofer mode.	
[2,1,1,0,*]	Bi-amp set	Bi-amp		
[3,1,1,1,*]	Confirm Bi-amp	Confirm Yes	Set operating mode to 0 Call 2 back functions	On press ->Call NOSUB_download
[3,1,1,2,*]	Confirm Bi-amp	Back	Call back function	

[2,1,2,*,*]	Stereo + Sub set	Stereo + sub		
[3,1,2,1,*]	Confirm S+sub	Confirm Yes	Set operating mode to 1 Call 2 back functions	On press->Call subModeEnable(mute/gain addresses here);
[3,1,2,2,*]	Confirm S+sub	Back	Call back function	
[2,1,3,*,*]	Back out Speaker setup	..	On press ->Call Back function	
[1,2,*,*,*]	Crossover	Crossover		
[2,2,1,*,*]	Filter type [active]	Filter type: [active**] (string array of types)	Read active filter type as shown E.G. Crossover Type: LR24	
[2,2,1,0,*]	Update Active filter type, push turn state	.. Update crossover type index variable	If button pressed ->Update active type global, call calculate crossover >calculate variables, after safeload call back^^	Call: setCrossover(freq, 8 biquad filters) Requires each biquad to be initialized as such: LP1L ->setBiquad(bq_type_lowpass LR, DEFAULT_CROSSOVER_FC / sampleRate, 1, 0);
[2,2,2,*,*]	Frequency	Frequency: [active**]	Read crossover freq array index	
[2,2,2,0,*]	Update crossover fc, push turn state	Index ++ or -- [crossover.freq[index]]	If button pressed ->Update crossover freq array index, -> call setCrossover call back^^	Call: setCrossover(freq, 8 biquad filters)
[2,2,3,*,*]	Crossover back out	Back	Call back function ^^	
[1,3,*,*,*]	Equalizer	Equalizer		
[2,3,1,*,*]	Filter 1	Filter 1		
[3,3,1,1,*]	Filter 1 – Type **	Filter 1: filter1-> setType	Read filter type, send enumerated type to print function On push turn: set new type	safeLoadFilter(filter1, address of first destination coeff)
[3,3,1,2,*]	Filter 1 - Q **	Filter 1: filter1-> setQ	Read q index, send Q value to screen On push turn: increment Q index	safeLoadFilter(filter1, address of first destination coeff)

[3,3,1,3,*]	Filter 1 - Freq **	Filter 1: filter1-> setFc	Read Fc index, send Fc value to screen On push turn: increment Fc index	safeLoadFilter(filter1, address of first destination coeff)
[3,3,1,4,*]	Filter 1 - Peak Gain	Filter 1: filter1-> setPeakGain	Read gain index, send gain value to screen On push turn: increment gainLookup index	safeLoadFilter(filter1, address of first destination coeff)
[3,3,1,5,*]	Filter 1 - Back	Filter 1 - Back	Call back function Write filter 1 index variables to file	
<i>Repeat</i>	<i>structure</i>	<i>for filters</i>	<i>2 through 6</i>	<i>Adjust [*,*,X,*,*] where X = filter number</i>
[1,4,*,*,*]	Speaker Protection	Speaker Protection		
[2,4,1,*,*]	Limiter	Limiter		
[3,4,1,1,*]	Woofer Limiter	WF Limit: [wf_lim_i -6]dB	Read WF limit index	
[3,4,1,1,0]	Woofer Limiter Adj	..	If push turn– update wf_lim_i within range [-40 dB to 0 dB] corresponding to dB array Else-Call back	Read index, utilize dB lookup table., setLimiter(dBLookup[limiterIndex-15], DEVICE_ADDR_IC_1, MOD_LIMITERRIGHT_LOW_ALGO_THRESHOLD_ADDR, MOD_LIMITERLEFT_LOW_ALGO_THRESHOLD_ADDR);
[3,4,1,2,*]	Tweeter Limiter	TW Limit: [twLimitdB_i-6]dB		
[3,4,1,2,0]	Tweeter Limiter Adj, push turn statej	..	If push turn– update tw_lim_i within range [-40 dB to 0 dB] corresponding to dB array Else-Call back	setLimiter(dBLookup[limiterIndex-15], DEVICE_ADDR_IC_1, MOD_LIMITERRIGHT_LOW_ALGO_THRESHOLD_ADDR, MOD_LIMITERLEFT_LOW_ALGO_THRESHOLD_ADDR);
[3,4,1,3,*]	Limiter Back	Back	*	*
[4,4,1,3,1]	Limiter Back [SEL]	Back	Call Back Twice	
[2,4,2,*,*]	Protection HP, set in bootup	Low Cut: [20 + 5*HP_prot_i]	Read HP_prot_i	
[2,4,2,0,*]	Woofer HP Push turn state	HP_prot_i ++, update filter index	++,ii update	safeLoadHP (hpLookup[HP_prot_i])

[2,4,3,*,*]	Speaker Pro Back	Back	*	*
[1,5,*,*,*]	Top Back	Back	If pressed - call back -> volume	

**This parameter operates using push turn, using is_pressed, and is_released in combination with turn direction to determine parameter updates.

^Italic states indicate menu states not implemented in actual system either for memory reasons or because they were deemed unnecessary.

Active Parameter Update - Safe Load

Safe load is the soft write functionality enabled on Analog Device DSP boards that works by using 10 dedicated registers to temporarily store 5 parameters worth of data and their respective memory locations. The main control register contains a bit specifically for toggling the safe load download which essentially tells the DSP to transfer the bytes in the safe load data registers into the locations designated by the values stored in the safe load address registers at the next free moment the processor has so as to not disturb the audio stream. These 5 register writes are enough to update a single Biquad and filters that require multiple Biquads simply repeat this loading process as many times as needed. For instance, the crossover update requires 8 such safe load writes because it requires 8 Biquads to update a 2 channel 4th order crossover. One important thing to note about the DSP is that it stores Biquad coefficients A1 and A2 inverted in memory and as such those values must be inverted prior to being sent to the board. Furthermore the DSP stores all variables in 5.23 fixed point format in its 28 bit registers and thus it is important for the safe write register function to convert input coefficients to the correct data format, and then manually split the bytes for data transmission.

In completing a safe load write the following write order must be followed: upper byte safe load data register, lower byte safe load data register, 0x00, 4 bytes of parameter data MSB to LSB, send data, safe load address register upper byte, safe load address register lower byte, destination register upper byte, destination register lower byte, transmit data^[20]. We tested this coefficient conversion and transmission process using a simple on/off command (1 coefficient written) followed by a full filter update using both pre-generated coefficient data and Biquad calculated parameters.

Memory Allocation

The arduino and most like-costed microcontrollers have a limited amount of local variable memory for use in executing functions and large arrays such as the DSP's Param_Data tend to exceed the allotted 2 kB of variable memory..The program data and parameter memory required to boot the DSP board must be stored in progmem (ROM storage within the Arduino) and extracted using special pgm_read functions to avoid dynamic memory overflow. Specifically, the DSP program and data registers must be stored in progmem because the boot sequence reads these 2 primary arrays that control DSP core function over I²C and writing them without

splitting it into pieces and progmem storage causes dynamic memory issues. In order to overcome dynamic memory limitations, each register of program memory and parameter memory is written one at a time with it's respective number of bytes using separate functions(5, 4 in order). The DSP boot sequence requires approximately 6.5 kB of memory which is a large portion of the total system code, next to the menu in its entirety.

In developing the user interface and system overall, a number of challenges derive from the Arduino's limited code space (32 kB). In order to have enough memory to run Serial.print, the primary testing and verification procedure for filter calculation and update, the display was disabled, or sections of the menu - via comment. The end decision to implement two separate firmwares resulted from the thought that there should be an advanced - computer filter load firmware with a very limited menu of volume and crossover frequency control, and a more in depth - live menu edit mode. With approximately 128 kB of memory, both of these functional systems would run together and provide both user presets, Room Equalizer Wizard integration, and other advanced functionality. To expand the functionality of the system while remaining under 85% uC memory usage (above which menu overflow prevents the DSP from booting), both halves of the system utilize separate microcontroller and DSP code. See appendix B for information regarding filter format.

Enclosure Design

The enclosure design utilized cardboard mockup followed by SolidWorks modeling. MDF was chosen as the material of choice because it is easy to cut and takes a duratex finish (patterned black paint with enhanced durability used in professional audio gear). See Figure 4.8 below for determining required width and depth using an arbitrary box (height constrained by power supply and wire clearance).

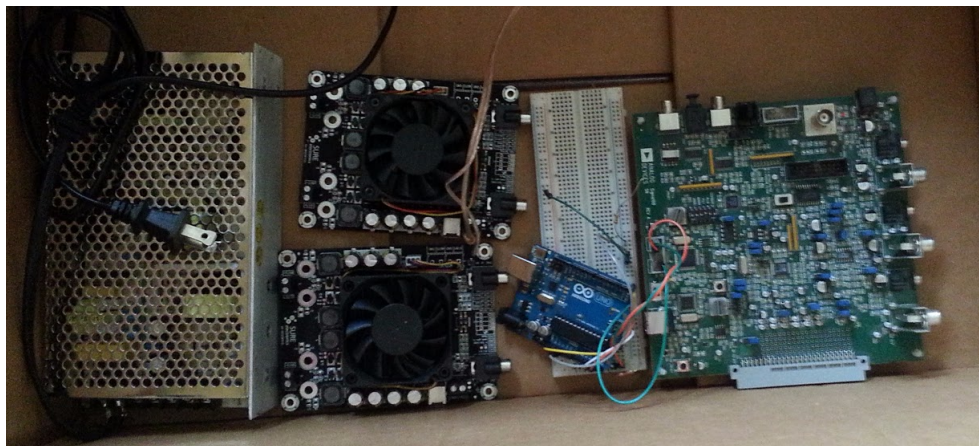


Figure 4.8: Cardboard Mock-up For Enclosure Design

The following box was designed for MDF ¼ inch material using the following model (multiple views included).

Figures 4.9-4.11 show the Solidworks modeling of the enclosure.

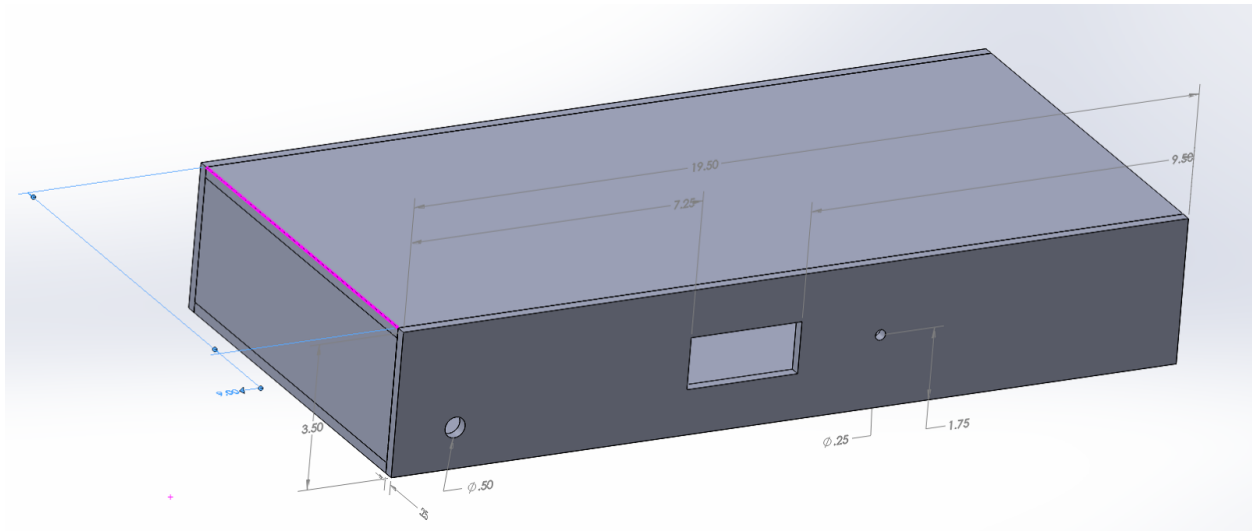


Figure 4.9: Front View Enclosure Model

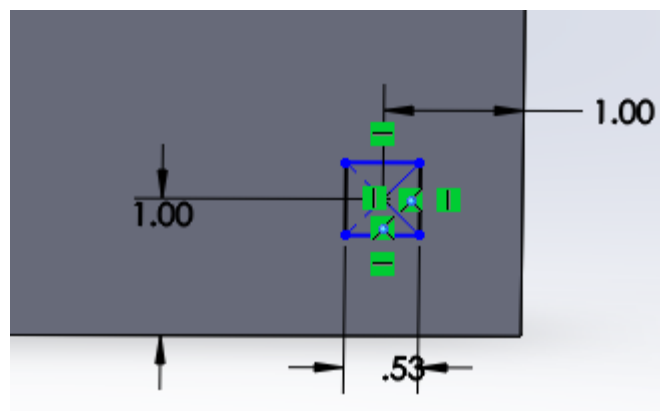


Figure 4.10: Power Cord Recess Dimensions

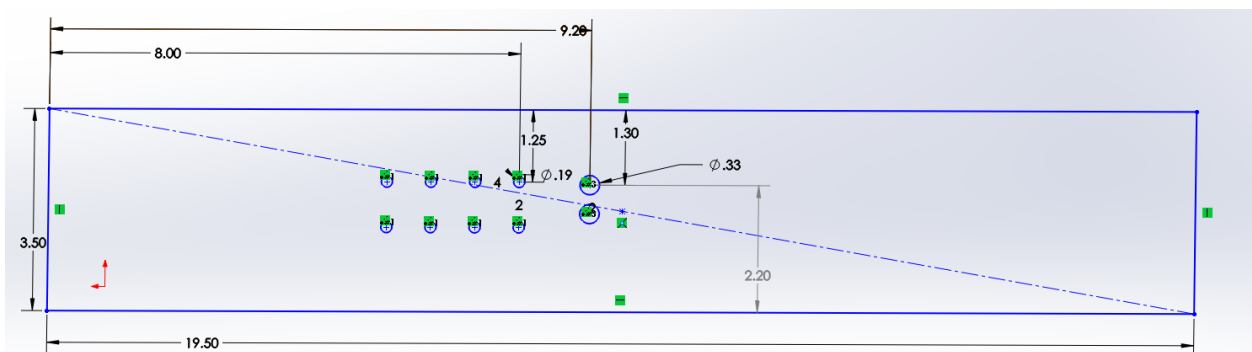


Figure 4.11: Enclosure Back Panel

The enclosure is finished using Duratex speaker coating for durability and its sleek look (see Figure 4.12 below). The holes for speaker terminals, display cutoff, and power switch are machined using a combination of CNC routing and a drill press.

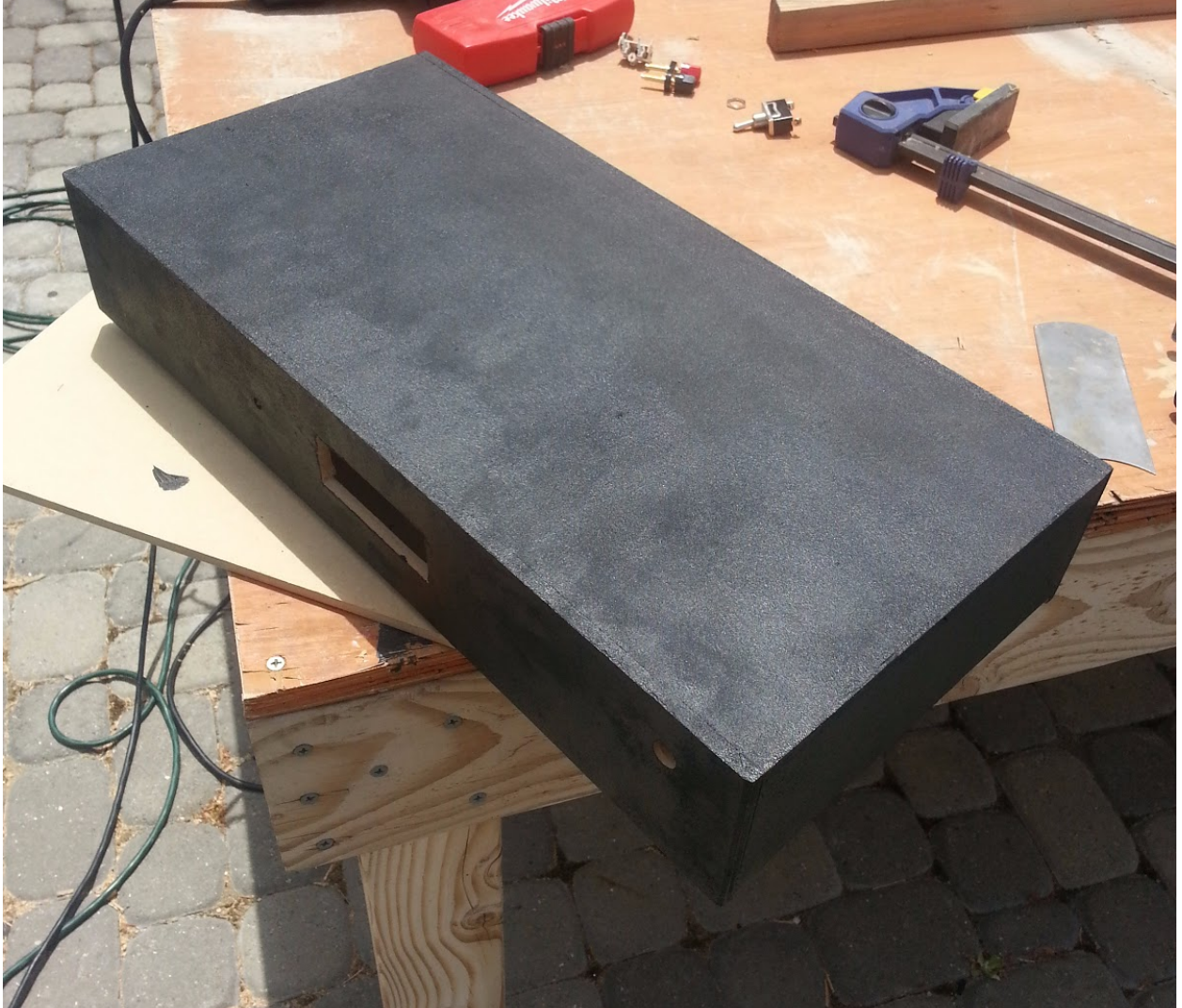


Figure 4.12: Painted Amplifier Enclosure

Chapter 5: Testing

I²C Communication Testing

Verifying I²C Bus Connection

The first step necessary to ensure communication between the ATmega328p and the ADAU1702 DSP board is to establish an I²C bus. Using a pre-written Arduino sketch called I2CScanner.ino (see Appendix B for code below), device connection can be observed via the serial monitor on the computer. Figure 5.1 below displays the serial monitor recording of I²C bus communication along with the addresses of each device. The I²C device address for the ADAU1702 DSP board is identified as 0x34.

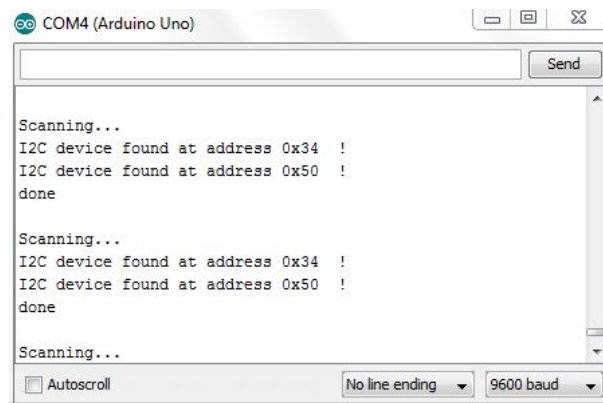


Figure 5.1: I²C Bus Verification

ADAU1702 Sigma Studio Boot Sequence - Audio Output

Before testing that the microcontroller can boot the DSP board via I²C, the board is first tested using the ADAU1702 Sigma Studio software GUI to verify that audio can be received and outputted from the DSP board. Figure 5.2 shows the software setup GUI for receiving, processing and outputting audio signals. The DSP correctly filters the signal when booted from the USB connection.

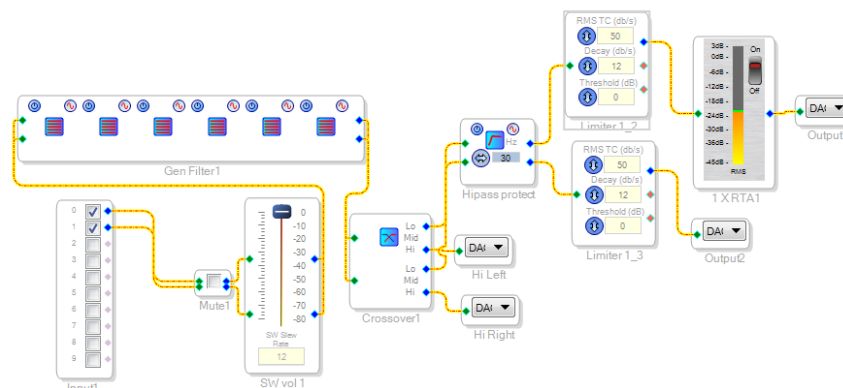


Figure 5.2: Sigma Studio Basic Audio Adjustment GUI Interfacing

Figure 5.2 displays the signal chain of the test system. The bank of filters controls the EQ, high-pass at 30 Hz with the limiter protects the woofers, and the volume analyzer monitors the signal level for testing purposes.

I²C Transmission Verification

In order for the boot sequence to be successfully received by the ADAU1702 via the arduino, the information sent over I²C must be verified to match the desired array values. Using an oscilloscope set for I²C transmission, the SCLK (clock) and SDA (data) signals were captured during data transmission to observe the hexadecimal values sent. Figure 5.3 below displays the first transmission I²C from the microcontroller to the ADAU1702.

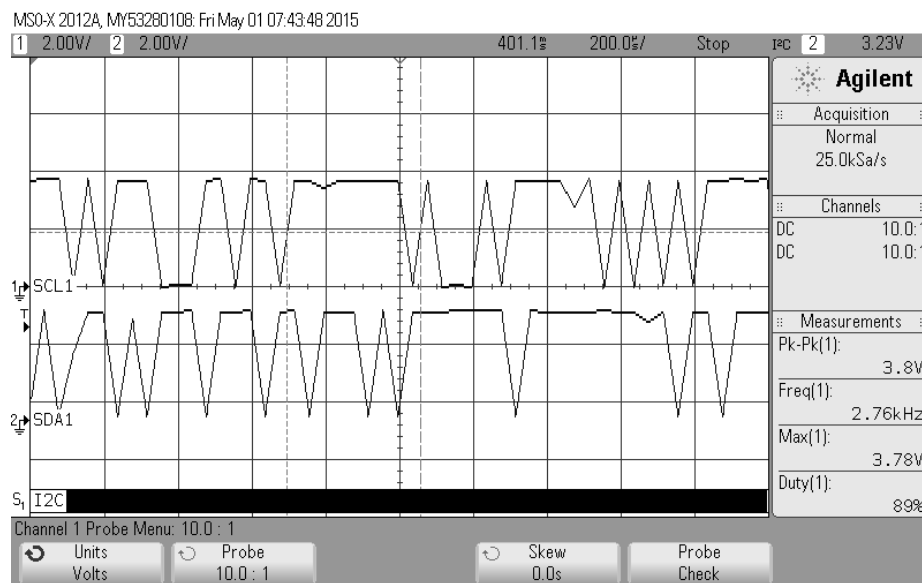


Figure 5.3: First Attempt at I²C Transmission

Note the jagged transmissions observed above. I²C needs to have very flat distinct high and low digital logic levels to ensure correct bit transmission. This problem was solved by establishing pull-up resistors to 3.3V instead of 5V. The ADAU1701 needs to operate on a 3.3V bus while the arduino operates on a 5V bus. Figure 5.4 below shows the transmission after changing the pull-up rail voltage.

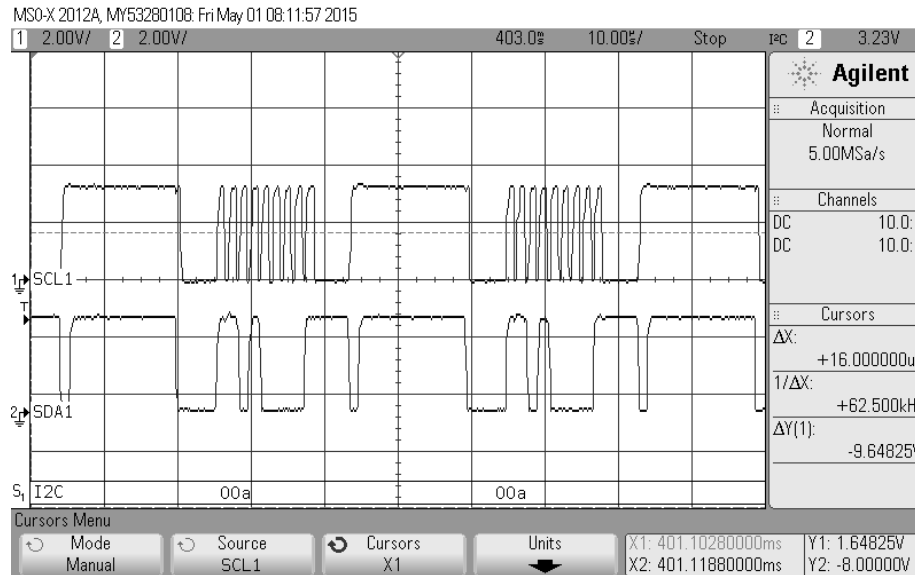


Figure 5.4: Second Attempt at I²C Transmission

After adjusting the rail to the appropriate 3.3V, the signal clarity improved drastically. Transmission is verified as usable, but the correct data is still unverified.

Boot Sequence Verification

In order for the arduino to turn on the DSP board, 1536 bytes of program data and parameter data must be sent to the board via I2C (see main_showcase_IC_1.h in the appendix for exact arrays). To avoid dynamic memory overflow, these arrays are stored in flash memory using the pgmspace.h library. By type defining the large Program_Data_IC_1 and Param_Data_IC_1 as PROGMEM, the arduino can avoid attempting to write the entirety of both arrays during I2C transmission. However, the arduino can only extract one byte at a time when using PROGMEM. Using appropriate logic and pointers to the location of the desired arrays, the necessary information can be extracted from PROGMEM and sent to the DSP board.

The boot sequence function (default_download_IC_1) is broken down into 3 different functions described below:

SIGMA_WRITE_REGISTER_PROG: The program data must be sent in 5 byte increments.

SIGMA_WRITE_REGISTER_PARAM: The parameter data must be sent in 4 byte increments.

SIGMA_WRITE_REGISTER_BLOCK: Register setting data that is 1 byte only.

The core register of the ADAU1702 must first be set for data transmission using SIGMA_WRITE_REGISTER_BLOCK. The program data and parameter data is then loaded using SIGMA_WRITE_REGISTER_PROG and SIGMA_WRITE_REGISTER_PARAM.

After the parameter data is fully loaded, the core register must be loaded with a few more bytes using SIGMA_WRITE_REGISTER_BLOCK. The boot sequence is verified to work by powering both the arduino and the DSP board with 7VDC, providing an input audio signal on input 0/1 and listening to the audio source transmit with the firmware defined lowpass (implemented with the crossover) on the DAC output 0/1 through headphones.

Figures 5.5 and 5.6 below illustrate the boot sequence Program and Param data being sent over I2C as monitored using Serial.print commands. Note that serial print prints hex values without the 0x prefix and ignores leading 0's.

The screenshot shows an IDE with several files open: EXPERIMENT, Biquad.cpp, Biquad.h, LCDDriver.h, LCD_Driver.cpp, RotaryEncoder.cpp, RotaryEncoder.h, SigmaStudioFw.h, defines.h, main_final_IC_1.h, main_final_IC_1_PARAM.h, and main_final_IC_1_REG.h. The code in the editor includes a loop for writing program data to registers. A serial monitor window titled 'Snipping Tool' is open, showing the transmitted data for 'COM11'. The data is a list of hex values representing program data, starting with '0, 0, 0, 0, 1' and ending with '0, 31, 8, 20, 1'. The serial monitor also shows 'Done uploading' and 'Autoscroll' options.

```

185     addtemp = address;
186     #ifndef cbi
187     #define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
188     #endif
189     for(regcount=0;regcount<length_reg;regcount++)
190     {
191         Wire.beginTransmission(devAddress);
192         Wire.write(((addtemp+regcount) & 0xFF00) >> 8);
193         Wire.write((addtemp+regcount) & 0x00FF);
194         Serial.print(pgm_read_byte(pData+addcount),HEX);
195         Serial.print(" ");
196         Serial.print(pgm_read_byte(pData+addcount+1),HEX);
197         Serial.print(" ");
198         Serial.print(pgm_read_byte(pData+addcount+2),HEX);
199         Serial.print(" ");
200         Serial.print(pgm_read_byte(pData+addcount+3),HEX);
201         Serial.print(" ");
202         Serial.print(pgm_read_byte(pData+addcount+4),HEX);
203         Serial.println("");
204         Wire.write(pgm_read_byte(pData+addcount));
205         Wire.write(pgm_read_byte(pData+addcount+1));
206         Wire.write(pgm_read_byte(pData+addcount+2));
207         Wire.write(pgm_read_byte(pData+addcount+3));
208         Wire.write(pgm_read_byte(pData+addcount+4));
209         addcount+=5;
210     }
211 }
212 }
213 //Writes parameter memory in 4 byte chunks
214 void SIGMA_WRITE_REGISTER_PARAM(int devAddress, int address, int length)
215 int addtemp = 0;

```

```

66 /* DSP Program Data */
67 #define PROGRAM_SIZE_IC_1 2560
68 #define PROGRAM_SIZE_IC_2 512 // Number of Prog registers to write during boot-up
69
70 #define PROGRAM_ADDR_IC_1 1024
71
72 ADI_REG_TYPE Program_Data_IC_1[PROGRAM_SIZE_IC_1] = {
73 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
74 0x00, 0x00, 0x00, 0xE8, 0x01,
75 0x00, 0x00, 0x00, 0x00, 0x01,
76 0x00, 0x00, 0x00, 0xE8, 0x01,
77 0x00, 0x02, 0x00, 0x20, 0x01,
78 0x00, 0x10, 0x00, 0xE2, 0x01,
79 0x00, 0x0A, 0x00, 0x20, 0x01,
80 0x00, 0x18, 0x00, 0xE2, 0x01,
81 0xFF, 0xF2, 0x01, 0x20, 0x01,
82 0x00, 0x21, 0x08, 0x22, 0x41,
83 0x00, 0x40, 0x00, 0xE2, 0x01,
84 0x00, 0x31, 0x00, 0x20, 0x01.

```

```

Program Data Begin
0, 0, 0, 0, 1,
0, 0, 0, E8, 1,
0, 0, 0, 0, 1,
0, 8, 0, E8, 1,
0, 2, 0, 20, 1,
0, 10, 0, E2, 1,
0, A, 0, 20, 1,
0, 18, 0, E2, 1,
FF, F2, 1, 20, 1,
0, 21, 8, 22, 41,
0, 40, 0, E2, 1,
0, 31, 8, 20, 1.

```

Done uploading.

Sketch uses 25,664 bytes (79%) of program storage space. Maximum is 32,256 bytes.

Global variables use 725 bytes (35%) of dynamic memory, leaving 1,323 bytes for local variables. Maximum is 2,048 bytes.

Figure 5.5: Program Data Serial.print Transmission Test

The screenshot shows an IDE with files: EXPERIMENT, Biquad.cpp, Biquad.h, LCDDriver.h, and LCD_Driver.cpp. The code defines parameters for parameter data transmission. A serial monitor window titled 'COM11' is open, showing the transmitted data for 'Param Data Begin'. The data is a list of hex values representing parameter data, starting with '0, 80, 0, 0' and ending with '0, 80, 0, 0'. The serial monitor also shows 'Send' and 'Autoscroll' options.

```

593 #define PARAM_SIZE_IC_2 1024
594 #define PARAM_ADDR_IC_1 0
595 ADI_REG_TYPE Param_Data_IC_1[PARAM_SIZE_IC_1] = {
596 0x00, 0x80, 0x00, 0x00,
597 0x00, 0x80, 0x00, 0x00,
598 0x00, 0x00, 0x08, 0x00,
599 0x00, 0x80, 0x00, 0x00,
600 0x00, 0x00, 0x00, 0x00,
601 0x00, 0x00, 0x00, 0x00,
602 0x00, 0x00, 0x00, 0x00,
603 0x00, 0x00, 0x00, 0x00,
604 0x00, 0x80, 0x00, 0x00,
605 0x00, 0x00, 0x00, 0x00,
606 0x00, 0x00, 0x00, 0x00,
607 0x00, 0x00, 0x00, 0x00,
608 0x00, 0x00, 0x00, 0x00,
609 0x00, 0x80, 0x00, 0x00.

```

```

Param Data Begin
0, 80, 0, 0,
0, 80, 0, 0,
0, 0, 8, 0,
0, 80, 0, 0,
0, 0, 0, 0,
0, 0, 0, 0,
0, 0, 0, 0,
0, 0, 0, 0,
0, 80, 0, 0,
0, 0, 0, 0,
0, 0, 0, 0,
0, 0, 0, 0,
0, 0, 0, 0,
0, 0, 0, 0,
0, 80, 0, 0.

```

Figure 5.6: Param Data Serial.print Transmission Test

Once the DSP correctly boots, audio transmits to the output DACs and whatever baseline filters specified in the exported SigmaStudio project code are implemented. The next important step in communicating with the DSP is safe loading parameters, the process of updating filters without audible artifacts, necessary for all real time control and parameter updates. The testing of safe load is contained within filter read in as each filter is transmitted using safeLoadFilter (each call loads the 5 parameters pertaining to that particular filter into the corresponding stage in parameter memory).

Filter Import Testing

In designing the filter import code, we discovered that since Arduino doesn't have an operating system it cannot access files on a hard drive and the only file access it can understand requires additional hardware (SD shield). At this point the system doesn't have the I/O to accommodate a shield and the time to integrate one was lacking. However, in order to prove that such a system would work, we developed a filter load and parse function that utilizes EEPROM to store the text file in internal memory. The restrictions are that it has to be accessed on a per-byte basis as opposed to using standard C functions such as fopen and fgets. As implemented the function reads characters from EEPROM until it reaches a return character (placed at each end of line for this purpose), places a null character at the end of the string, and parses that line by using spaces to delimit. This method requires the filter data to be loaded from strings in a separate project file and as such isn't viable beyond testing and advanced user purposes. See Figure 5.7 below for an overview of how the parsing function works:

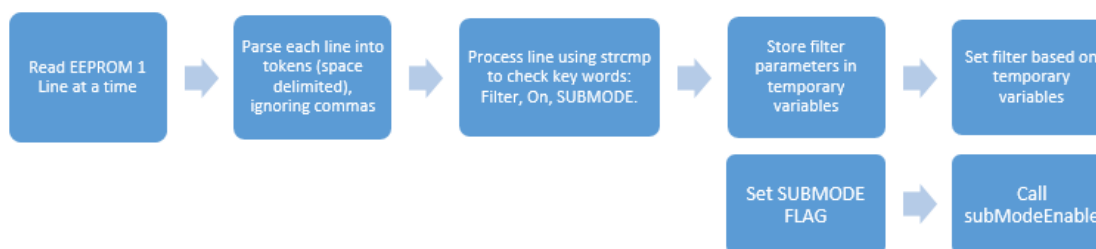


Figure 5.7: File Parsing Process Flow Chart

First the parsing system must be verified to see if the lines are correctly being read from memory and don't contain junk data. At first test, the buffered lines Serial.print displayed garbage data following the last character of the line because it lacked a null character and thus was printing to the end of the allocated memory. Furthermore, arduino through couldn't find errors when trying to include certain C++ libraries required for parsing files. To work around this restriction the parse code was rewritten using basic C commands including atof to convert strings to doubles and while loops instead of fgets. Once null characters were added to the buffer string, the code correctly parsed as displayed in the following figure. In Figure 5.8 below each string is printed followed by the individual tokens (words, separated by commas).

```

COM7 [Arduino Uno]
3.0
dB
4.00
-----
53.3
Finished
Filter 2: ON Modal Fc 56.0 Hz Gain -3.6 dB Q 7.90 T60 target 300 ms
Filter
2:
ON
Modal
Fc
56.0
Hz
Gain
-3.6
dB
Q
7.90
T60
target
300
ms
Filter 3: ON LS 12dB Fc 75.4 Hz Gain 3.0 dB
Filter
3:
ON
LS
12dB
Fc
75.4
Hz
Gain
3.0
dB
Filter 4: ON HS 12dB Fc 7,710 Hz Gain 2.5 dB
Filter
4:
ON
HS
12dB
Fc
7,710
Hz
Gain
2.5
dB
SUBMODE
SUBMODE
Sub Mode is
1
Filter 6: ON Sine
Filter
6:
ON
SUBMODE
SUBMODE
Sub Mode is
1
Autoscroll No line ending 9600 baud

EXPERIMENT | Arduino 1.6.0
file Edit Sketch Tools Help
EXPERIMENTS | Bossard.cpp | Biquad.h | LCDDriver.h | LCD_Driver.cpp | RotaryEncoder.cpp | RotaryEncoder.h | SigmaBadoFw.h | ...
101
102 if(!strcmp(token[2], "ON")) //filter is enabled
103
104 if(!strcmp(token[3], "P")) // Peak filter
105 // Serial.println("Here....."); // Peak filter
106 // Serial.println(token[5]);
107 fcTemp = atof(token[5]);
108 // Serial.println(fcTemp);
109 // Serial.println("Passband");
110 gainTemp = atof(token[8]);
111 qTemp = atof(token[11]);
112 type = bq_type_peak;
113 }
114 else if(!strcmp(token[3], "LS"),2) // lowshelf
115 qTemp = 1;
116 fcTemp = atof(token[4]);
117 gainTemp = atof(token[9]);
118 type = bq_type_lowshelf;
119 }
120 else if(!strcmp(token[3], "HS"),2) // highshelf
121 qTemp = 1;
122 fcTemp = atof(token[4]);
123 gainTemp = atof(token[9]);
124 type = bq_type_highshelf;
125 }
126 else if(!strcmp(token[3], "HP"),2) // highpass
127 qTemp = 1;
128 gainTemp = 0;
129 fcTemp = atof(token[5]); // change to index 6 if include 120B/24B in input file
130 type = bq_type_highpass;
131 }
}
}

Sketch uses 25,976 bytes (80%) of program storage space. Maximum is 32,256 bytes.
Global variables use 764 bytes (37%) of dynamic memory, leaving 1,284 bytes for local variables. Maximum is 2,048 bytes.
  
```

Figure 5.8: String Parse Test

The next step is utilizing these strings and their token location (fixed given correct formatting) to interpret filter instructions and set the correct filter parameters in the next available biquad. The following figure displays filter coefficients and parameters being compared to the parsed and imported ones. These parameters being correct verifies that the filters being generated (including the string conversion process using the `atof` function) behave the same as manually set filters using fixed parameters. The image also shows that `SUBMODE` flag properly goes high based on the `SUBMODE` command occupying one of the filter lines. `SUBMODE` is then audibly confirmed by listening to the woofer outputs and noting that a mono bass signal is coming out of a single channel.

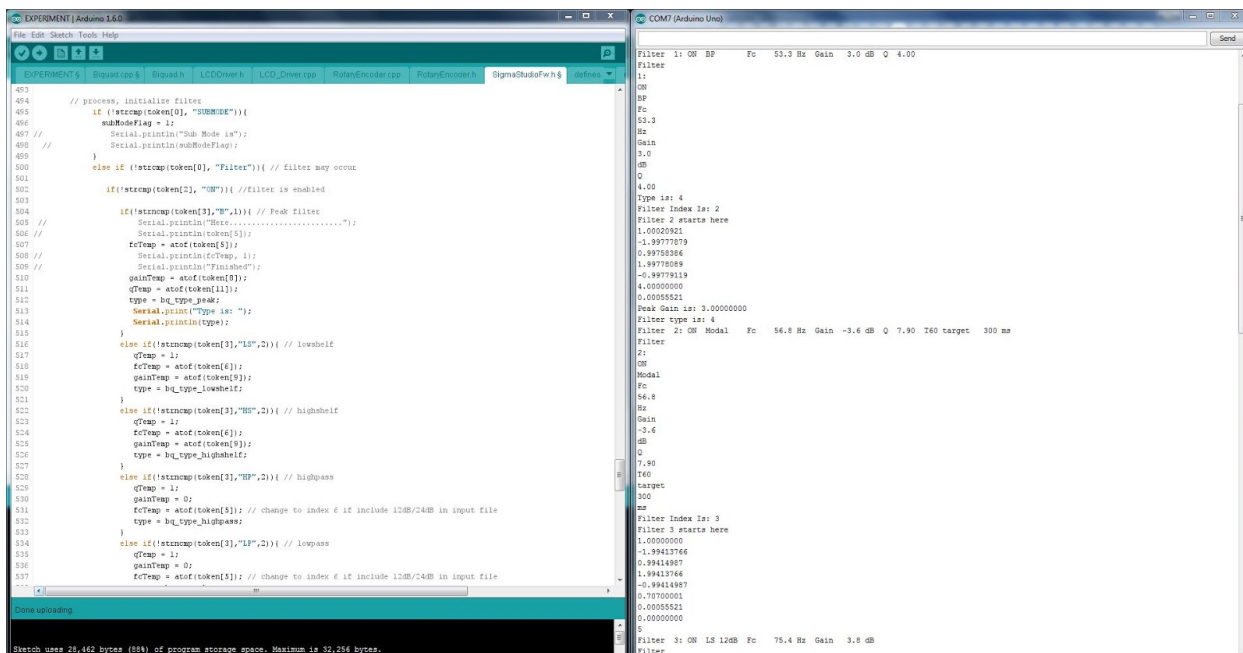


Figure 5.9 Filter String Import With Parameter Decoding

To show that not only are the correct filter parameters being read from EEPROM but that the coefficients are being correctly calculated the below figure shows the coefficients sent to the DSP to implement a -10 dB parametric filter with a Q of 1 centered at 1 kHz over I2C. The serial monitor calculated coefficients: B0, B1, B2, A1, A2 of 0.97337179, -1.87919673, 0.90809135, 1.87674655, and -0.88391313 are within reasonable tolerance of the SigmaStudio reference values of (coefficients generated using the GUI software that implement the same filter) 0.9624, -1.886, 0.9276, 1.886, and -0.8900. See Figure 5.9 below for a visual verification.

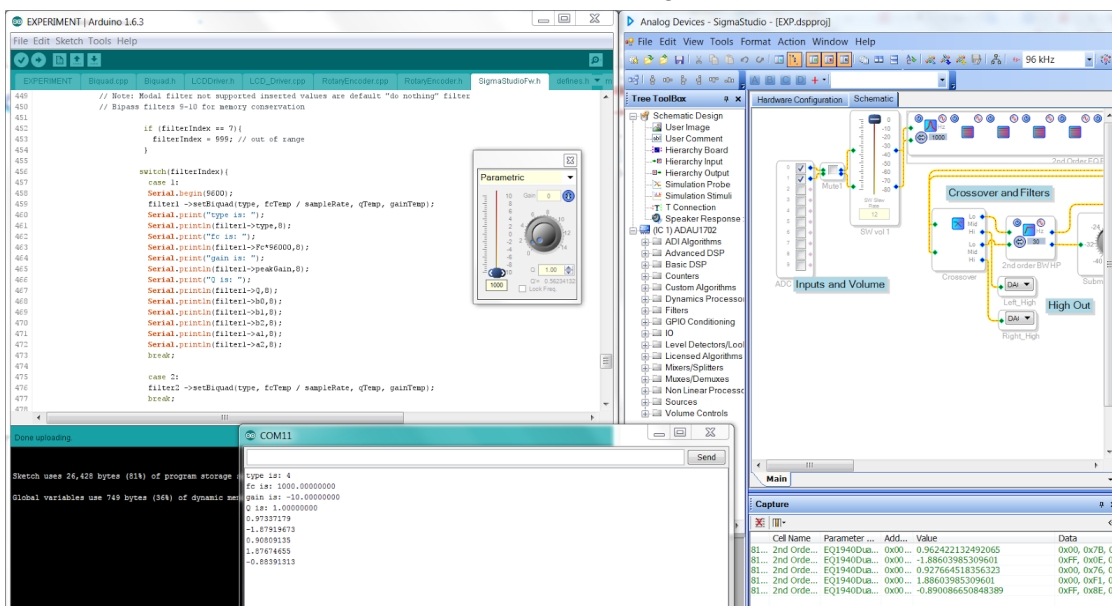


Figure 5.10: Parametric Coefficient Calculation Verification Read From EEPROM

Hardware and System Integration Testing

Testing the voltage regulator, power supply voltage, and switch polarity are all important prerequisites for connecting the amplifiers, DSP, and microcontroller together. The following table summarizes the tested output voltages for the above circuits.

Table 5.1: Hardware Test Output Voltage Summary

Circuit	Voltage (V)
DC Voltage Regulator (powers uC and DSP)	7.1008 DC
24V DC Power Supply	24.123 DC
AC Switch, Up Position	120.1 VAC

Once the power source voltages checked out and the regulator is tuned using the onboard potentiometer to 7V to power the microcontroller and DSP board the next step is to test connect the signal path starting at the DSP and ending at the amplifier banana terminals to ensure that connections and wires are secure. This test was quickly performed using a pass/fail metric and in the process a few loose wires required attention. See Figure 5.11 of this system integration process.

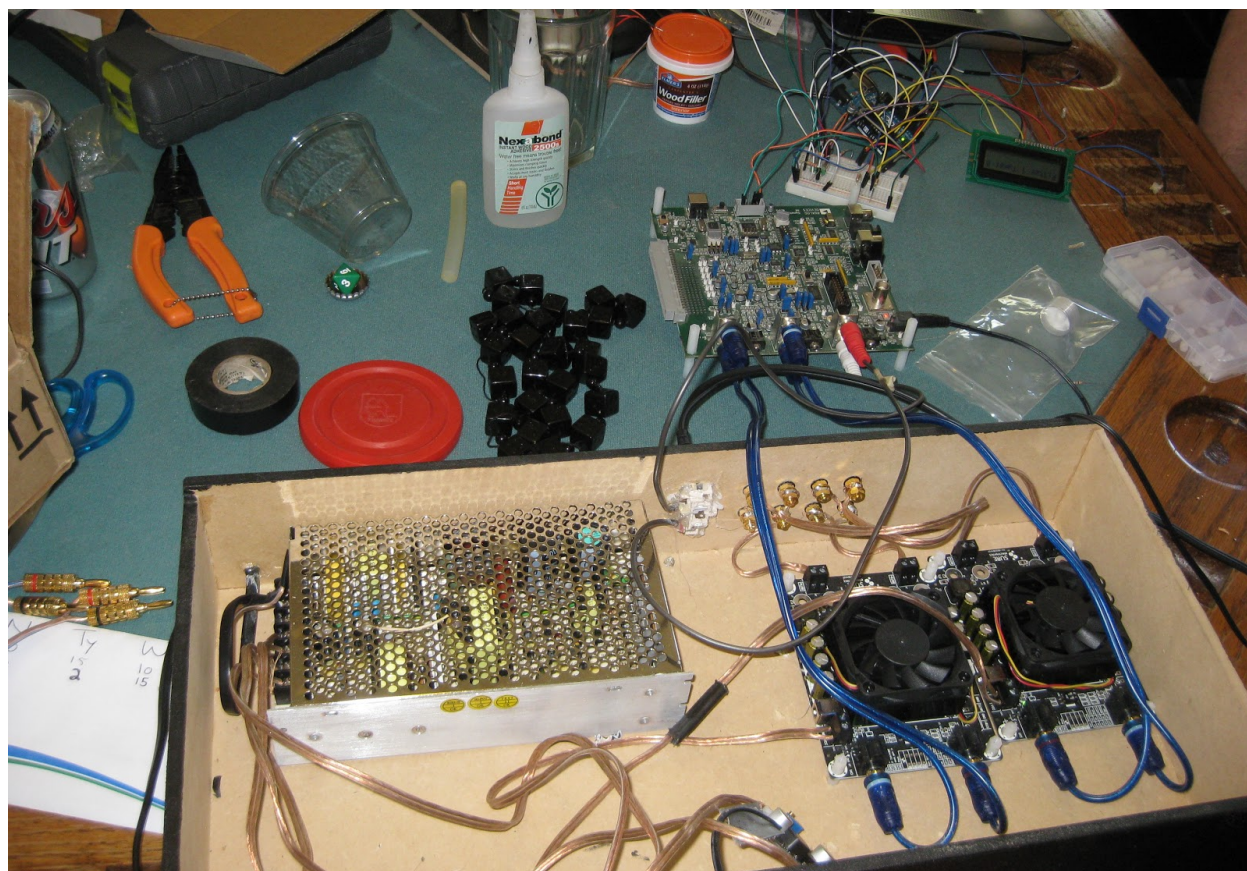


Figure 5.11: System Integration and Connection Testing

Filter Frequency Response Verification

Most software testing described in the previous section including filter implementation, signal verification, and data transmission required no more than headphones connected to the two DSP analog audio outputs. However, in order to verify that the filters properly affect filter response on the output terminals, oscilloscope measurements using white noise stimulus offer much more reliable data than simple hearing tests. The following images highlight the frequency response impact of the following filters using white noise input: 180 Hz low shelf with a gain of 4 dB, 400 Hz high shelf with a gain of -6 dB combined with 500 Hz low pass, 125 Hz high pass, and 300 Hz notch filter with a Q of 4.

Despite inaccuracies inherent to using white noise as an input source and the finite accuracy of measuring the FFT average value when it has 10 dB of dynamic range, the data measured correctly reflects the filters read in from EEPROM as shown in the Figure 5.12-5.15.

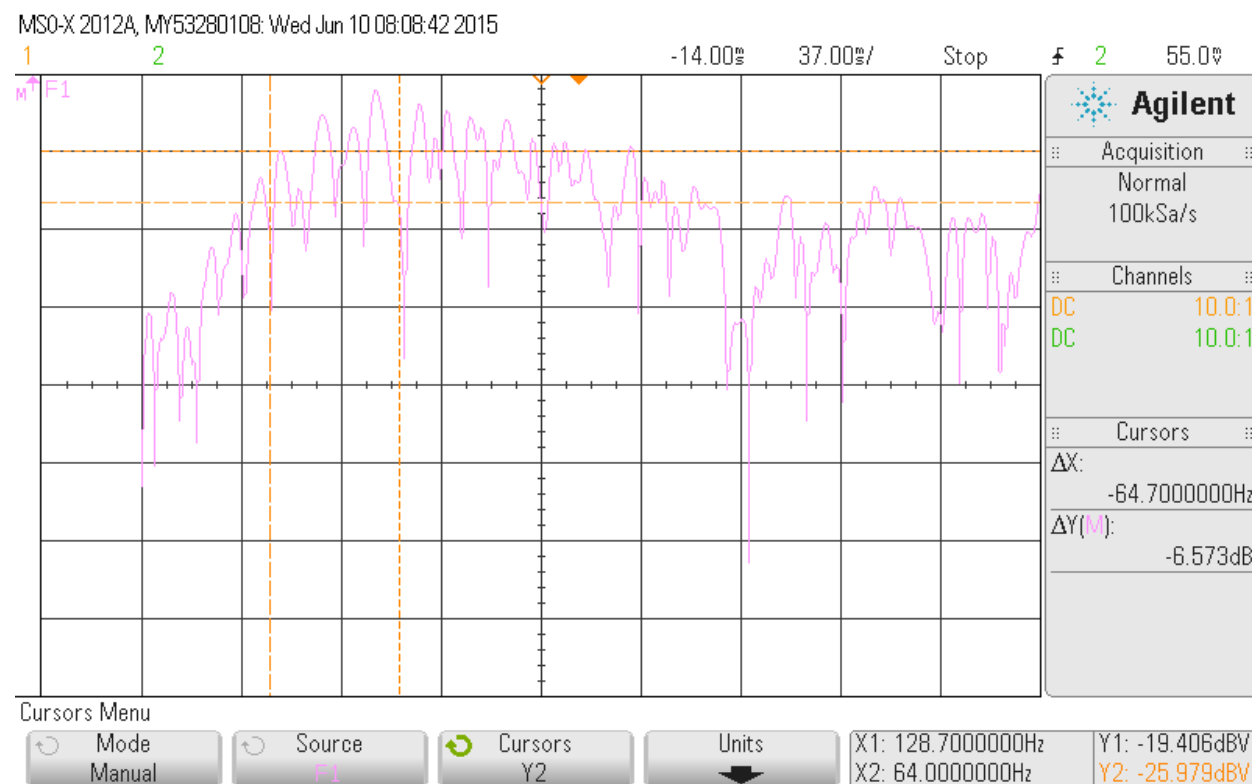


Figure 5.12: 125 Hz High Pass Illustrating a 6 dB/Octave 1st Order Slope Response should be down 6 dB, 1 octave below the cutoff frequency and it is within measurement tolerance at 6.57 dB.

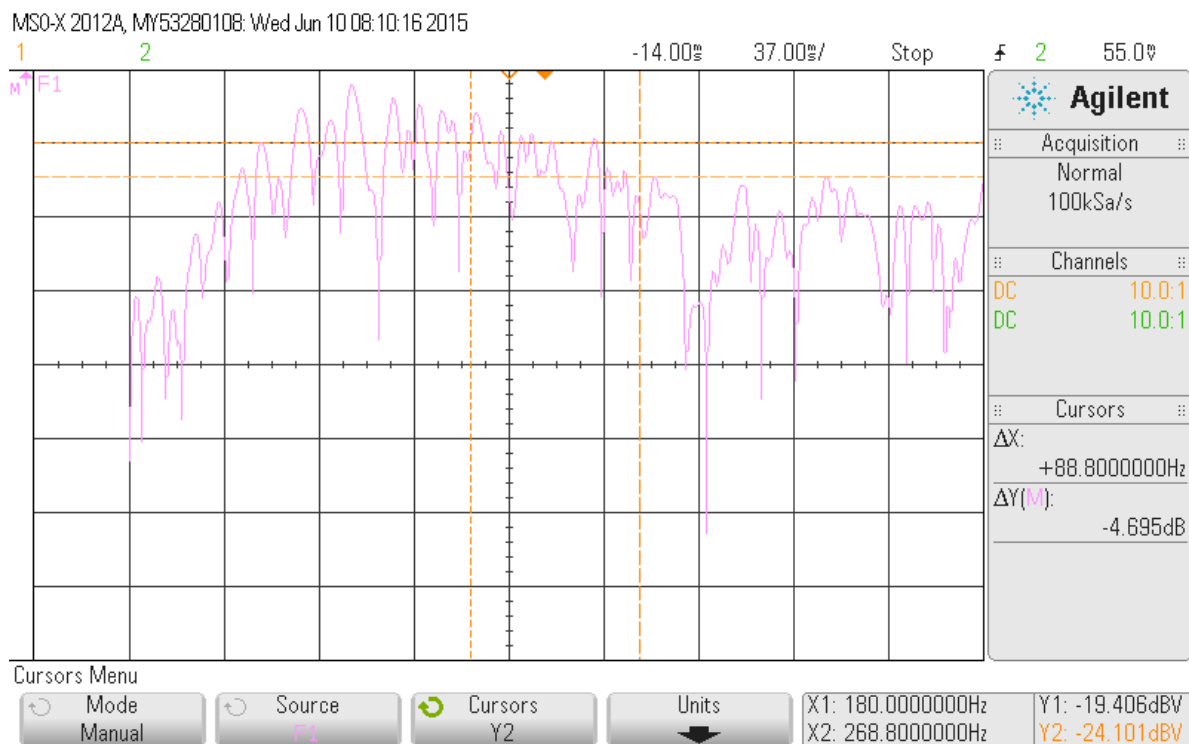


Figure 5.13: 4 dB Low Shelf with F_c of 180 Hz

Response is expected to be 4 dB greater within the passband of the low shelf filter and is measured at 4.69 dB in the above figure.

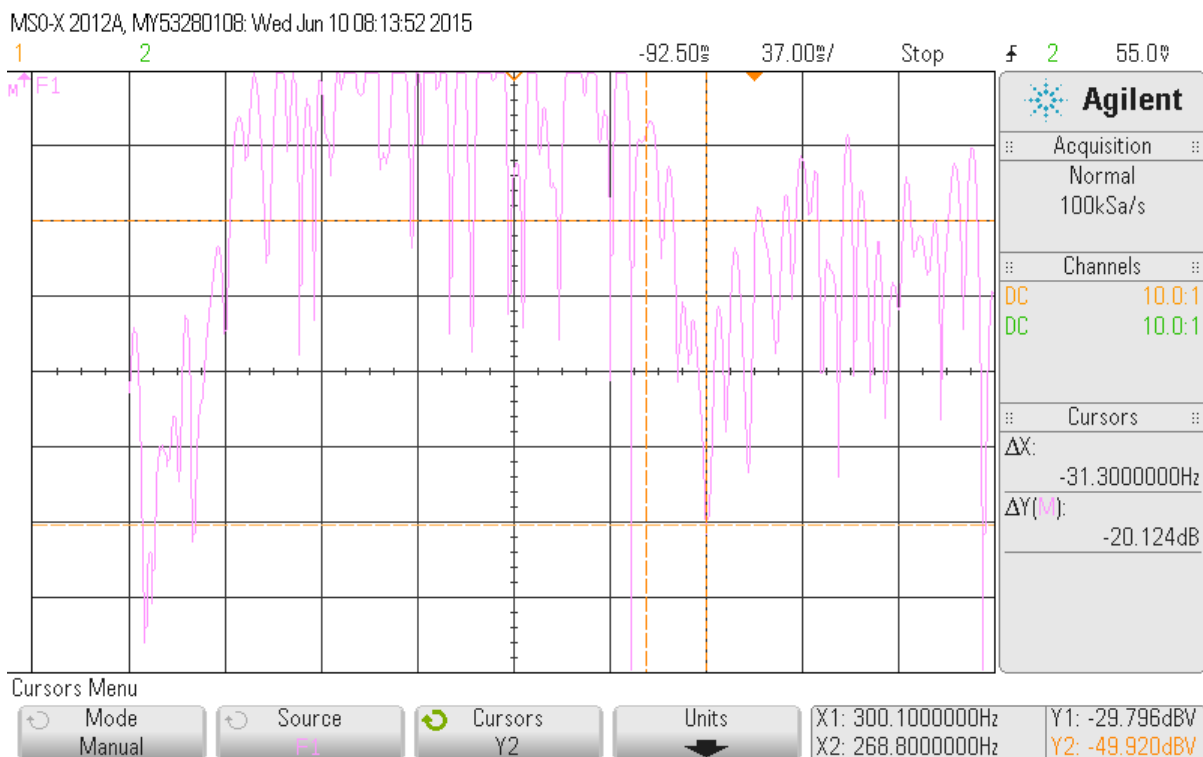


Figure 5.14: Notch Filter, F_c of 300 Hz, $Q=4$

Response (figure 5.14) measures 20 dB down at 300 Hz as expected and the narrow Q is reflected in the ~ 70 Hz bandwidth.

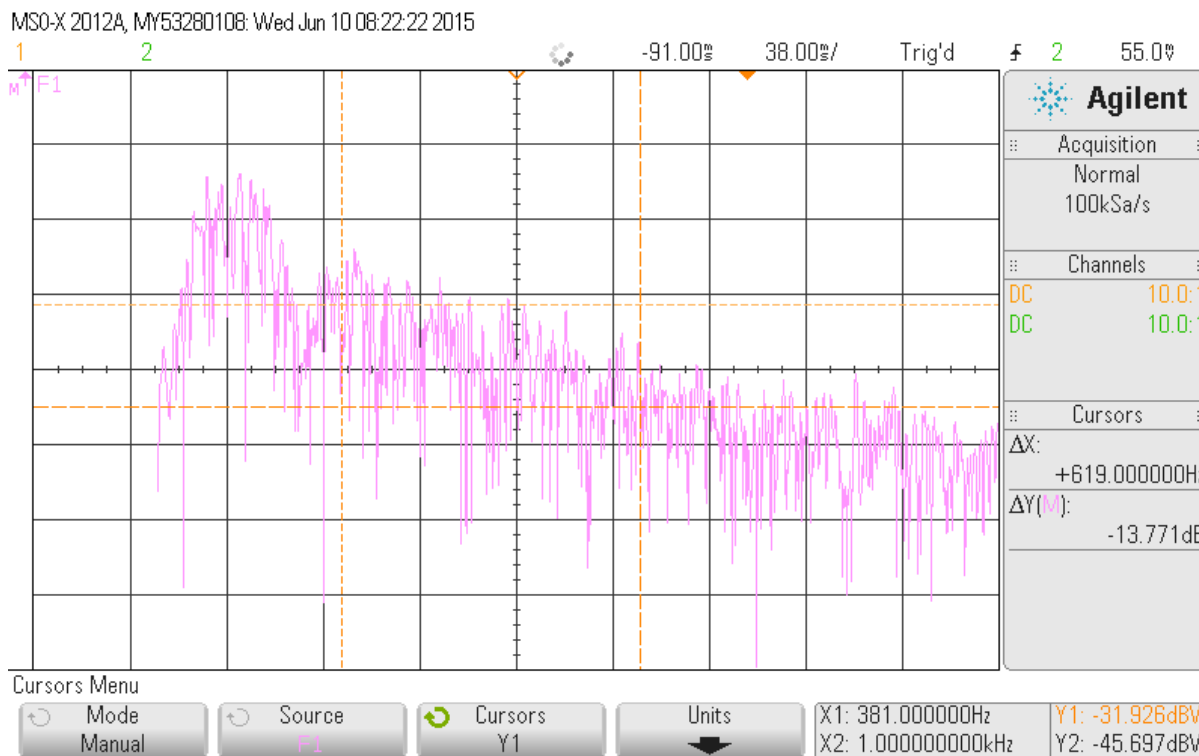


Figure 5.15: Lowpass at 500 Hz Compounded with -6 dB High Shelf Centered at 400 Hz

Response is expected to be -13 dB at 1 kHz due to the sum of -6 dB from a high shelf and -7 dB from being 1.25 octaves above the cutoff frequency, measuring quite close at -13.77 dB.

Altogether these tests verify that not only are we generating sufficiently accurate filter coefficients, but also that the resulting filters being implemented are accurate to within the measurement tools at our disposal.

Chapter 6: Conclusions and Discussion

The final system performs all of the key functions described in the original requirements but a combination of memory limitations and time constraints meant that the functional capabilities of the DSP are significantly limited by the menu and microcontroller interfacing. The code contains functions to set all of the implemented functional blocks but some of these aren't called because the menu couldn't be expanded to accommodate live editing of their parameters. For example, limiter gain setting is fully functional but not contained in the final menu, all 10 filters can be loaded but not without exceeding microcontroller dynamic memory, and setting the crossover filter type and slope is works but is disabled for user friendliness and menu constraints. The rest of this section explains the thought process behind what how select features were chosen to demonstrate with the limited memory available and where system performance and flexibility could be expanded given additional time and resources.

Hardware Choices

Arduino

After developing the system and attempting to load the entire project onto the arduino, it became apparent that significant memory overflow has occurred. The Atmega328p only has 32kB of flash memory and 2kB of dynamic memory. To avoid dynamic memory issues, PROGMEM is used as a means to circumvent the issue, which in turn causes new issues. At the cost of being able to push large arrays into flash memory, the pgmspace.h library can only extract one byte of these arrays at a time from flash memory. To transfer these arrays to the DSP board, the data transmission function had to be split into three separate functions, reading from specific address locations of each array.

In addition to dynamic memory limitations, the flash memory of the arduino reaches maximum flash memory limitations. Implementing 6 actively updateable parametric filters alongside the fully integrated menu is impossible with only 32kB of dedicated memory. The final prototype designed on the Atmega328p only has active volume and crossover frequency adjustments through the menu, while all 6 filters are statically loaded upon booting the DSP board. In order to change these filters, a separate file must be manually edited and loaded into the microcontroller's EEprom, which is restricted to 512 bytes. In the future, an arduino with significantly larger memory (such as the Arduino Mega [22]) would be used to incorporate the full active parameter adjustment menu.

Rotary Encoder

The rotary encoder is the ideal component for menu control for the Audio DSP Amplifier. Using open-source community written code modified by ourselves for this specific project, the rotary encoder successfully navigates the menu and actively updates filter parameters and volume. Using a rotary encoder is preferred over multiple buttons and potentiometers as it reduces complexity for the product user.

Addressing Memory Limitations

Upon finishing the beginning of the EQ branch of the menu, memory limitations arose as the DSP would no longer boot with system memory above 83%. In order to test dynamic filter updating it was clear that the menu needed to be rewritten with only the most important functions and a single menu layer.

DSP Board

While the DSP board accomplishes all functionality required for this project, the provided documentation concerning the ADAU1702 development board from Analog Devices is lackluster at best for I2C transmission and active filter parameter editing. Jumper locations for I2C bus and multiple pinouts are mislabeled or difficult to find, despite the product being advertised as easily interfaceable with a microcontroller over I2C. For a prototype this board is a decent choice, but a marketable product would use a custom built PCB based around the ADAU1702 DSP that does not include the extraneous features that are built into the development board.

Software Design Process

Successes

In developing such a complex menu, the need for a pseudocode outline and graphical structure uses sufficient comments to navigate the large switch case state machine it entails. This layout of the menu drives much of the following code design process as it suggests important functions and helps plan the DSP interfacing methods. Hard coded state machine logic makes rotary encoder debugging straightforward and memory efficient. Push turn logic is both useful in reducing the complexity of the menu structure by reducing the levels of menu given the 7-segment LCDs limited display space and intuitive to use for parameter adjustments. Furthermore, on the fly filter calculation using coefficient lookup tables is an efficient option for limiting microcontroller resource requirements and helps restrict selecting parameters out of range for DSP functions.

Challenges

In getting a DSP board to boot, the most important milestone is achieving serial communication and it is a mistake to underestimate the complexity of sending large amounts of serial data. The manual was sparse on information regarding I2C pinouts and multiple hardware switch positions on the DSP EVAL board required counter intuitive (contradicting manual information at times) switch positions in order for the DSP to receive I2C data. Allot the most time for device interfacing as it has the most opportunity for failure and complexity.

Degrees of Success

Table 6.1 below addresses how well the engineering requirements are met for this project.

Table 6.1: Final Requirements Conclusions

Engineering Requirement	Met?	Why?
2. The <i>total harmonic distortion</i> at rated output power should be <0.1 %	At reduced power output (25 W per channel) yes ^[21]	This amplifier represents an acceptable cost to performance trade off and utilizes RCA inputs and screw terminal outputs.
3. Should be able to sustain an average <i>output power</i> > 50 watts to each main output assuming 4 ohm loads for each channel.	Yes, at 10% distortion ^[21]	The chosen amplifier is rated for this power output.
4. Should have an efficiency ($\eta > 80\%$)	Yes, >90% ^[21]	Class D topology satisfies this requirement
5. Average initial setup time should not exceed 30 minutes.	Yes, plugging in the system, connecting a source, and adjusting every parameter takes less than 15 minutes	The compressed menu structure makes navigation intuitive and requires minimal user effort to edit parameters. Also, many parameters omitted due to memory considerations.
6. The dimensions should not exceed 18.19" wide by 5.256" tall by 14" deep	All but width, 19.5"x3.5"x9"	The DSP eval board, amplifier boards, and microcontroller are less compressed than ideal to make them easier to configure, a custom system with a single PCD would require half the space and the power supply size could be reduced. Parts chosen based on availability and technical capability.
7. The dsp unit should have at least 6 parametric filters, custom High/Low pass filters with multiple orders with varying topologies,	6 parametric filters can be implemented using filter read in and can be used for custom high/low pass applications. Crossover topology is fixed to	Microcontroller memory constrains the number of filters implemented, crossover topology is beyond the knowledge of expected user and simplifying it to LR24 optimizes performance and user experience. Delay is limited to the point of being

and delay/phase for each output.	LR24 dB/octave, and delay/phase control is not implemented	<p>useless by the DSP board (not enough registers to buffer audio given a 96 kHz sample rate) and as such isn't implemented.</p> <p>Phase control beyond inversion is complex for users to understand and can be achieved by simply flipping the banana plugs on the output.</p>
8. Device should generate less than 35 dBa of sound under average load.	Yes	Only sound generating components are the amplifier fans and at 50w per channel class D they don't dissipate much heat and thus the fans generate minimal noise (inaudible from outside the case).
9. Production price must not exceed \$500	Yes	Borrowing the DSP board and the power supply (the two largest expenses of the project) significantly reduced the overall prototype cost overhead. Total of our purchased components is \$119.06.
10. Device must not draw more than 5 amps on a 120 V circuit under a worst case scenario.	Yes	The chosen power supply is rated for less than 2 A of current draw at 120V and as such even a worst case scenario such as each amplifier dissipation maximum power conservatively meets this requirement. Using class D amplifier boards allows maximum output power within the limits of the selected power supply.

References

[1] Earl R. Geddes. "Audio Acoustics in Small Rooms" Powerpoint Presentation hosted by GedLee LLC., Retrieved from
'<http://gedlee.azurewebsites.net/Papers/Audio%20Acoustics%206%2012%2005.ppt>'

Description of Source: The result of audio research Geddes conducted on the topics of acoustics and small rooms, this resource has extensive insight as to the limitations of small rooms and the impact that it has on speaker response. This resource is used as a justification for the need for DSP and displays the benefits through the form of response plots that DSP can have in terms of improving speaker performance. Earl Geddes has hundreds of times for his acoustic research into the way we perceive sound and has a PHD on studying the response of non-rectangular rooms. He is the one of the most experienced and respected acoustic engineers with 17 active patents and multiple pending patents.

[2] L.W. Lee and E.R. Geddes, "Auditory Perception of Nonlinear Distortion," Paper presented at the Audio Engineering Society 115th Convention - Paper 5891 (2005, Oct.)

Description of Source: This published paper on distortion explores the nature of our hearing and the subjective audible differences between different types and amounts of distortion. This research paper explains that human perception of distortion is frequency dependent and non-linear, meaning that 12% distortion could seem less distorted than 10%. It conducts an experiment on a group of 37 people to see how different amounts of THD (total harmonic distortion) and IMD (intermodulation distortion) affected our perceived sound quality. Furthermore, the paper proposes a new metric G_m that quantifies with great accuracy the perceived quality and is calculated from above mentioned metrics. This resource was chosen because it reinforces that there is much more to excellent audio than raw harmonic distortion measurements and that when judging sound quality one must accept that human ears are imperfect.

[3] Earl R. Geddes. "Small Room Acoustics in the Statistical Region," presented at the Audio Engineering Society 15th International Conference - Paper 15-006 (1998, Oct.)

Description of Source: This paper goes into depth about the measuring loudspeaker systems in rooms and the different regions dominated by room modes versus other stimuli. This paper was referenced because it discusses the nature of measuring sound in the statistical region (mid to high frequencies) which require more in depth measuring

methods. Being published at an AES international conference, this acoustics research carries considerable credibility.

[4] Texas Instruments. “*MSP430 Ultra-Low-Power Microcontrollers*,” ti.com/lit/. [Online]. Available:<http://www.ti.com/lit/sg/slabb034z/slabb034z.pdf> [Accessed: Oct. 13, 2014].

Description of Source: This product sheet goes into detail comparing all the variations of the MSP430 F series and is helpful in determining which microcontroller will provide the I/O necessary for the project. Texas Instruments is known for their excellent documentation, design environment, and example code which is one of the reasons that we are highly considering the FRAM series of MSP430. The FR series has capacitive touch capabilities which will prove useful for the LCD interface segment of the project.

[5] Texas Instruments, “*Interfacing an I2S Device to an MSP430 Device*,” ti.com/lit/. [Online]. Available:<http://www.ti.com/lit/an/slaa449a/slaa449a.pdf> [Accessed: Oct. 23, 2014].

Description of Source: This article details the process of converting an SPI interface for I2S operation, commonly used to drive DSP units. This may be necessary to interface the MSP430 with the chosen DSP unit, based on device flexibility (if it needs I2S, then this workaround will be important).

[6] miniDSP (2011, Oct. 7), “miniDSP 2x8 User Manual”. [Online]. Available: <http://www.minidsp.com/images/documents/miniDSP%202x8%20User%20manual%20v1.1.pdf>

Description of Source: This is the manual for the most likely DSP unit for the system. It contains details on microcontroller interfacing parameters and overall connectivity. This resource will be imperative in configuring the DSP microcontroller interface and learning how to communicate with it will determine the extent of features the final system implements.

[7] Occupational Safety and Health Organization (2013, Aug. 15). “Noise” in *OSHA Technical Manual*. ch. 5, sec. 3. [Online]. Available:https://www.osha.gov/dts/osta/otm/new_noise/index.html

Description of Source: The occupational safety and Health Administration is the US government sanctioned entity responsible for setting safe sound levels for the workplace

and within this project is used to establish safe listening levels for the playback of loudspeaker systems employing this or a comparable DSP amplifier.

[8] Parts Express (2011, Sept. 1). "250 Watt Subwoofer Plate Amplifier Model: SPA250". [Online]. Available:
<http://www.parts-express.com/pedocs/manuals/300-803-dayton-audio-spa250-manual.pdf>

Description of Source: Competitor mono subwoofer datasheet.

[9] Behringer. "iNUKE NU6000DSP/NU3000DSP/NU1000DSP User Manual". [Online]. Available:
http://www.behringer.com/assets/NU6000DSP_NU3000DSP_NU1000DSP_M_EN.pdf

Description of Source: Competing stereo DSP amplifier competitor datasheet.

[10] Peavey. "IPR2000/3000 DSP Operating Manual". [Online]. Available:
http://assets.peavey.com/literature/manuals/118438_26188.pdf

Description of Source: Competing stereo DSP amplifier operating manual.

[11] Cityfeet. "San Luis Obispo Retail Space" [Online]. Available:
<http://www.cityfeet.com/cont/ca/san-luis-obispo-retail-space#pgNum=3>

Description of Source: Used to approximate local office retail rent in San Luis Obispo.

[12] Payscale Human Capital. "Electrical Engineer Salary". [Online]. Available:
http://www.payscale.com/research/US/Job=Electrical_Engineer/Salary

Description of Source: Approximation of average electrical engineer annual salary.

[13] Consumer Electronics Association. "Industry Revenues to Reach Record High". [Online]. Available:
[http://www.ce.org/News/News-Releases/Press-Releases/2013-Press-Releases/CE-Industry-Revenues-to-Reach-Record-High-\\$209-Bil.aspx](http://www.ce.org/News/News-Releases/Press-Releases/2013-Press-Releases/CE-Industry-Revenues-to-Reach-Record-High-$209-Bil.aspx)

Description of Source: Consumer Electronics Market Association

[14] Westegg. "The Inflation Calculator". [Online]. Available:
<http://www.westegg.com/inflation/infl>

Description of Source: Inflation calculator for market size conversion.

[15] Consumer Electronics Association. "Home Audio Rebounds". [Online]. Available: <http://www.ce.org/i3/VisionArchiveList/VisionArchive/2011/October/Home-Audio-Rebounds.aspx>

Description of Source: Approximation of the present size of the home audio market.

[16] Dayton Audio. "2.1 Channel Class D Amplifier". [Online]. Available: <http://www.parts-express.com/dayton-audio-mca2250e-21-channel-class-d-plate-amplifier--300-771>

Description of Source: Competing product specification document.

[17] Herr, Norman. "Television and Health". [Online]. Available: <http://www.csun.edu/science/health/docs/tv&health.html>

Description of Source: Statistical information on project market consumer.

[18] US Census Bureau. "Computer and Internet Use in the United States". [Online]. Available: <http://www.census.gov/prod/2013pubs/p20-569.pdf>

Description of Source: Statistical information on project market consumer.

[19] Analog Devices. "SigmaDSPTM 28-/56-Bit Audio Processor Evaluation Board Operation Manual". [Online]. Available: http://www.analog.com/static/imported-files/eval_boards/EVAL-AD1940AZ.pdf

Description of Source: This is the manual for a likely DSP board for the system. It contains details on microcontroller interfacing parameters and overall connectivity. This resource will be imperative in configuring the DSP microcontroller interface and learning how to communicate with it will determine the extent of features the final system implements.

[20] Analog Devices Engineer Zone. "Real Time Control TUTORIAL: SAFE LOAD, Fixing and Breaking variables". [Online]. Available: <https://ez.analog.com/message/61175#61175>

[21] Parts-Express. "2x50W TDA7492 Class-D Amplifier Board". [Online]. Available: <http://www.parts-express.com/2x50w-tda7492-class-d-amplifier-board--320-301>

[22] Arduino. "Arduino Mega 2560". [Online]. Available: <http://www.arduino.cc/en/Main/ArduinoBoardMega2560>

Appendix A:Senior Project Analysis

Project Title: Audio DSP Amplifier

Student(s): Will Saba and Nick Barany

Advisor: Dr. Bridget Benson

1. Summary of Functional Requirements

a. Describe the overall capabilities or functions of your project or design.

Describe what your project does.

- i. The Audio DSP Amplifier is a wall-powered device that processes and splits an analog audio signal with the ability to power up to loudspeakers, including four main outputs (50 WPC) with the ability to drive a mono-summed subwoofer.
- ii. The system shall be programmable through external controls such as capacitive sliders or buttons and an LCD display as well as through USB for flexible setup and configuration.
- iii. The DSP section will have 6 independent parametric filters per channel on top of limiting, crossover, time delay, and phase control capabilities.

2. Primary Constraints

a. Describe significant challenges or difficulties associated with your project or implementation. Explain limiting factors or issues that influenced your chosen approach.

- i. Creating an effective menu system that is simple enough for basic users to operate but powerful enough to adjust independent filters, limiter, and crossovers for each channel.
- ii. It is difficult to find a DSP board (stand alone with standard RCA input and output terminals on a PCB) that has more than 4 output channels while still being reasonable (<\$400) in price.
- iii. All components must operate off a single 12 V DC power supply.
- iv. The system requires custom code to facilitate profile data storage, preset recall, remote read, and specially formatted file export to be transmitted over I²C [5].

3. Economic

a. What economic impacts result?

- i. **Human Capital:** The development of this device creates enough work to support jobs in engineering, manufacturing, and sales distribution.
- ii. **Financial Capital:** Profit may result from the development of the system and the price savings of the final product will be passed on to customers. It has potential to stimulate the production of competing products as other audio companies attempt to replicate its functionality.
- iii. **Natural Capital:** The product utilizes power amplifiers, a DSP board, a microcontroller, and a steel chassis whose components require electronic-waste recycling due to the presence of rare-earth metals.
- iv. **Costs:** The commercial sale price is largely dependent on how cheap the DSP section can be designed as the amplifier and microcontroller sections are only marginally cheaper in bulk than the development price. The direct profit from the device would come from product markup which would need to be approximately 45% to accommodate for development time and manufacturing scaling cost. Using an estimated production cost (including economy of scale) of \$380, the retail price would be at least \$600.

Table A.1: Audio DSP Amplifier Estimated Costs

Item	Number	Company	Cost (\$)
10A/24V Power Supply	1	Parts Express	115.00
7V Voltage Regulator	1	DROK	6.20
Microcontroller	1	Atmel	20.00
DSP Board (2 Input, 4 output)	1	Analog Devices [19]	600.00
2x50W/Channel Amplifier	2	Digikey or Parts Express	60.00
LCD Display	1	Hantronix Inc.	10.00
Chassis	1	Custom MDF	10.00

Mounting Hardware/Accessories	*	Multiple	30.00
Labor	300 hours		5400 (at \$18/hr)
Total			6251.20

Table A.1 above demonstrates the prospective total cost of parts and labor for the design of this project. The optimistic total (T_a) estimates the total cost to be \$6251.20. The most likely cost total (T_m) estimates the total to be \$6851.20. The pessimistic cost total (T_b) estimates the total cost to be \$7451.20. These projected estimations are calculated assuming labor costs of \$20/hr and \$22/hr for most likely cost total and pessimistic cost total, respectively. Below lists the Ford and Caulston [6] cost estimation formula.

$$\frac{(T_a + 4T_m + T_b)}{6}$$

Calculating the expected cost using the formula above, the result is shown below.

$$T = (6251.20 + (4 * 6851.20) + 7451.20)/6 = \$6851.20$$

4. If manufactured on a commercial basis:
 - a. 800 systems will be sold in the first year.
 - b. The total cost of the prototype can be deduced from Table A.1 above. Subtracting \$6000 for labor (20\$/hr) from the estimated total cost of \$6851.20 results in prototype total of \$851.20. Actual manufacturing product line systems will assemble for about **\$500**.
 - c. Estimated purchase price for each device: \$800
 - d. Estimated profit per year: $(800 * \$800) - (800 * \$500) = \$240,000$
 - e. Estimated cost for user to operate device, per unit time: Using an electricity cost of \$0.15/KWh, an average "normal use" current draw of approximately 2/3 amp at 120 V AC. This metric is calculated with the following expected average power figures and efficiency ratings:
 $15 * 4 W$ (Amplifier outputs) + $10 W$ (DSP) + $5 W$ (Microcontroller)
- 5) Environmental
 - a) The environmental impact of this project comes from the fabrication of the components. The microcontroller, DSP board, and the amplifiers (4 Channel and Mono Subwoofer) all use PCB fabrication which has an

indirect effect on environment through parts manufacturing. Factories are needed to produce the components, and those factories all use energy and caustic materials during production. As with all electronics, proper recycling in accordance with electronics disposal regulations is needed to reduce environmental impact. If not properly recycled, harmful chemicals may leak into the surroundings, directly affecting all local animal life and other people.

6) Manufacturability

- a) The manufacturing of the chassis will be the most difficult portion of the manufacturing process. The enclosure will be fabricated in shop using sheet metal. However, since there is significant power dissipation, heat dispersion will be a necessity to ensure the safety of components and strategic heatsinks and active fans may be necessary.

7) Sustainability

- a) In regards to product life, the components most likely to break first are the buttons and the rotary encoder due to constant physical stress from handling. For active electronic components, the sub amplifier will be run at higher power than the rest of the components in the device, resulting in a shorter life cycle than the other electronic components. These high stress components may require replacement to maintain system sustainability.
- b) Describe how this project impacts the sustainable use of resources:
 - i) By combining multiple products into a single housing, the result is less overall electronic waste and less housing material (sheet metal) being necessary for the same design goals.
- c) Describe any upgrades that would improve the design of the project.
 - i) One method of improving the design is to use a larger microcontroller to implement more functionality and more menu customization. This would create a more robust sound system that gives the user more control.
 - ii) The system could use a dedicated subwoofer amplifier to improve the output power to a subwoofer. This would increase the internal component size of the system, which could be difficult to fit for the original chassis to hold. The chassis may need to be increased in size to accommodate for this.

8) Ethical

- a) Positive ethical implications that this product provides for residential use would be for personal enjoyment of high quality audio (when listening to music, movies, etc). Users of this device will have their audio entertainment experience enhanced greatly. In accordance with the IEEE code of ethics, our system also guarantees that performance specifications will be met with absolute certainty through actual use testing (load testing) other than ideal circumstances.

9) Health & Safety

A health and safety concern associated with this project is its role in the ability to generate dangerous sound pressure levels of greater than 100 dBA (for exposure exceeding 15 minutes) in conjunction with speakers which provides physical harm to the ear in accordance to OSHA standards(See Figure A.1 below) [7].

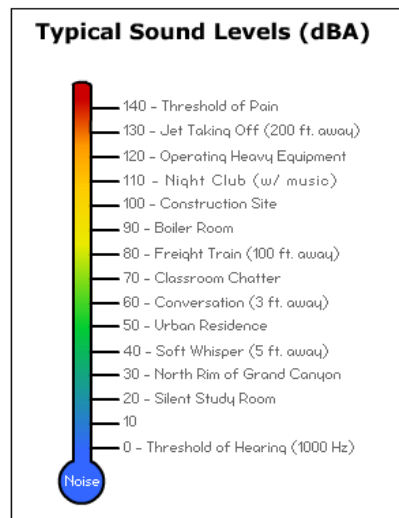


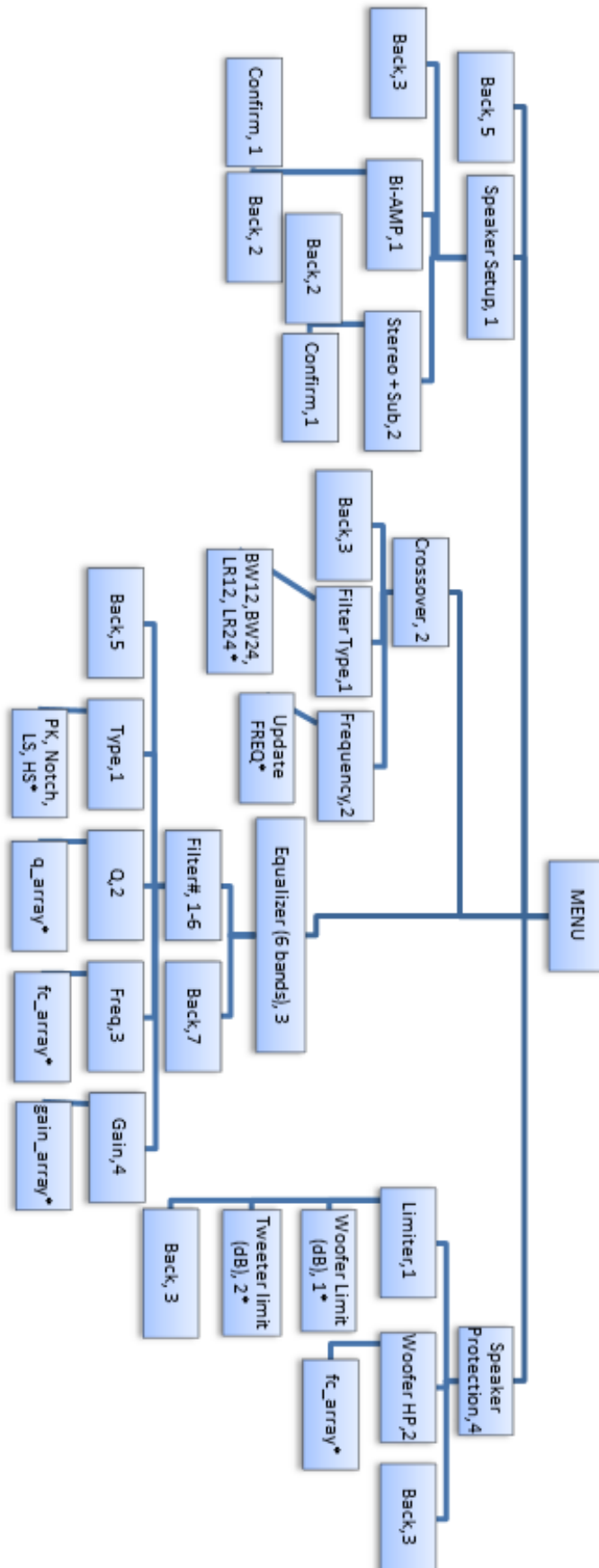
Figure A.1: OSHA Sound Level Safety Chart

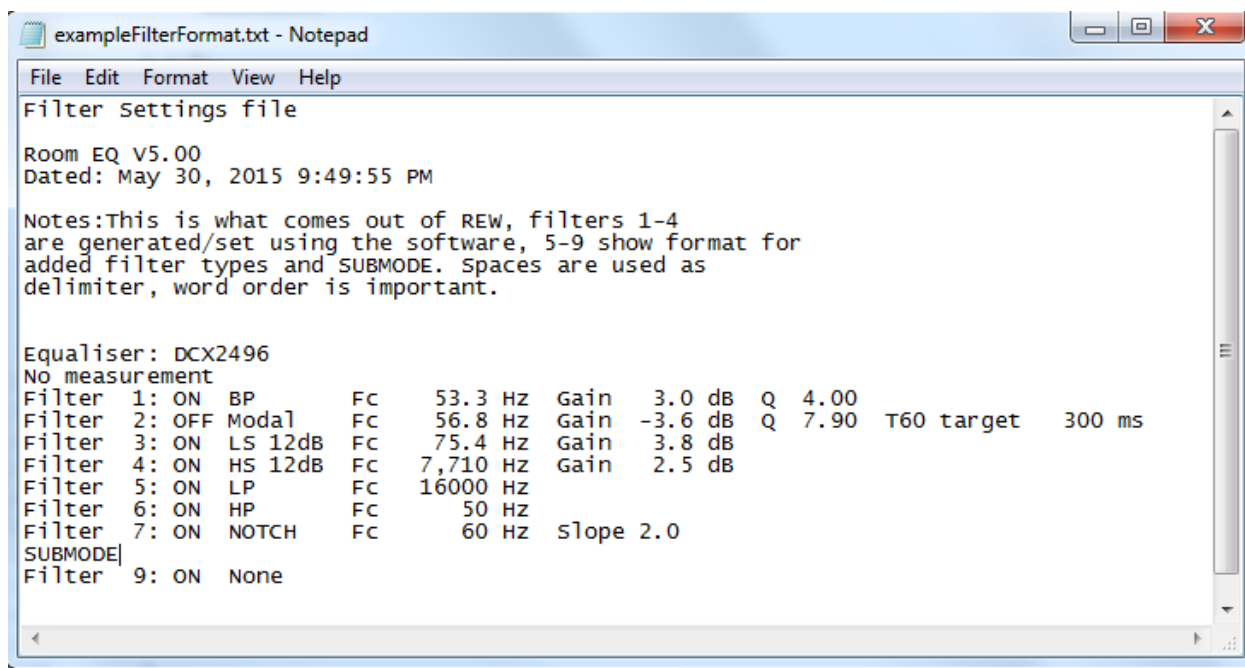
10) Social and Political

- a) A significant social impact with this devices is only prevalent when the Audio DSP amplifier is connected to speakers. The amplifier system could function as a replacement contractor amplifier with dsp capabilities if the power supply was replaced with one that runs off of a 70V power supply. This would function as an audio system for social gatherings at large meeting rooms, lecture halls, etc.

Appendix B: Software Design

Figure B.1: Original Menu Flow Diagram





```

exampleFilterFormat.txt - Notepad
File Edit Format View Help
Filter settings file

Room EQ V5.00
Dated: May 30, 2015 9:49:55 PM

Notes:This is what comes out of REW, filters 1-4
are generated/set using the software, 5-9 show format for
added filter types and SUBMODE. Spaces are used as
delimiter, word order is important.

Equaliser: DCX2496
No measurement
Filter 1: ON BP FC 53.3 HZ Gain 3.0 dB Q 4.00
Filter 2: OFF Modal FC 56.8 HZ Gain -3.6 dB Q 7.90 T60 target 300 ms
Filter 3: ON LS 12dB FC 75.4 HZ Gain 3.8 dB
Filter 4: ON HS 12dB FC 7,710 HZ Gain 2.5 dB
Filter 5: ON LP FC 16000 HZ
Filter 6: ON HP FC 50 HZ
Filter 7: ON NOTCH FC 60 HZ slope 2.0
SUBMODE|
Filter 9: ON None

```

Figure B.2: Example REW Filter File

*Note: Main - Menu, and highpass update altered/removed in final version (see Expo section).
See Filter Import (final version) for pieces of code that occur in both sketches (Arduino Projects).*

Microcontroller Code, Expo

Main - Maximum Menu Implemented

```

char foo; // junk code
// Work in progress, Senior Project Showcase Code - 5/29/15
// See "final" code for more comments on common functions
// Created by Will Saba and Nick Barany to interface an LCD 7-segment display and ADAU 1702 EVAL DSP board
// to an Arduino Uno
// Require selfboot off, s5 set left to "spi" in order to get data on scope i2c

#include <Wire.h>
#include <EEPROM.h>
#include "SigmaStudioFw.h"
#include "main_final_IC_1_REG.h"
#include <avr/io.h>
#include "Biquad.h"
#include "main_final_IC_1.h"
#include "main_final_IC_1_PARAM.h"

#include "LCDDriver.h"
#include "RotaryEncoder.h"

//Set sample rate for filter calculation
#define sampleRate 96000.0
#define I2C_ADDR_ADAU1702W 0x34 // i2c Write address of tested ADAU DSP board
#define type_peak 0
#define type_lowshelf 1
#define type_highshelf 2
#define type_notch 3
#define SUB_MODE_FLAG_HIGH 2
#define DEFAULT_CROSSOVER_FC 100.0
typedef PROGMEM const unsigned char ADI_REG_TYPE;

//Define crossover biquads
Biquad *LP1L = new Biquad();
Biquad *LP2L = new Biquad();
Biquad *LP1R = new Biquad();
Biquad *LP2R = new Biquad();
Biquad *HP1L = new Biquad();
Biquad *HP2L = new Biquad();
Biquad *HP1R = new Biquad();
Biquad *HP2R = new Biquad();

// This struct contains index variables for each filter parameter corresponding to its named lookup table
typedef struct Filter {
    int q = 10, gain = 10, fc = 33, type = type_peak;
    // int filterType = 0; // Initialize type to off
}Filter;

// declare 1 index storage variables out of 6 corresponding to Biquad filter1
Filter filter1i;//,filter2i,filter3i,filter4i,filter5i,filter6i;

//Define 6 blank biquads (BP gain of 0 dB), for now don't update previous settings from memory

Biquad *filter1 = new Biquad();
/* Biquad *filter2 = new Biquad();
Biquad *filter3 = new Biquad();
Biquad *filter4 = new Biquad();
Biquad *filter5 = new Biquad();

```

```

    Biquad *filter6 = new Biquad();
*/
int volume = 37; // default -> -3 dB DSP gain
// -40 -> 0 dB lookup table stored in decimal format for volume control and wherever -dB gain is needed as a decimal (dB values stored
as linear decimals in registers)
double dBLookup [41] =
{0.01,0.011220185,0.012589254,0.014125375,0.015848932,0.017782794,0.019952623,0.022387211,0.025118864,0.028183829,0.031622777,0.0354813
39,0.039810717,0.044668359,0.050118723,0.056234133,0.063095734,0.070794578,0.079432823,0.089125094,0.1,0.112201845,0.125892541,0.141253
754,0.158489319,0.177827941,0.199526231,0.223872114,0.251188643,0.281838293,0.316227766,0.354813389,0.398107171,0.446683592,0.501187234
,0.562341325,0.630957344,0.707945784,0.794328235,0.891250938,1};
// Stores common 1/3 octave increments of quality factor used for parametric filters. Doubles as lookup table for slope parameter in
high-shelf and low-shelf filters.
double qLookup [20] = {0.127,0.152, 0.182, 0.22, 0.267, 0.33, 0.4, 0.51, 0.67, 0.92, 1.04, 1.41, 1.9,2.14,2.87,4.32,5.76,7.2,8.65,10};

int is_pressed = 0;
int is_released = 0;
int push_turn = 0;
int turn_dir = 0;
int pos = 0;
int crossoverFc_i = 34; // crossover fc index
char buf[20];
char x= 0; // keeps track of hp index (first array dimension), defaults to 30 Hz HP in firmware, can be set in setup
RotaryEncoder encoder(A2, A3);
volatile int count = 0;
int menu[5] = {0,0,0,0,0};

//Function Declarations
void back();
void menu_update();
// Boot the LCD
int LCDscreen_init()
{
    //Wait for startup
    _delay_ms(50);
    LCD_4pin_init(12, 11, 10, 7, 6, 5, 4);

    //Functions set. 4 bit, 2 lines, display on
    LCD_cmd_function(0, 1, 1);

    //Display ON, Cursor off, Blink off
    LCD_cmd_display(1, 0, 0);

    //Display CLR
    LCD_cmd_clr();

    //Entry Mode (Shift off, increment)
    LCD_cmd_entry(1, 0);
    //Initialization END.
    return 0;
}

int lcd_write_str(char buf[]) //find length of string, send 1 char at a time for each index of str until it ends
{
    int i=0;
    int len;

    len = strlen(buf);
    for(i=0; i<len; i++)
    {
        LCD_send_data(buf[i]);
    }
}

```

```

        return 0;
    }
    // End boot the LCD//

    // Back Functions
    // Goes up 1 level in menu
    void back()
    {
        //go up a level
        is_pressed = 0;
        menu[menu[0]] = 0;
        menu[0]--;
    }
    // Menu Update
    void menu_update()
    {
        // Serial.print(F("turn_dir: "));
        // Serial.println(turn_dir);
        // Serial.print(F("is_pressed: "));
        // Serial.println(is_pressed);
        // Serial.print(F("is_released: "));
        // Serial.println(is_released);
        // Serial.println(push_turn);

        switch(menu[1])
        {
            case 0:
                if(turn_dir) //if knob is turned right or left no button press
                {
                    volume += turn_dir;
                    sprintf(buf, "Volume: %d dB", volume-40);
                    setVolume(dBLookup[volume], I2C_ADDR_ADAU1702W);
                    // Serial.println(dBLookup[volume]);

                }
                else if(is_pressed == 1)
                {
                    is_pressed = 0;
                    //one menu level deeper
                    menu[0]++;
                    menu[1]++;
                }
                else if(is_released)
                {
                    is_released = 0;
                    sprintf(buf, "Volume: %d dB", volume-40);

                }
                else // catchall scenario
                {
                    is_pressed = 0;
                    is_released = 0;
                    turn_dir = 0;
                }
            break; //break menu[1]

            case 1:
                switch(menu[2])
                {
                    case 0:
                        //crossoverh [1,1,0,0,0]

```

```

if(is_released == 1)
{
    sprintf(buf, "X-over Fc: %d", fcLookup[crossoverFc_i]);
    is_released = 0;
}
is_released = 0;

    if(turn_dir == 1 || turn_dir == -1)
    {
        if(push_turn == 1){
            //
            is_pressed = 0;
            crossoverFc_i += turn_dir;

            LP1L ->setBiquad(bq_type_lowpassLR, (double)fcLookup[crossoverFc_i]/ sampleRate, 1, 0);
            LP2L ->setBiquad(bq_type_lowpassLR, (double)fcLookup[crossoverFc_i] / sampleRate, 1, 0);
            LP1R ->setBiquad(bq_type_lowpassLR, (double)fcLookup[crossoverFc_i] / sampleRate, 1, 0);
            LP2R ->setBiquad(bq_type_lowpassLR, (double)fcLookup[crossoverFc_i] / sampleRate, 1, 0);
            HP1L ->setBiquad(bq_type_highpassLR, (double)fcLookup[crossoverFc_i] / sampleRate, 1, 0);
            HP2L ->setBiquad(bq_type_highpassLR, (double)fcLookup[crossoverFc_i] / sampleRate, 1, 0);
            HP1R ->setBiquad(bq_type_highpassLR, (double)fcLookup[crossoverFc_i] / sampleRate, 1, 0);
            HP2R ->setBiquad(bq_type_highpassLR, (double)fcLookup[crossoverFc_i] / sampleRate, 1, 0);
            setCrossover((double)fcLookup[crossoverFc_i]/ sampleRate, LP1L, LP1R,LP2L,LP2R,HP1L,HP1R,HP2L,HP2R);

            sprintf(buf, "X-over Fc: %d",fcLookup[crossoverFc_i]);

        }
        else// not push turn
        {
            menu[2] += turn_dir;
            if(turn_dir == 1){
                sprintf(buf, "Filter 1 Type: %d", filter1i.type);
            }
            else
            {
                sprintf(buf, "Back");
                menu[2] = 5; // wraparound
            }
        }
    }
}
else if(is_pressed == 1){
    push_turn = 1;
    is_pressed = 0;
}
else{ //is released
    // sprintf(buf, "X-over Fc: %d",fcLookup[crossoverFc_i]);
    is_pressed = 0;
    push_turn=0;
}
break;

case 1:

is_released = 0;

    if(turn_dir){
        if(push_turn == 1){
            is_pressed = 0;
            filter1i.type+=turn_dir;

```

```

    if(filter1i.type==4){
        filter1i.type=0;
    }
    else if (filter1i.type==1){//type = 3
        filter1i.type = 3;
    }
    if(filter1i.type == 0)
    {
        filter1->setType(bq_type_peak);
        sprintf(buf, "Filter 1: Pk"); // update once #defines proven
    }
    else if(filter1i.type == 1)
    {
        filter1->setType(bq_type_lowshelf);
        sprintf(buf, "Filter 1: LS");
    }
    else if(filter1i.type == 2)
    {
        filter1->setType(bq_type_highshelf);
        sprintf(buf, "Filter 1: HS");
    }
    else if(filter1i.type == 3)
    {
        filter1->setType(bq_type_notch);
        sprintf(buf, "Filter 1: NOTCH");
    }
    safeLoadFilter(filter1, MOD_2NDORDEREQFILTERBANK_ALGO_STAGE0_B0_ADDR);

} //end if push turn
else
{
    menu[2] += turn_dir;
    if(turn_dir == 1){
        sprintf(buf, "Filter 1 Q:%d", filter1i.q);
    }
    else
    {
        sprintf(buf, "X-over Fc: %d", fcLookup[crossoverFc_i]);
    }
    turn_dir = 0;
}
} //end if turn dir

else if(is_pressed == 1){
    push_turn = 1;
    is_pressed = 0;
}
else
{
    is_pressed = 0;
    push_turn=0;
    sprintf(buf, "Filter 1 Type: %d",filter1i.type);
}
break;
case 2:
    //Filter 1 Q

    is_released = 0;

```

```

if(turn_dir){
    if(push_turn == 1){
        is_pressed = 0;

        filter1i.q+=turn_dir;
        if(filter1i.q == 21)
        {
            filter1i.gain = 20;
        }
        else if(filter1i.q == -1){
            filter1i.gain = 0;
        }

        //Print the Q
        filter1->setQ(qLookup[filter1i.q]);
        sprintf(buf,"Filter 1 Q: %d", filter1i.q);
        safeLoadFilter(filter1, MOD_2NDORDEREQFILTERBANK_ALGO_STAGE0_B0_ADDR);
    }
    else{
        menu[2] += turn_dir; //Wrap around right side of menu
        if(turn_dir == 1){
            sprintf(buf, "Filter 1 Gain: %d", gainLookup[filter1i.gain]);
        }
        else{
            sprintf(buf, "Filter 1 Type: %d",filter1i.type);
        }
    }
    //end push turn if
    //end turn dir if

    else if(is_pressed == 1){
        push_turn = 1;
        is_pressed = 0;
    }

    else // is released
    {
        turn_dir = 0;
        is_pressed = 0;
        push_turn=0;
        sprintf(buf,"Filter 1 Q: %d", filter1i.q);
    }
}
break;
case 3:
//Filter 1 gain
is_released = 0;
if(turn_dir)
{
    if(push_turn == 1){
        is_pressed = 0;
        filter1i.gain+=turn_dir;
        if(filter1i.gain == 22)
        {
            filter1i.gain = 21;
        }
    }
    else if(filter1i.gain == -1){
        filter1i.gain = 0;
    }
    //Print the gain
    filter1->setPeakGain((double)gainLookup[filter1i.gain]);
    sprintf(buf,"Filter 1 Gain: %d", gainLookup[filter1i.gain]);
}

```

```

    safeLoadFilter(filter1, MOD_2NDORDEREQFILTERBANK_ALG0_STAGE0_B0_ADDR);
}
else
{
    menu[2] += turn_dir; //Wrap around right side of menu
    if(turn_dir == 1){
        sprintf(buf, "Filter Fc: %d", fcLookup[filter1i.fc]);
    }
    else{
        sprintf(buf, "Filter 1 Q: %d",filter1i.q);
    }
} //end push turn if
} //end turn dir if
else if(is_pressed == 1){
    push_turn = 1;
    is_pressed = 0;
}

else{ // is released
    turn_dir = 0;
    is_pressed = 0;
    push_turn=0;
    sprintf(buf,"Filter 1 Gain: %d", gainLookup[filter1i.gain]);
}
break;

case 4:

//Filter 1 freq
is_released = 0;
if(turn_dir){
    if(push_turn == 1){
        is_pressed = 0;
        filter1i.fc+=turn_dir;
        //Protect range of Fc lookup table from overflow
        if(filter1i.fc == 174)
        {
            filter1i.fc = 173;
        }
        else if(filter1i.fc == -1){
            filter1i.fc = 0;
        }
        //Print the gain
        filter1->setFc((double)fcLookup[filter1i.fc]/sampleRate);
        sprintf(buf,"Filter Fc: %d", fcLookup[filter1i.fc]);
        safeLoadFilter(filter1, MOD_2NDORDEREQFILTERBANK_ALG0_STAGE0_B0_ADDR);
    }
    else
    {
        menu[2] += turn_dir; //Wrap around right side of menu
        if(turn_dir == 1){
            sprintf(buf, "Back");
        }
        else{
            sprintf(buf, "Filter 1 Gain: %d",gainLookup[filter1i.gain]);
        }
    }
} //end push turn if
} //end turn dir if

else if(is_pressed == 1){
    push_turn = 1;

```



```

        is_pressed = 0;
    }
    else // is released
    {
        turn_dir = 0;
        is_pressed = 0;
        push_turn=0;
        sprintf(buf, "Filter Fc: %d", fcLookup[filter1i.fc]);
    }
    break;
    case 5:
        //Back level
        if(turn_dir == 1)
        {
            menu[2] = 1; // Wrap around to Speaker Setup
            sprintf(buf, "X-over Fc: %d", fcLookup[crossoverFc_i]);
        }
        else if(turn_dir == -1)
        {
            menu[2] += turn_dir;
            sprintf(buf, "Filter Fc: %d", fcLookup[filter1i.fc]);
        }
        else if(is_pressed == 1){
            back();
        }
        break;
    }
    break;
}
// End of Menu
LCD_cmd_clr();
delay(100);
lcd_write_str(buf);
}

// End of menu_update

void setup() {
// Serial.begin(9600);

//Wait for DSP to initialize clock
delay(350);

//Write "firmware" version of code to DSP

default_download_IC_1();
delay(100);
LCDscreen_init(); // init LCD
delay(100);
// Initialize volume to -3 dB
setVolume(dBLookup[volume], I2C_ADDR_ADAU1702W);
// Setup for rotary encoder handling
pinMode(2, INPUT_PULLUP);
sei();
PCICR |= (1 << PCIE1); // This enables Pin Change Interrupt 1 that covers the Analog input pins or Port C.
PCICR |= (1 << PCIE2); //This enables Pin Change Interrupt 2 that covers Port D.
PCIFR |= (1 << PCIF2); //Enables a flag in response to logic change to trigger ISR2. Flag cleared after completion of ISR.
PCMSK1 |= (1 << PCINT10) | (1 << PCINT11); // This enables the interrupt for pin 2 and 3 of Port C.
PCMSK2 |= (1 << PCINT18); // This enables the interrupt for pin 2 of Port D.

```

```

/*
//Example function calls
updateHP(hpLookup_loc, 0, DEVICE_ADDR_IC_1, MOD_HIPASSPROTECT_ALGO_STAGE0_B0_ADDR);
setLimiter(dBLookup[limiterIndex-15], DEVICE_ADDR_IC_1,
MOD_LIMITERRIGHT_LOW_ALGO_THRESHOLD_ADDR,MOD_LIMITERLEFT_LOW_ALGO_THRESHOLD_ADDR);
setVolume(dBLookup[0], DEVICE_ADDR_IC_1, MOD_SWVOL1_ALGO_TARGET_ADDR,MOD_SWVOL1_ALGO_STEP_ADDR);
filter1 ->setBiquad(bq_type_peak, (double)pgm_read_word_near(fcLookup+filter1i.fc-65) / sampleRate, qLookup[filter1i.q],
gainLookup[filter1i.gain+80]);
*/
//Initialize crossover to 100 Hz LR 24 dB/octave//

LP1L ->setBiquad(bq_type_lowpassLR, DEFAULT_CROSSOVER_FC / sampleRate, 1, 0);
LP2L ->setBiquad(bq_type_lowpassLR, DEFAULT_CROSSOVER_FC / sampleRate, 1, 0);
LP1R ->setBiquad(bq_type_lowpassLR, DEFAULT_CROSSOVER_FC / sampleRate, 1, 0);
LP2R ->setBiquad(bq_type_lowpassLR, DEFAULT_CROSSOVER_FC / sampleRate, 1, 0);
HP1L ->setBiquad(bq_type_highpassLR, DEFAULT_CROSSOVER_FC / sampleRate, 1, 0);
HP2L ->setBiquad(bq_type_highpassLR, DEFAULT_CROSSOVER_FC / sampleRate, 1, 0);
HP1R ->setBiquad(bq_type_highpassLR, DEFAULT_CROSSOVER_FC / sampleRate, 1, 0);
HP2R ->setBiquad(bq_type_highpassLR, DEFAULT_CROSSOVER_FC / sampleRate, 1, 0);
setCrossover(DEFAULT_CROSSOVER_FC/sampleRate, LP1L, LP1R,LP2L,LP2R,HP1L,HP1R,HP2L,HP2R);
}

// ISRs for handling push, and rotate for rotary encoder
ISR(PCINT1_vect) {
  noInterrupts();
  encoder.tick(); // just call tick() to check the state.
  interrupts();
}

//Debouncing issue, ISR keeps getting called on logic change
ISR(PCINT2_vect) {
  noInterrupts();
  delay(50);
  if(digitalRead(2) == 0) //active low button aka press down
  {
    is_pressed = 1;
  }
  else // button release
  {
    is_released = 1;
  }
  interrupts();
}

void loop() {

  int newPos = encoder.getPosition();
  if (newPos != pos) {
    //Serial.print(newPos);
    //Serial.println();
    turn_dir = newPos - pos;
    pos = newPos;
  }

  //Variable updates depending on whether turn right/left or button
  if(turn_dir || is_pressed || is_released) //if any menu change interrupt occurs
  {
    menu_update(); //change menu
    turn_dir = 0; //reset position vector
  }
}

```

```
}
}
```

Update High-Pass Protect

// This code was removed in the final version (from SigmaStudioFw.h) because highpass implemented using filter read-in method and as such a separate function was redundant and the memory it took up to store all the coefficients for 14 high-pass values // was repurposed to implement other functionality.

```
int gainLookup [21] = {-10,-9,-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10};
```

```
#define HP_LOOKUP_SIZE 280
```

```
int limiterIndex = 41; // used to initiate limiters to -0 dB via dBLookup for a full scale output range.
```

```
// 20Hz (index 0) -- 80 Hz (index 12) $96kHz BW HP 2nd order in 5.23 format (DSP) index 13 = 500 Hz -- testing
/* Define Hi-pass protect values, calculated using SigmaStudio table calculator, converted to comma delimited array.*/
```

```
ADI_REG_TYPE hpLookup[HP_LOOKUP_SIZE] = {
0x00,0x7F,0xE1,0xAF,0xFF,0x00,0x3C,0xA2,0x00,0x7F,0xE1,0xAF,0x00,0xFF,0xC3,0x57,0xFF,0x80,0x3C,0x9B,
0x00,0x7F,0xDA,0x1C,0xFF,0x00,0x4B,0xC8,0x00,0x7F,0xDA,0x1C,0x00,0xFF,0xB4,0x2D,0xFF,0x80,0x4B,0xBD,
0x00,0x7F,0xD2,0x89,0xFF,0x00,0x5A,0xED,0x00,0x7F,0xD2,0x89,0x00,0xFF,0xA5,0x03,0xFF,0x80,0x5A,0xDD,
0x00,0x7F,0xCA,0xF7,0xFF,0x00,0x6A,0x12,0x00,0x7F,0xCA,0xF7,0x00,0xFF,0x95,0xD8,0xFF,0x80,0x69,0xFC,
0x00,0x7F,0xC3,0x65,0xFF,0x00,0x79,0x35,0x00,0x7F,0xC3,0x65,0x00,0xFF,0x86,0xAE,0xFF,0x80,0x79,0x19,
0x00,0x7F,0xBB,0xD4,0xFF,0x00,0x88,0x58,0x00,0x7F,0xBB,0xD4,0x00,0xFF,0x77,0x84,0xFF,0x80,0x88,0x34,
0x00,0x7F,0xB4,0x43,0xFF,0x00,0x97,0x7A,0x00,0x7F,0xB4,0x43,0x00,0xFF,0x68,0x5A,0xFF,0x80,0x97,0x4D,
0x00,0x7F,0xAC,0xB3,0xFF,0x00,0xA6,0x9A,0x00,0x7F,0xAC,0xB3,0x00,0xFF,0x59,0x2F,0xFF,0x80,0xA6,0x64,
0x00,0x7F,0xA5,0x23,0xFF,0x00,0xB5,0xBA,0x00,0x7F,0xA5,0x23,0x00,0xFF,0x4A,0x05,0xFF,0x80,0xB5,0x7A,
0x00,0x7F,0x9D,0x93,0xFF,0x00,0xC4,0xD9,0x00,0x7F,0x9D,0x93,0x00,0xFF,0x3A,0xDB,0xFF,0x80,0xC4,0x8E,
0x00,0x7F,0x96,0x04,0xFF,0x00,0xD3,0xF8,0x00,0x7F,0x96,0x04,0x00,0xFF,0x2B,0xB1,0xFF,0x80,0xD3,0xA0,
0x00,0x7F,0x8E,0x76,0xFF,0x00,0xE3,0x15,0x00,0x7F,0x8E,0x76,0x00,0xFF,0x1C,0x86,0xFF,0x80,0xE2,0xB0,
0x00,0x7F,0x86,0xE7,0xFF,0x00,0xF2,0x31,0x00,0x7F,0x86,0xE7,0x00,0xFF,0x0D,0x5C,0xFF,0x80,0xF1,0xBF,
0x00,0x7D,0x12,0x74,0xFF,0x05,0xDB,0x18,0x00,0x7D,0x12,0x74,0x00,0xFA,0x13,0xC2,0xFF,0x85,0xC9,0xF2,
};
```

```
void safeWriteHP(int devAddress, int address, int length_reg, ADI_REG_TYPE* pData){
```

```
int regcount=0, addcount=0, addtemp=0;
```

```
for(regcount=0;regcount<length_reg;regcount++)
```

```
{
```

```
Wire.beginTransmission(devAddress); // 0x34 (write port)
```

```
//send data register 1 byte at a time
```

```
Wire.write((((SAFELOAD_DATA1+regcount) >> 8) & 0xFF)); // send upper byte safeload data register
```

```
Wire.write(((SAFELOAD_DATA1+regcount) & 0xFF)); // send lower byte
```

```
Wire.write(0x00); //load 00 into byte 3 as required by register write format
```

```
// pData references the starting address of the target array, addcount keeps track of "register offset, ie 4 bytes"
```

```
Wire.write(pgm_read_byte(pData+addcount));
```

```
Wire.write(pgm_read_byte(pData+addcount+1));
```

```
Wire.write(pgm_read_byte(pData+addcount+2));
```

```
Wire.write(pgm_read_byte(pData+addcount+3));
```

```
Wire.endTransmission();
```

```
Wire.beginTransmission(devAddress); // 0x34 (write port)
```

```
//Send address register 1 byte at a time
```

```
Wire.write((((SAFELOAD_ADDR1+regcount) >> 8) & 0xFF)); // send upper byte (0x08), safeload address register
```

```
Wire.write(((SAFELOAD_ADDR1+regcount) & 0xFF)); // send lower byte
```

```
Wire.write(((address+regcount) >> 8) & 0xFF)); // destination register
```

```
Wire.write((address+regcount) & 0xFF);
```

```
Wire.endTransmission();
```

```
addcount+=4;
```

```
}
```

```
}
```

```

void updateHP(ADI_REG_TYPE* pData, char turnDir, short deviceaddress, short address){

// 20 bytes per HP frequency -> 20 byte offset array allowing HP frequency selection by index offset. Necessary because progmem can't
handle 3D arrays
int hpLookupOffset[14] = {0,20,40,60,80,100,120,140,160,180,200,220,240,260};
int i=0, x=0;
unsigned char hp_temp[20];
ADI_REG_TYPE* hp_temp_i=0;
//incorporate rotary encoder turn for adjusting the frequency
x+=turnDir;
hp_temp_i = pData+hpLookupOffset[x];
// Call safeWriteHP passing the pointer hp_temp_i storing the location of the beginning of the chosen HP value
safeWriteHP(deviceaddress, address, 5, hp_temp_i);
set_core_IST_bit(I2C_ADDR_ADAU1702W);
}

```

Microcontroller Code, Filter Import

Main

```

char junk; // junk code, included to avoid arduino bug
/*
 * File:           C:\Users\wsaba\Google Drive\Senior Project\uC and DSP Code\EXPERIMENT\experiment.ino
 *
 * Created:        Monday, June 01, 2015 9:43:02 AM
 * Description:    Main, Filter Import
 *
 * This software is distributed in the hope that it will be useful,
 * but is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
 * CONDITIONS OF ANY KIND, without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
 *
 *
 * Purpose: This file controls the menu, LCD driver, and filter creation
 *
 */

/*****-#includes - Libraries-*****/
// -----i2c communication-----//
#include <Wire.h>
// -----EEPROM Preset Reading-----//
#include <EEPROM.h>
//-----DSP Interaction Functions-----//
#include "SigmaStudioFw.h"
//----- #defines for DSP Control Register Parameters-----//
#include "main_final_IC_1_REG.h"
//-----STDIO Equivalent for Arduino-----//
#include <avr/io.h>
//-----Biquad Calculation, Biquad Class-----//
#include "Biquad.h"
//-----DSP Boot Sequence and Register Storage-----//
#include "main_final_IC_1.h"
//-----#defines for DSP Parameter Registers (Name Based References)-----//
#include "main_final_IC_1_PARAM.h"
//-----Necessary for using the 7-segment display----//
#include "LCDDriver.h"
//-----Rotary Encoder Handling-----//
#include "RotaryEncoder.h"
/*****End ->#includes - Libraries-*****/

```

```

//Set samplerate for filter calculation
#define sampleRate 96000.0
// i2c Write address of the ADAU DSP board
#define I2C_ADDR_ADAU1702W 0x34
//#defines corresponding to enumerated filter types for menu-display and code readability//
#define type_peak 0
#define type_lowshelf 1
#define type_highshelf 2
#define type_notch 3
//Return value indicating that the Import File designates subwoofer mode//
#define SUB_MODE_FLAG_HIGH 2
// Default crossover Fc, in Hz
#define DEFAULT_CROSSOVER_FC 60
//DSP registers stored in progmem for reduced dynamic memory access, read out via default download -- boot sequence
//typedef used to visually indicate these variables and to match default_download type format//
typedef PROGMEM const unsigned char ADI_REG_TYPE;

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

//8 biquads necessary to implement 4th order crossover, LP = low pass, HP = high pass, 1 = 1st filter,
// 2 = 2nd filter (order in cascade), R = right channel, L = left channel
Biquad *LP1L = new Biquad();
Biquad *LP2L = new Biquad();
Biquad *LP1R = new Biquad();
Biquad *LP2R = new Biquad();
Biquad *HP1L = new Biquad();
Biquad *HP2L = new Biquad();
Biquad *HP1R = new Biquad();
Biquad *HP2R = new Biquad();

//Default Volume, initialized in Setup(), this is the index, points to dBLookup
int volume = 37;
// -40 -> 0 dB lookup table stored in decimal format for volume control and wherever -dB gain is needed as a decimal (dB values stored
as linear decimals in registers)
double dBLookup [41] =
{0.01,0.011220185,0.012589254,0.014125375,0.015848932,0.017782794,0.019952623,0.022387211,0.025118864,0.028183829,0.031622777,0.0354813
39,0.039810717,0.044668359,0.050118723,0.056234133,0.063095734,0.070794578,0.079432823,0.089125094,0.1,0.112201845,0.125892541,0.141253
754,0.158489319,0.177827941,0.199526231,0.223872114,0.251188643,0.281838293,0.316227766,0.354813389,0.398107171,0.446683592,0.501187234
,0.562341325,0.630957344,0.707945784,0.794328235,0.891250938,1};

// Globals that handle menu operations
int is_pressed = 0;
int is_released = 0;
int push_turn = 0;
int turn_dir = 0;
int pos = 0;
// Index responsible for handling menu crossover adjustment, points to fcLookup
int crossoverFc_i = 34; //

// Display buffer
char buf[20];
//Initialize Analog inputs 2 and 3 as encoder L and R
RotaryEncoder encoder(A2, A3);
volatile int count = 0;
//Declare menu variable
int menu[5] = {0,0,0,0,0};

/*****--END ->Global Variables--*****/

/*****--Function Declarations--*****/

```

```

void menu_update();

// Boot the LCD
int LCDscreen_init()
{
    //Wait for startup
    _delay_ms(50);
    LCD_4pin_init(12, 11, 10, 7, 6, 5, 4);

    //Functions set. 4 bit, 2 lines, display on
    LCD_cmd_function(0, 1, 1);

    //Display ON, Cursor off, Blink off
    LCD_cmd_display(1, 0, 0);

    //Display CLR
    LCD_cmd_clr();

    //Entry Mode (Shift off, increment)
    LCD_cmd_entry(1, 0);
    //Initialization END.
    return 0;
}

int lcd_write_str(char buf[]) //find length of string, send 1 char at a time for each index of str until it ends
{
    int i=0;
    int len;

    len = strlen(buf);
    for(i=0; i<len; i++)
    {
        LCD_send_data(buf[i]);
    }
    return 0;
}
// End boot the LCD//

// Main Menu, called on rotary encoder interrupts: is_pressed, is_released, is_turned in - loop()
void menu_update()
{
    switch(menu[1])
    {
        case 0:
            if(turn_dir) //if rotary encoder is turned right or left
            {
                volume += turn_dir;
                //Prevent out of range (>0 dB or <-40 dB)
                if (volume >40){
                    volume = 40;
                }
                else if (volume < 0){
                    volume = 0;
                }
                sprintf(buf, "Volume: %d dB", volume-40);
                setVolume(dBLookup[volume], I2C_ADDR_ADAU1702W);
            }
            else if(is_pressed == 1)
            {
                is_pressed = 0;
                //one menu level deeper
            }
    }
}

```

```

    menu[0]++;
    menu[1]++;
}
else if(is_released)
{
    is_released = 0;
    sprintf(buf, "Volume: %d dB", volume-40); // convert dB index position to dB, offset since range is -40->0 dB
}
else // catchall scenario
{
    is_pressed = 0;
    is_released = 0;
    turn_dir = 0;
}
break; //break menu[1]

case 1:
switch(menu[2])
{
    case 0:
        //crossoverh [1,1,0,0,0]
        if(is_released == 1)
        {
            sprintf(buf, "X-over Fc: %d", fcLookup[crossoverFc_i]);
            is_released = 0;
        }
        if(turn_dir == 1 || turn_dir == -1){
            if(push_turn == 1){
                //
                is_pressed = 0;
                crossoverFc_i +=turn_dir;
                //Update 8 crossover biquads
                LP1L ->setBiquad(bq_type_lowpassLR, (double)fcLookup[crossoverFc_i]/ sampleRate, 1, 0);
                LP2L ->setBiquad(bq_type_lowpassLR, (double)fcLookup[crossoverFc_i] / sampleRate, 1, 0);
                LP1R ->setBiquad(bq_type_lowpassLR, (double)fcLookup[crossoverFc_i] / sampleRate, 1, 0);
                LP2R ->setBiquad(bq_type_lowpassLR, (double)fcLookup[crossoverFc_i] / sampleRate, 1, 0);
                HP1L ->setBiquad(bq_type_highpassLR, (double)fcLookup[crossoverFc_i] / sampleRate, 1, 0);
                HP2L ->setBiquad(bq_type_highpassLR, (double)fcLookup[crossoverFc_i] / sampleRate, 1, 0);
                HP1R ->setBiquad(bq_type_highpassLR, (double)fcLookup[crossoverFc_i] / sampleRate, 1, 0);
                HP2R ->setBiquad(bq_type_highpassLR, (double)fcLookup[crossoverFc_i] / sampleRate, 1, 0);
                //Load 8 updated biquads into DSP
                setCrossover((double)fcLookup[crossoverFc_i]/ sampleRate, LP1L, LP1R,LP2L,LP2R,HP1L,HP1R,HP2L,HP2R);

                sprintf(buf, "X-over Fc: %d", fcLookup[crossoverFc_i]);
            }
            else// not push turn, go to back
            {
                menu[2] = 1;
                sprintf(buf, "Back");
            }
        }
        else if(is_pressed == 1){
            push_turn = 1;
            is_pressed = 0;
        }
        else{ //is released
            // sprintf(buf, "X-over Fc: %d",fcLookup[crossoverFc_i]);
            is_pressed = 0;
            is_released = 0;
            push_turn=0;
        }
    }
}

```

```

        }
        break;
    case 1:
        if(turn_dir)
        {
            sprintf(buf, "X-over Fc: %d", fcLookup[crossoverFc_i]);
            menu[2] = 0;
        }
        else if(is_pressed == 1)
        {
            //->back() function removed, hard coded to avoid bugs
            menu[1] = 0;
            menu[2] = 0;
            menu[0] = 1;
            is_pressed = 0;
        }
        is_released = 0;
        break;
    }
    break;
}
//Write the buffered string to the display
LCD_cmd_clr();
delay(100);
lcd_write_str(buf);
//Serial.print(F("Menu: "));
//Serial.print(menu[0]);
// Serial.print(menu[1]);
// Serial.print(menu[2]);
// Serial.print(menu[3]);
// Serial.println(menu[4]);
//Serial.println(buf);
//Serial.println("");
//End temp menu disable
}

// End of menu_update

void setup() {
    Biquad *filter1 = new Biquad();
    Biquad *filter2 = new Biquad();
    Biquad *filter3 = new Biquad();
    Biquad *filter4 = new Biquad();
    Biquad *filter5 = new Biquad();
    Biquad *filter6 = new Biquad();
    /*
    Biquad *filter7 = new Biquad();
    Biquad *filter8 = new Biquad();
    Biquad *filter9 = new Biquad();
    Biquad *filter10 = new Biquad();
    */
    int subMode = 0; // temporary, testing submode

    //Wait for DSP to initialize clock
    delay(350);
    //Write "firmware" version of code to DSP
    default_download_IC_1();
    //Wait for DSP to load memory
    delay(100);

```



```

// Initialize LCD
LCDscreen_init();
//Diagnostic display delay
delay(100);
//Pullup on rotary press enable
pinMode(2, INPUT_PULLUP);
sei();
PCICR |= (1 << PCIE1); // This enables Pin Change Interrupt 1 that covers the Analog input pins or Port C.
PCICR |= (1 << PCIE2); //This enables Pin Change Interrupt 2 that covers Port D.
PCIFR |= (1 << PCIF2); //Enables a flag in response to logic change to trigger ISR2. Flag cleared after completion of ISR.
PCMSK1 |= (1 << PCINT10) | (1 << PCINT11); // This enables the interrupt for pin 2 and 3 of Port C.
PCMSK2 |= (1 << PCINT18); // This enables the interrupt for pin 2 of Port D.

/*****--Load Default DSP filters (different from firmware version)--*****/

setVolume(dBLookup[volume], I2C_ADDR_ADAU1702W);
//Initialize crossover to DEFAULT_CROSSOVER_FC Hz LR 24 dB/octave//

LP1L ->setBiquad(bq_type_lowpassLR, DEFAULT_CROSSOVER_FC / sampleRate, 1, 0);
LP2L ->setBiquad(bq_type_lowpassLR, DEFAULT_CROSSOVER_FC / sampleRate, 1, 0);
LP1R ->setBiquad(bq_type_lowpassLR, DEFAULT_CROSSOVER_FC / sampleRate, 1, 0);
LP2R ->setBiquad(bq_type_lowpassLR, DEFAULT_CROSSOVER_FC / sampleRate, 1, 0);
HP1L ->setBiquad(bq_type_highpassLR, DEFAULT_CROSSOVER_FC / sampleRate, 1, 0);
HP2L ->setBiquad(bq_type_highpassLR, DEFAULT_CROSSOVER_FC / sampleRate, 1, 0);
HP1R ->setBiquad(bq_type_highpassLR, DEFAULT_CROSSOVER_FC / sampleRate, 1, 0);
HP2R ->setBiquad(bq_type_highpassLR, DEFAULT_CROSSOVER_FC / sampleRate, 1, 0);

setCrossover(DEFAULT_CROSSOVER_FC/ sampleRate, LP1L, LP1R,LP2L,LP2R,HP1L,HP1R,HP2L,HP2R);
// load text file function//

// Import filters from Room Equalizer Wizard export format text pattern, Text is currently stored in EEPROM for testing purposes
starting at address 0
subMode = importFilter(filter1,filter2,filter3,filter4,filter5,filter6);//,filter7,filter8);//,filter9,filter10); // 7-10 Disabled
for memory test

// note that filter1 starts at "stage0" so filter names are stage+1
safeLoadFilter(filter1, MOD_2NDORDEREQFILTERBANK_ALG0_STAGE0_B0_ADDR);
safeLoadFilter(filter2, MOD_2NDORDEREQFILTERBANK_ALG0_STAGE1_B0_ADDR);
safeLoadFilter(filter3, MOD_2NDORDEREQFILTERBANK_ALG0_STAGE2_B0_ADDR);
safeLoadFilter(filter4, MOD_2NDORDEREQFILTERBANK_ALG0_STAGE3_B0_ADDR);
safeLoadFilter(filter5, MOD_2NDORDEREQFILTERBANK_ALG0_STAGE4_B0_ADDR);
safeLoadFilter(filter6, MOD_2NDORDEREQFILTERBANK_ALG0_STAGE5_B0_ADDR);
// Maximum number of filters to implement given current firmware = 10, 7-10 disabled due to memory constraints
/* safeLoadFilter(filter7, MOD_2NDORDEREQFILTERBANK_ALG0_STAGE6_B0_ADDR);
safeLoadFilter(filter8, MOD_2NDORDEREQFILTERBANK_ALG0_STAGE7_B0_ADDR);
safeLoadFilter(filter9, MOD_2NDORDEREQFILTERBANK_ALG0_STAGE8_B0_ADDR);
safeLoadFilter(filter10, MOD_2NDORDEREQFILTERBANK_ALG0_STAGE9_B0_ADDR);
*/
//Convert Left and Right (low) outputs to a singular left low mono subwoofer output by ...
// Toggling 2 mutes and adjusting mux gain to compensate for the 2x gain of the summed signals
if(subMode == SUB_MODE_FLAG_HIGH){

subModeEnable(MOD_SUBMODETOGGLE_MUTENOSLEWALG2MUTE_ADDR,MOD_RIGHTMUTEIFSUB_MUTENOSLEWALG3MUTE_ADDR,MOD_SUBMODEGAINADJ_SINGLECTRLMIXERNE
W19401_ADDR);
}

/*****-- Example Function Calls for Updating DSP Parameters--*****/
/*
//Example function calls
updateHP(hpLookup_loc, 0, DEVICE_ADDR_IC_1, MOD_HIPASSPROTECT_ALG0_STAGE0_B0_ADDR);

```

```

    setLimiter(dBLookup[limiterIndex-15], DEVICE_ADDR_IC_1,
MOD_LIMITERRIGHT_LOW_ALGO_THRESHOLD_ADDR,MOD_LIMITERLEFT_LOW_ALGO_THRESHOLD_ADDR);
    setVolume(dBLookup[0], DEVICE_ADDR_IC_1, MOD_SWVOL1_ALGO_TARGET_ADDR,MOD_SWVOL1_ALGO_STEP_ADDR);
    filter1 ->setBiquad(bq_type_peak, (double)pgm_read_word_near(fcLookup+filter1i.fc-65) / sampleRate, qLookup[filter1i.q],
gainLookup[filter1i.gain+80]);
*/
}

// ISRs for handling push, and rotate for rotary encoder
ISR(PCINT1_vect) {
    noInterrupts();
    encoder.tick(); // just call tick() to check the state.
    interrupts();
}

//Debouncing issue, ISR keeps getting called on logic change
ISR(PCINT2_vect) {
    noInterrupts();
    delay(50);
    if(digitalRead(2) == 0) //active low button aka press down
    {
        is_pressed = 1;
    }
    else // button release
    {
        is_released = 1;
    }
    interrupts();
}

void loop() {

    int newPos = encoder.getPosition();
    if (newPos != pos) {
        //Serial.print(newPos);
        //Serial.println();
        turn_dir = newPos - pos;
        pos = newPos;
    }

    //Variable updates depending on whether turn right/left or button
    if(turn_dir || is_pressed || is_released) //if any menu change interrupt occurs
    {
        menu_update(); //change menu
        turn_dir = 0; //reset position vector
    }
}

```

Biquad.cpp

```

//
// Biquad.cpp
//
// Created by Nigel Redmon on 11/24/12
// EarLevel Engineering: earlevel.com
// Copyright 2012 Nigel Redmon
//
// For a complete explanation of the Biquad code:
// http://www.earlevel.com/main/2012/11/26/biquad-c-source-code/
//

```

```
// License:
//
// This source code is provided as is, without warranty.
// You may copy and distribute verbatim copies of this document.
// You may modify and use this source code to create binary code
// for your own purposes, free or commercial.
//
// Modified by Will Saba and Nick Barany 6/8/2015, all filter formulas
// replaced with SigmaStudio equivalents (type 2 filters)
// See: http://wiki.analog.com/resources/tools-software/sigmastudio/toolbox/filters/general2ndorder
// or, https://ez.analog.com/thread/42007
// for details
```

```
#include <math.h>
#include "Biquad.h"
```

```
Biquad::Biquad() {
    //Default instantiated filter parameters, since peakGain = 0, default filter does nothing
    type = bq_type_peak;
    a0 = 1.0;
    b0 = a1 = a2 = b1 = b2 = 0.0;
    Fc = 0.001042; // ~100 Hz
    Q = 0.707;
    peakGain = 0.0;
    z1 = z2 = 0.0;
}
```

```
Biquad::Biquad(int type, double Fc, double Q, double peakGainDB) {
    setBiquad(type, Fc, Q, peakGainDB);
    z1 = z2 = 0.0;
}
```

```
Biquad::~Biquad() {
}
```

```
void Biquad::setType(int type) {
    this->type = type;
    calcBiquad();
}
```

```
void Biquad::setQ(double Q) {
    this->Q = Q;
    calcBiquad();
}
```

```
void Biquad::setFc(double Fc) {
    this->Fc = Fc;
    calcBiquad();
}
```

```
void Biquad::setPeakGain(double peakGainDB) {
    this->peakGain = peakGainDB;
    calcBiquad();
}
```

```
//Note, Fc is passed as normalized frequency
```

```
void Biquad::setBiquad(int type, double Fc, double Q, double peakGainDB) {
    this->type = type;
    this->Q = Q;
    this->Fc = Fc;
}
```

```

    setPeakGain(peakGainDB);
}

void Biquad::calcBiquad(void) {

    double A = pow(10, peakGain / 40.0); //

    double alpha;
    double gainLinear = 1;//pow(10, peakGain/20); 1 = pass through rest of spectrum, only boost fc
    double w0;
    w0 = 2*M_PI*Fc;
    double dw0,b;
    double S = Q; // Slope parameter passed as Q, keep range 0->2

    // Used in Notch filter, notation comes from Analog Devices thread, see top of file
    dw0 = w0/Q;
    b = 1 / (1 + tan(dw0 / 2));

    switch (this->type) {
        case bq_type_lowpass://BW
            alpha = sin(w0)/2 * 1/sqrt(2);
            a0 = 1+alpha;
            a1 = -2*cos(w0);
            a2 = 1-alpha;
            b0 = (1-cos(w0))*gainLinear/2;
            b1 = 1-cos(w0)*gainLinear;
            b2 = (1-cos(w0))*gainLinear/2;
            break;
        case bq_type_lowpassLR://Same as "lowpass" except alpha, 2nd order stage
            alpha = sin(w0)/(2*(1/sqrt(2)));
            a0 = 1+alpha;
            a1 = -(2*cos(w0));
            a2 = 1-alpha;
            b1 = (1-cos(w0))*gainLinear;
            b0 = b1/2;
            b2 = b0;
            break;

        case bq_type_highpass:
            alpha = sin(w0)/2 * 1/sqrt(2);
            a0 = 1+alpha;
            b0 = (1+cos(w0))*gainLinear/2;
            b1 = -(1+cos(w0))*gainLinear;
            a1 = -2*cos(w0);
            b2 = (1+cos(w0))*gainLinear/2;
            a2 = 1-alpha;
            break;
        case bq_type_highpassLR: // Same as highpass except alpha, 2nd order stage
            alpha = sin(w0)/(2*(1/sqrt(2)));
            a0 = 1+alpha;
            a1 = -2*cos(w0);
            a2 = 1-alpha;
            b1 = -(1+cos(w0))*gainLinear;
            b0 = -b1/2;
            b2 = b0;
            break;

        case bq_type_bandpass: // Not implemented, not useful for this application

            break;
    }
    //Notch filter with variable width based on passed "Q" parameter...

```

```

// altering the "S" or slope parameter defined in Sigma Studio
    case bq_type_notch:
        a0 = 1;
        b0 = gainLinear*b;
        b1 = gainLinear*(-2*b*cos(w0));
        b2 = gainLinear*b;
        a1 = -2*b*cos(w0);
        a2 = (2*b-1);

        break;
// Parametric Filter Type
    case bq_type_peak:
        alpha = sin(w0)/(2.0000*A*Q);
        a0 = 1.0000+alpha/A;
        a1 = -2.0000*cos(w0);
        a2 = 1.0000-alpha/A;
        b0 = (1.0000+alpha*A)*gainLinear;
        b1 = -(2.0000*cos(w0))*gainLinear;
        b2 = (1.0000-alpha*A)*gainLinear;
        /* If you wish to implement an actual "peak" filter (inverse of notch)

        a0 = 1;
        a1 = - 2 * b * cos(w0);
        a2 = (2 * b - 1);
        b0 = gainLinear * (1 - b);
        b1 = 0;
        b2 = gainLinear * (b - 1);
        break;
        */
//Primer for shelving filters:
//Variable Slope -> S, passed as Q, '1' gives a smooth transition,
// greater slope increases transition rate, and yields an overdamped looking frequency response
// lower slope decreases and yields underdamped looking frequency response.
    case bq_type_lowshelf:
        alpha = sin(w0)/2.0000000*sqrt(((double)A+1.000000/A)*(1.0000/S-1)+2);
        a0= (A+1) + (A-1)*cos(w0) + 2*sqrt(A)*alpha;
        a1= -2*((A-1)+(A+1)*cos(w0));
        a2 = (A+1) + (A-1)*cos(w0) - 2*sqrt(A)*alpha;
        b0 = A*((A+1)-(A-1)*cos(w0)+2*sqrt(A)*alpha)*gainLinear;
        b1 = 2*A*((A-1)-(A+1)*cos(w0))*gainLinear;
        b2 = A*((A+1) - (A-1)*cos(w0)-2*sqrt(A)*alpha)*gainLinear;
        break;
    case bq_type_highshelf:

        alpha = sin(w0)/2.000000*sqrt((A+1.000000/A)*(1.0000/S-1)+2);
        a0= (A+1) - (A-1)*cos(w0) + 2*sqrt(A)*alpha;
        a1= 2*((A-1)-(A+1)*cos(w0));
        a2 = (A+1) - (A-1)*cos(w0) - 2*sqrt(A)*alpha;
        b0 = A*((A+1)+(A-1)*cos(w0)+2*sqrt(A)*alpha)*gainLinear;
        b1 = -2*A*((A-1)+(A+1)*cos(w0))*gainLinear;
        b2 = A*((A+1) + (A-1)*cos(w0)-2*sqrt(A)*alpha)*gainLinear;
        break;
} //End of filter types
//Normalize all values by dividing by a0 to make a0 1 (not stored in dsp)
// -> thus only requiring 5 parameters to define each filter
a1 = (double)a1/(double)a0;
a2 = (double)a2/(double)a0;
b0 = (double)b0/(double)a0;
b1 = (double)b1/(double)a0;
b2 = (double)b2/(double)a0;
//invert A1, A2 for parameter ram storage, see Wiki for explanation

```

```

    a1 = a1*-1;
    a2 = a2*-1;

    return;
}

```

Biquad.h

```

//
// Biquad.h
//
// Created by Nigel Redmon on 11/24/12
// EarLevel Engineering: earlevel.com
// Copyright 2012 Nigel Redmon
//
// For a complete explanation of the Biquad code:
// http://www.earlevel.com/main/2012/11/25/biquad-c-source-code/
//
// License:
//
// This source code is provided as is, without warranty.
// You may copy and distribute verbatim copies of this document.
// You may modify and use this source code to create binary code
// for your own purposes, free or commercial.
//
// Edited by Will Saba and Nick Barany -- all calcBiquad equations rewritten based on Sigma Studio (Analog Devices) 5/25/15

```

```

#ifndef Biquad_h
#define Biquad_h

```

```

enum {
    bq_type_lowpass = 0,
    bq_type_highpass,
    bq_type_bandpass,
    bq_type_notch,
    bq_type_peak,
    bq_type_lowshelf,
    bq_type_highshelf,
    // Added LR versions to differentiate these 2nd order Butterworth stages (used in cascade for 4th order LR response)
    bq_type_lowpassLR,
    bq_type_highpassLR
};

```

```

class Biquad {
public:
    Biquad();
    Biquad(int type, double Fc, double Q, double peakGainDB);
    ~Biquad();
    void setType(int type);
    void setQ(double Q);
    void setFc(double Fc);
    void setPeakGain(double peakGainDB);
    void setBiquad(int type, double Fc, double Q, double peakGain);
    float process(float in);
    // Make all these public because need read access -> Serial.print verification
    void calcBiquad(void);

    int type;
    double a0, a1, b0, a2, b1, b2;

    double Fc, Q, peakGain;

```

```

    double z1, z2;
};

inline float Biquad::process(float in) {
    double out = in * a0 + z1;
    z1 = in * a1 + z2 - b1 * out;
    z2 = in * a2 - b2 * out;
    return out;
}

#endif // Biquad_h

```

SigmaStudioFw.h

```

/*
 * File:                SigmaStudioFW.h
 *
 * Description:         SigmaStudio System framework macro and function declarations
 *
 * This file/software originated as an Analog Devices copyrighted product but since has been
 * completely rewritten and stripped of copyrighted content for the express purposes
 * of integrating the arduino uno with the ADAU 1702 EVAL board and helping others
 * trying to do the same. As such:
// This source code is provided as is, without warranty.
// You may copy and distribute verbatim copies of this document.
// You may modify and use this source code to create binary code
// for your own purposes, free or commercial.
 * Purpose:
 *     This document contains various methods of computing and sending parameter data over i2c to the
 *     ADAU 1702 dev board. Individual function comments explain there purposes and in most cases
 *     multiple functions have been implemented where fewer is possible for the sake of readability
 * Last Revised: 6/9/2015 by Will Saba and Nick Barany
 * If there are improvements that can/should be made or you require the PARAM #define file, feel free to contact me,
 * Will Saba, at bartallen101@gmail.com
 */

/*****--#includes -> reference libraries and register address definitions--*****/
#ifndef __SIGMASTUDIOFW_H__
#define __SIGMASTUDIOFW_H__

#include "Biquad.h"
/** Stores location of all default parameters and their respective locations in DSP memory */
#include "main_final_IC_1_PARAM.h"
/** Stores location of all default register settings and their respective locations in DSP memory */
#include "main_final_IC_1_REG.h"

/***** Safeload registers used to load parameters on the fly without audible glitching *****/
#define SAFELoad_ADDR1 0x0815
#define SAFELoad_ADDR2 0x0816
#define SAFELoad_ADDR3 0x0817
#define SAFELoad_ADDR4 0x0818
#define SAFELoad_ADDR5 0x0819
#define SAFELoad_DATA1 0x0810
#define SAFELoad_DATA2 0x0811
#define SAFELoad_DATA3 0x0812
#define SAFELoad_DATA4 0x0813
#define SAFELoad_DATA5 0x0814
// Define double versions of 1 and 0 and 1/sqrt(2) to reduce instances of magic numbers
#define UNO 1.0
#define ZERO 0.0

```

```

#define HALF_POWER 0.707945823669434
// Sub mode return value if found "SUBMODE" as first word on its own line in filterImport
#define SUB_MODE_FLAG_HIGH 2
/** Numeric constant required by system volume control to set the slew rate */
#define VOLUME_STEP 0.000244140625
/** Sample Rate of DSP*/
#define sampleRate 96000.0

/** Experimentally determined I2C address(es) for all inter-board communication */
#define I2C_ADDR_ADAU1702W 0x34 // i2c Write address experimental of ADAU DSP board
#define I2C_ADDR_ADAU1702R 0x50 // i2c Read address experimental of ADAU DSP board
/** Byte length of standard DSP registers = 2 (such as core control) */
#define Address_Length 2
//Unused typedef
//typedef unsigned short ADI_DATA_U16;

/** Define PROGMEM type to store byte constant DSP code in program memory and extract on-demand to overcome memory limitations */
typedef PROGMEM const unsigned char ADI_REG_TYPE;

/*****--Functions that write to DSP--*****/
//Initialiize safeload transfer by toggling respective IST bit in control register//
void set_core_IST_bit(int deviceaddress);
//Writes one register of parameter data passed as a double (function converts to fixed point)//
void safeWriteReg(double coeff,short deviceaddress, short data_reg, short addr_reg, short address);
//Writes passed filter Biquad to the DSP with a sequence of 5 safeWriteReg commands and the necessary IST call//
void safeLoadFilter( Biquad* filterX, int addressStart);
//Writes arbitrarily long number of bytes to DSP memory directly//
void SIGMA_WRITE_REGISTER_BLOCK(int devAddress, int address, int length, ADI_REG_TYPE* pData);
//Writes Parameter Memory in 4 byte chunks, length_reg indicates number of registers (not bytes)//
void SIGMA_WRITE_REGISTER_PARAM(int devAddress, int address, int length_reg, ADI_REG_TYPE* pData);
//Writes Program Memory in 5 byte chunks to DSP memory directly, length_reg indicates number of registers (bytes/5)//
void SIGMA_WRITE_REGISTER_PROG(int devAddress, int address, int length_reg, ADI_REG_TYPE* pData);
//Load volume passed as a decimal less than 1, (double) range 0-> 1.00
void setVolume(double coeff, int deviceaddress);
//Load digital fc (need to divide by sampleRate before passing) into 8 biquad filters representing 4th order LR filter//
void setCrossover(double fc, Biquad* LP1L,Biquad* LP1R,Biquad* LP2L,Biquad* LP2R,Biquad* HP1L,Biquad* HP1R,Biquad* HP2L,Biquad* HP2R);
// Parse a txt file and set filters 1-6 for loading, 7-10 to be implemented with increased uC memory
int importFilter(Biquad* filter1,Biquad* filter2,Biquad* filter3,Biquad* filter4,Biquad* filter5,Biquad* filter6);//,Biquad*
filter7,Biquad* filter8);//,Biquad* filter9, Biquad* filter10);
// Pass address of each parameter location, loads these 3 registers using safeload transfer//
void subModeEnable(int subModeToggleAdd, int rightMuteAdd, int subModeGainAdd);

/*
 * Parameter data format
 */
#define SIGMASTUDIOTYPE_FIXPOINT 0
#define SIGMASTUDIOTYPE_INTEGER 1

/*
 * Write to a single Device register
 */
#define SIGMA_WRITE_REGISTER( devAddress, address, dataLength, data ) {/*TODO: implement macro or define as function*/}

/*
 * TODO: CUSTOM MACRO IMPLEMENTATION
 * Write to multiple Device registers
 */
void setVolume(double coeff, int deviceaddress){
    safeWriteReg( coeff, deviceaddress, SAFELoad_DATA1, SAFELoad_ADDR1, MOD_SWVOL1_ALGO_TARGET_ADDR); // address 1 = 0x01 = target
(volume)
    safeWriteReg( VOLUME_STEP , deviceaddress, SAFELoad_DATA2, SAFELoad_ADDR2, MOD_SWVOL1_ALGO_STEP_ADDR);// address 2 = 0x02 = step

```


(correlates to slew rate, unchanged)

```
    set_core_IST_bit(I2C_ADDR_ADAU1702W);
}
```

```
void setCrossover(double fc, Biquad* LP1L,Biquad* LP1R,Biquad* LP2L,Biquad* LP2R,Biquad* HP1L,Biquad* HP1R,Biquad* HP2L,Biquad* HP2R){
```

```
    LP1L ->setFc(fc);
    LP2L ->setFc(fc);
    LP1R ->setFc(fc);
    LP2R ->setFc(fc);
    HP1L ->setFc(fc);
    HP2L ->setFc(fc);
    HP1R ->setFc(fc);
    HP2R ->setFc(fc);
```

```
//LP1L - alg0_low_filt1
```

```
safeWriteReg (LP1L->b0, I2C_ADDR_ADAU1702W, SAFELoad_DATA1,SAFELoad_ADDR1, MOD_CROSSOVER_ALG0_LOW_FILT1_PARAMB0_ADDR);
safeWriteReg (LP1L->b1, I2C_ADDR_ADAU1702W, SAFELoad_DATA2,SAFELoad_ADDR2, MOD_CROSSOVER_ALG0_LOW_FILT1_PARAMB1_ADDR);
safeWriteReg (LP1L->a1, I2C_ADDR_ADAU1702W, SAFELoad_DATA3,SAFELoad_ADDR3, MOD_CROSSOVER_ALG0_LOW_FILT1_PARAMS1_ADDR);
safeWriteReg (LP1L->b2, I2C_ADDR_ADAU1702W, SAFELoad_DATA4,SAFELoad_ADDR4, MOD_CROSSOVER_ALG0_LOW_FILT1_PARAMB2_ADDR);
safeWriteReg (LP1L->a2, I2C_ADDR_ADAU1702W, SAFELoad_DATA5,SAFELoad_ADDR5, MOD_CROSSOVER_ALG0_LOW_FILT1_PARAMS2_ADDR);
set_core_IST_bit(I2C_ADDR_ADAU1702W);
```

```
//LP2L - alg 1 low_filt 1
```

```
safeWriteReg (LP2L->b0, I2C_ADDR_ADAU1702W, SAFELoad_DATA1,SAFELoad_ADDR1, MOD_CROSSOVER_ALG1_LOW_FILT1_PARAMB0_ADDR);
safeWriteReg (LP2L->b1, I2C_ADDR_ADAU1702W, SAFELoad_DATA2,SAFELoad_ADDR2, MOD_CROSSOVER_ALG1_LOW_FILT1_PARAMB1_ADDR);
safeWriteReg (LP2L->a1, I2C_ADDR_ADAU1702W, SAFELoad_DATA3,SAFELoad_ADDR3, MOD_CROSSOVER_ALG1_LOW_FILT1_PARAMS1_ADDR);
safeWriteReg (LP2L->b2, I2C_ADDR_ADAU1702W, SAFELoad_DATA4,SAFELoad_ADDR4, MOD_CROSSOVER_ALG1_LOW_FILT1_PARAMB2_ADDR);
safeWriteReg (LP2L->a2, I2C_ADDR_ADAU1702W, SAFELoad_DATA5,SAFELoad_ADDR5, MOD_CROSSOVER_ALG1_LOW_FILT1_PARAMS2_ADDR);
set_core_IST_bit(I2C_ADDR_ADAU1702W);
```

```
//LP1R - alg0 low_filt2
```

```
safeWriteReg (LP1R->b0, I2C_ADDR_ADAU1702W, SAFELoad_DATA1,SAFELoad_ADDR1, MOD_CROSSOVER_ALG0_LOW_FILT2_PARAMB0_ADDR);
safeWriteReg (LP1R->b1, I2C_ADDR_ADAU1702W, SAFELoad_DATA2,SAFELoad_ADDR2, MOD_CROSSOVER_ALG0_LOW_FILT2_PARAMB1_ADDR);
safeWriteReg (LP1R->a1, I2C_ADDR_ADAU1702W, SAFELoad_DATA3,SAFELoad_ADDR3, MOD_CROSSOVER_ALG0_LOW_FILT2_PARAMS1_ADDR);
safeWriteReg (LP1R->b2, I2C_ADDR_ADAU1702W, SAFELoad_DATA4,SAFELoad_ADDR4, MOD_CROSSOVER_ALG0_LOW_FILT2_PARAMB2_ADDR);
safeWriteReg (LP1R->a2, I2C_ADDR_ADAU1702W, SAFELoad_DATA5,SAFELoad_ADDR5, MOD_CROSSOVER_ALG0_LOW_FILT2_PARAMS2_ADDR);
set_core_IST_bit(I2C_ADDR_ADAU1702W);
```

```
//LP2R - alg1 low_filt2
```

```
safeWriteReg (LP2R->b0, I2C_ADDR_ADAU1702W, SAFELoad_DATA1,SAFELoad_ADDR1, MOD_CROSSOVER_ALG1_LOW_FILT2_PARAMB0_ADDR);
safeWriteReg (LP2R->b1, I2C_ADDR_ADAU1702W, SAFELoad_DATA2,SAFELoad_ADDR2, MOD_CROSSOVER_ALG1_LOW_FILT2_PARAMB1_ADDR);
safeWriteReg (LP2R->a1, I2C_ADDR_ADAU1702W, SAFELoad_DATA3,SAFELoad_ADDR3, MOD_CROSSOVER_ALG1_LOW_FILT2_PARAMS1_ADDR);
safeWriteReg (LP2R->b2, I2C_ADDR_ADAU1702W, SAFELoad_DATA4,SAFELoad_ADDR4, MOD_CROSSOVER_ALG1_LOW_FILT2_PARAMB2_ADDR);
safeWriteReg (LP2R->a2, I2C_ADDR_ADAU1702W, SAFELoad_DATA5,SAFELoad_ADDR5, MOD_CROSSOVER_ALG1_LOW_FILT2_PARAMS2_ADDR);
set_core_IST_bit(I2C_ADDR_ADAU1702W);
```

```
//HP1L - alg0 high_filt 1
```

```
safeWriteReg (HP1L->b0, I2C_ADDR_ADAU1702W, SAFELoad_DATA1,SAFELoad_ADDR1, MOD_CROSSOVER_ALG0_HIGH_FILT1_PARAMB0_ADDR);
safeWriteReg (HP1L->b1, I2C_ADDR_ADAU1702W, SAFELoad_DATA2,SAFELoad_ADDR2, MOD_CROSSOVER_ALG0_HIGH_FILT1_PARAMB1_ADDR);
safeWriteReg (HP1L->a1, I2C_ADDR_ADAU1702W, SAFELoad_DATA3,SAFELoad_ADDR3, MOD_CROSSOVER_ALG0_HIGH_FILT1_PARAMS1_ADDR);
safeWriteReg (HP1L->b2, I2C_ADDR_ADAU1702W, SAFELoad_DATA4,SAFELoad_ADDR4, MOD_CROSSOVER_ALG0_HIGH_FILT1_PARAMB2_ADDR);
safeWriteReg (HP1L->a2, I2C_ADDR_ADAU1702W, SAFELoad_DATA5,SAFELoad_ADDR5, MOD_CROSSOVER_ALG0_HIGH_FILT1_PARAMS2_ADDR);
set_core_IST_bit(I2C_ADDR_ADAU1702W);
```

```
//HP2L - alg 1 high_filt1
```

```
safeWriteReg (HP2L->b0, I2C_ADDR_ADAU1702W, SAFELoad_DATA1,SAFELoad_ADDR1, MOD_CROSSOVER_ALG1_HIGH_FILT1_PARAMB0_ADDR);
safeWriteReg (HP2L->b1, I2C_ADDR_ADAU1702W, SAFELoad_DATA2,SAFELoad_ADDR2, MOD_CROSSOVER_ALG1_HIGH_FILT1_PARAMB1_ADDR);
safeWriteReg (HP2L->a1, I2C_ADDR_ADAU1702W, SAFELoad_DATA3,SAFELoad_ADDR3, MOD_CROSSOVER_ALG1_HIGH_FILT1_PARAMS1_ADDR);
safeWriteReg (HP2L->b2, I2C_ADDR_ADAU1702W, SAFELoad_DATA4,SAFELoad_ADDR4, MOD_CROSSOVER_ALG1_HIGH_FILT1_PARAMB2_ADDR);
safeWriteReg (HP2L->a2, I2C_ADDR_ADAU1702W, SAFELoad_DATA5,SAFELoad_ADDR5, MOD_CROSSOVER_ALG1_HIGH_FILT1_PARAMS2_ADDR);
set_core_IST_bit(I2C_ADDR_ADAU1702W);
```

```
//HP1R - alg 0 high_filt 2
```

```
safeWriteReg (HP1R->b0, I2C_ADDR_ADAU1702W, SAFELoad_DATA1,SAFELoad_ADDR1, MOD_CROSSOVER_ALG0_HIGH_FILT2_PARAMB0_ADDR);
safeWriteReg (HP1R->b1, I2C_ADDR_ADAU1702W, SAFELoad_DATA2,SAFELoad_ADDR2, MOD_CROSSOVER_ALG0_HIGH_FILT2_PARAMB1_ADDR);
```

```

safeWriteReg (HP1R->a1, I2C_ADDR_ADAU1702W, SAFELoad_DATA3,SAFELoad_ADDR3, MOD_CROSSOVER_ALG0_HIGH_FILT2_PARAMA1_ADDR);
safeWriteReg (HP1R->b2, I2C_ADDR_ADAU1702W, SAFELoad_DATA4,SAFELoad_ADDR4, MOD_CROSSOVER_ALG0_HIGH_FILT2_PARAMB2_ADDR);
safeWriteReg (HP1R->a2, I2C_ADDR_ADAU1702W, SAFELoad_DATA5,SAFELoad_ADDR5, MOD_CROSSOVER_ALG0_HIGH_FILT2_PARAMA2_ADDR);
set_core_IST_bit(I2C_ADDR_ADAU1702W);
//HP2R - alg 1 high filt 2
safeWriteReg (HP2R->b0, I2C_ADDR_ADAU1702W, SAFELoad_DATA1,SAFELoad_ADDR1, MOD_CROSSOVER_ALG1_HIGH_FILT2_PARAMB0_ADDR);
safeWriteReg (HP2R->b1, I2C_ADDR_ADAU1702W, SAFELoad_DATA2,SAFELoad_ADDR2, MOD_CROSSOVER_ALG1_HIGH_FILT2_PARAMB1_ADDR);
safeWriteReg (HP2R->a1, I2C_ADDR_ADAU1702W, SAFELoad_DATA3,SAFELoad_ADDR3, MOD_CROSSOVER_ALG1_HIGH_FILT2_PARAMA1_ADDR);
safeWriteReg (HP2R->b2, I2C_ADDR_ADAU1702W, SAFELoad_DATA4,SAFELoad_ADDR4, MOD_CROSSOVER_ALG1_HIGH_FILT2_PARAMB2_ADDR);
safeWriteReg (HP2R->a2, I2C_ADDR_ADAU1702W, SAFELoad_DATA5,SAFELoad_ADDR5, MOD_CROSSOVER_ALG1_HIGH_FILT2_PARAMA2_ADDR);
set_core_IST_bit(I2C_ADDR_ADAU1702W);
delay(50);
}

```

//Next 3 functions write the boot code to the DSP and initialize sound

//Writes program memory in 5 byte chunks

```

void SIGMA_WRITE_REGISTER_PROG(int devAddress, int address, int length_reg, ADI_REG_TYPE* pData){

```

```

    int addtemp = 0;
    int addcount =0, regcount=0;
    addtemp = address;
    #ifndef cbi
    #define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
    #endif
    for(regcount=0;regcount<length_reg;regcount++)
    {
        Wire.beginTransaction(devAddress);
        Wire.write(((addtemp +regcount) & 0xFF00) >> 8);
        Wire.write((addtemp+regcount) & 0x00FF);

        Wire.write(pgm_read_byte(pData+addcount));
        Wire.write(pgm_read_byte(pData+addcount+1));
        Wire.write(pgm_read_byte(pData+addcount+2));
        Wire.write(pgm_read_byte(pData+addcount+3));
        Wire.write(pgm_read_byte(pData+addcount+4));
        addcount+=5;
        Wire.endTransmission();
    }
}

```

//Writes parameter memory in 4 byte chunks

```

void SIGMA_WRITE_REGISTER_PARAM(int devAddress, int address, int length_reg, ADI_REG_TYPE* pData){

```

```

    int addtemp = 0;
    int addcount =0, regcount=0;
    addtemp = address;
    #ifndef cbi
    #define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
    #endif
    // Serial.begin(9600);
    for(regcount=0;regcount<length_reg;regcount++)
    {
        Wire.beginTransaction(devAddress);
        Wire.write(((addtemp +regcount) & 0xFF00) >> 8);
        Wire.write((addtemp+regcount) & 0x00FF);
        Wire.write(pgm_read_byte(pData+addcount));

        // Serial.print(pgm_read_byte(pData+addcount),HEX);
        // Serial.print(", ");
        Wire.write(pgm_read_byte(pData+addcount+1));
        // Serial.print(pgm_read_byte(pData+addcount+1),HEX);
        // Serial.print(", ");
        Wire.write(pgm_read_byte(pData+addcount+2));
        // Serial.print(pgm_read_byte(pData+addcount+2),HEX);
    }
}

```

```

//      Serial.print(" ");
Wire.write(pgm_read_byte(pData+addcount+3));
//      Serial.print(pgm_read_byte(pData+addcount+3),HEX);
//      Serial.println(" ");
      addcount+=4;
      Wire.endTransmission();
    }
//  Serial.end();
}
//Write small continuous blocks of data to DSP, such as the core register and other hardware parameters
void SIGMA_WRITE_REGISTER_BLOCK(int devAddress, int address, int length, ADI_REG_TYPE* pData){
  #ifndef cbi
  #define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
  #endif
  int i=0;

  Wire.beginTransmission(devAddress);

  Wire.write(((address) & 0xFF00) >> 8);
  Wire.write((address) & 0x00FF);
  //      Serial.println(((address) & 0xFF00) >> 8,HEX);
  //      Serial.println((address) & 0x00FF,HEX);

  for (i>0;i<length;i++){
  //      Serial.println(pgm_read_byte(pData+i),HEX);
    Wire.write(pgm_read_byte(pData+i));
  }
  Wire.endTransmission();
}

/*
 * Convert a floating-point value to SigmaDSP (5.23) fixed point format
 */
#define SIGMASTUDIOTYPE_FIXPOINT_CONVERT( _value )(_value * 0x00800000)

//Writes one filter based on 5 passed parameter values and a starting destination address
void safeLoadFilter( Biquad* filterX, int addressStart)
{
  //Load 5 filter coefficients into DSP safe-load registers, note a0 corresponds to b0 in SigmaDSP due to a naming convention conflict
  // Function call format: safeLoadFilter(filter1, address);

  safeWriteReg (filterX->b0, I2C_ADDR_ADAU1702W, SAFELoad_DATA1,SAFELoad_ADDR1, addressStart);
  safeWriteReg (filterX->b1, I2C_ADDR_ADAU1702W, SAFELoad_DATA2,SAFELoad_ADDR2, (addressStart+1));
  safeWriteReg (filterX->b2, I2C_ADDR_ADAU1702W, SAFELoad_DATA3,SAFELoad_ADDR3, (addressStart+2));
  safeWriteReg (filterX->a1, I2C_ADDR_ADAU1702W, SAFELoad_DATA4,SAFELoad_ADDR4, (addressStart+3));
  safeWriteReg (filterX->a2, I2C_ADDR_ADAU1702W, SAFELoad_DATA5,SAFELoad_ADDR5, (addressStart+4));
  //Send the OK to transmit filter
  set_core_IST_bit(I2C_ADDR_ADAU1702W);
}

//Writes 1 parameter register via Safeload Write over I2C, doesn't include IST call
void safeWriteReg(double coeff,short deviceaddress, short data_reg, short addr_reg, short address)
{
  long fixedPtCoeff = 0;
  unsigned char bytes[4]={0}; // 4 x Chars to "carry" the broken Float value

  fixedPtCoeff = SIGMASTUDIOTYPE_FIXPOINT_CONVERT(coeff) ; //Convert to 5.23 fixed point format
  // Separate fixed point variable into 4 byte sized chunks for transmission to the DSP
  bytes[0] = (fixedPtCoeff >> 24) & 0xFF;
  bytes[1] = (fixedPtCoeff >> 16) & 0xFF;

```

```

bytes[2] = (fixedPtCoeff >> 8) & 0xFF;
bytes[3] = fixedPtCoeff & 0xFF;

//Transmit Data//
Wire.beginTransmission(deviceaddress); // 0x34 (write port)
//send data register address 1 byte at a time
Wire.write(((data_reg >> 8) & 0xFF)); // send upper byte safeload data register
Wire.write((data_reg & 0xFF)); // send lower byte
Wire.write(0x00); //load 00 into byte 3 of safeload register as required by safeload write format
Wire.write(bytes,4); // write 4 bytes of parameter data
Wire.endTransmission();

Wire.beginTransmission(deviceaddress); // 0x34 (write port)
//send address register 1 byte at a time
Wire.write((addr_reg >> 8) & 0xFF); // send upper byte (0x08), safeload address register
Wire.write(addr_reg & 0xFF); // send lower byte
Wire.write((address >> 8) & 0xFF); // destination register
Wire.write(address & 0xFF);

Wire.endTransmission();
}

void subModeEnable(int subModeToggleAdd, int rightMuteAdd, int subModeGainAdd){
//Enable submode toggle by setting mute to "1" -> pass audio
safeWriteReg(UNO,I2C_ADDR_ADAU1702W, SAFELoad_DATA1, SAFELoad_ADDR1, subModeToggleAdd);
//Enable rightMute
safeWriteReg(ZERO,I2C_ADDR_ADAU1702W, SAFELoad_DATA2, SAFELoad_ADDR2, rightMuteAdd);
// Balance sub with mains by allowing half of left and half of right into the bass mix
safeWriteReg(HALF_POWER,I2C_ADDR_ADAU1702W, SAFELoad_DATA3, SAFELoad_ADDR3, subModeGainAdd);
// Transfer the settings
set_core_IST_bit(I2C_ADDR_ADAU1702W);
}

void set_core_IST_bit(int deviceaddress)
{
/*set the initiate safeload transfer bit ( 5th bit )
in the core control register to initiate the loading into RAM*/
Wire.beginTransmission(deviceaddress);
Wire.write(0x08);
Wire.write(0x1C); //address of DSP core control: 0x081C. Holds 2bytes data.
Wire.write(0x00);
Wire.write(0x3D); // 96 kHz, IST high
Wire.endTransmission();
}

int importFilter(Biquad* filter1, Biquad* filter2,Biquad* filter3,Biquad* filter4,Biquad* filter5,Biquad* filter6){//, Biquad*
filter7,Biquad* filter8}{//,Biquad* filter9, Biquad* filter10}{

const int MAX_CHARS_PER_LINE = 100;
const int MAX_TOKENS_PER_LINE = 20;
const char* const DELIMITER = " ";
int filterIndex = 1;
double fcTemp = 0;
double gainTemp = 0;
double qTemp = 0;
int address = 0, n = 0, i = 0;
char charTemp;
int type;
int subModeFlag = 0, memlength = 0;

```

```

/*
// If Had file system
/*FILE *fr; // Declare file pointer
// create a file-reading object
fr = fopen("test.txt","r"); // open a file
*/
// Load first character into temp variable (need for != Null condition)
charTemp = EEPROM.read(address);
address++;
while (charTemp != NULL){ // stop at end of EEPROM

    char buf[MAX_CHARS_PER_LINE];
    const char* token[MAX_TOKENS_PER_LINE] = {}; // initialize to 0
    while (charTemp != '\n'){
        // Throw out commas
        if (charTemp!= ','){
            buf[i] = charTemp;
        }
        else{
            i--; //Still looking for a valid character (counteracts i++)
        }
        charTemp = EEPROM.read(address);
        address++;
        i++;
    }
    buf[i] = '\0';
    // Serial.println(buf);
    // Reset buf index
    i = 0;
    // parse the line // first token
    token[0] = strtok(buf, DELIMITER);
    //Serial.println(token[0]);
    if (token[0]) // zero if line is blank
    {
        for (n = 1; n < MAX_TOKENS_PER_LINE; n++)
        {
            token[n] = strtok(0, DELIMITER); // subsequent tokens
            if (!token[n])
                break; // no more tokens
            else;
        }
        // Serial.println(token[n]); // Print each parsed word
    }
}
// Process a given line -> set filter
// If declared submode, rest of line will be blank, set flag and go to next line//
    if (!strcmp(token[0], "SUBMODE")){
        subModeFlag = 1;
    }
// This is used to ignore header lines that define REW parameters//
    else if (!strcmp(token[0], "Filter")){ // filter may occur
// If filter is disabled, don't implement it
        if(!strcmp(token[2], "ON")){ //filter is enabled
// Compare substring to find what type of filter is
// contained in the 4th word (see format)

// Note: all token[x] refer to the word # in the line - 1 (index starts at 0)

            if(!strcmp(token[3], "B",1)){ // Peak filter

                fcTemp = atof(token[5]);
                gainTemp = atof(token[8]);

```

```

    qTemp = atof(token[11]);
    type = bq_type_peak;
}
else if(!strncmp(token[3],"LS",2)){ // lowshelf
    qTemp = 1;
    fcTemp = atof(token[6]);
    gainTemp = atof(token[9]);
    type = bq_type_lowshelf;
}
else if(!strncmp(token[3],"HS",2)){ // highshelf
    qTemp = 1;
    fcTemp = atof(token[6]);
    gainTemp = atof(token[9]);
    type = bq_type_highshelf;
}
else if(!strncmp(token[3],"HP",2)){ // highpass 2nd order butterworth
    qTemp = 1;
    gainTemp = 0;
    fcTemp = atof(token[5]); // change to index 6 if include 12dB/24dB designation in input file
    type = bq_type_highpassLR;
}
else if(!strncmp(token[3],"LP",2)){ // lowpass 2nd order butterworth stage
    qTemp = 1;
    gainTemp = 0;
    fcTemp = atof(token[5]); // change to index 6 if include 12dB/24dB designation in input file
    type = bq_type_lowpassLR;
}

}
else if(!strncmp(token[3],"NOTCH",5)){ // notch filter
    qTemp = atof(token[8]); // Slope parameter, 0 to 2
    gainTemp = 0;
    fcTemp = atof(token[5]); //
    type = bq_type_notch;
}

}
else if(!strncmp(token[3],"NONE",4)||!strncmp(token[3],"Modal",4)){
    gainTemp = 0;
    qTemp = 0.707;
    type = bq_type_lowshelf;
}
}
// Note: Modal filter not supported inserted values are default "do nothing" filter
// Bypass filters 9-10 for memory conservation

    if (filterIndex == 7){
        filterIndex = 999; // out of range
    }

    switch(filterIndex){
        case 1:
            filter1 ->setBiquad(type, fcTemp / sampleRate, qTemp, gainTemp);
            break;

        case 2:
            filter2 ->setBiquad(type, fcTemp / sampleRate, qTemp, gainTemp);
            break;

        case 3:
            filter3 ->setBiquad(type, fcTemp / sampleRate, qTemp, gainTemp);
            break;

        case 4:

```

```

        filter4 ->setBiquad(type, fcTemp / sampleRate, qTemp, gainTemp);
        break;

        case 5:
        filter5 ->setBiquad(type, fcTemp / sampleRate, qTemp, gainTemp);
        break;

        case 6:
        filter6 ->setBiquad(type, fcTemp / sampleRate, qTemp, gainTemp);
        break;

/*
        case 7:
        filter7 ->setBiquad(type, fcTemp / sampleRate, qTemp, gainTemp);
        break;

        case 8:
        filter8 ->setBiquad(type, fcTemp / sampleRate, qTemp, gainTemp);
        break;

        case 9:
        filter9 ->setBiquad(type, fcTemp / sampleRate, qTemp, gainTemp);
        break;

        case 10:
        filter10 ->setBiquad(type, fcTemp / sampleRate, qTemp, gainTemp);
        break;

*/

        default:
        break; //Do nothing
    } //End case switch
    filterIndex++; // filter implemented, go to next open filter
} //Ends 'on if'

} //Ends 'filter if'
charTemp = EEPROM.read(address); // Read out start of next line
address++;
} //Done with reading

if (subModeFlag){
return SUB_MODE_FLAG_HIGH; // evaluates to 2
}
else
return 0;
}
#endif

```

SigmaStudio Main

```

/*
* File:      C:\Users\wsaba\Google Drive\Senior Project\uC and DSP Code\EXPERIMENT\main_final_IC_1.h
*
* Created:   Monday, June 01, 2015 9:43:02 AM
* Description: EXP:IC 1 program data.
*
* This software is distributed in the hope that it will be useful,
* but is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
* CONDITIONS OF ANY KIND, without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
*
* This software may only be used to program products purchased from

```

```

* Analog Devices for incorporation by you into audio products that
* are intended for resale to audio product end users. This software
* may not be distributed whole or in any part to third parties.
*
* Copyright ©2015 Analog Devices, Inc. All rights reserved.
*
* Detailed Description: Besides the auto-generated Sigma Studio DSP
* boot code stored in Program_Data_IC_1, Param_Data_IC_1 and the corresponding
* pointers appended _loc, This file contains the lookup tables used to
* Fc for the parametric EQ and Crossover.
* Further, #defines have been added for the register count for both program
* and parameter memory as required for the modified boot sequence functions.
* See boot sequence comments for more details (bottom).
*
* Purpose: This file stores numeric constants for use in booting and
* editing the DSP.
*/
#ifndef __MAIN_FINAL_IC_1_H__
#define __MAIN_FINAL_IC_1_H__

#include "SigmaStudioFW.h"
#include "main_final_IC_1_REG.h"

#define DEVICE_ARCHITECTURE_IC_1      "ADAU1702"
#define DEVICE_ADDR_IC_1              0x34
#define HP_LOOKUP_SIZE 280

/**** Store lookup arrays for highpass, frequency, gain, and Q. See if need to store them in progmem.
**** Update q,gain,fc,db lookup when solution found */

// Lookup table logarithmically spaced spanning 20 Hz to 20 kHz, used in crossover setting
int fcLookup [173] = {
20,21,22,23,24,25,26,27,28,29,30,32,34,36,38, // 0-14
40,42,44,46,48,50,52,54,56,58,60,62,64,66,68, //14-29
70,72,74,76,78,80,82,84,86,88,90,92,94,96,98, //30-44
100,107,114,120,128,135,142,152,162,172,    //45-54
180,192,202,212,227,240,257,272,287,300,    //55-64
317,332,347,360,382,402,422,442,462,482,    //65-74
502,540,578,616,654,692,730,770,810,850,    //75-84
900,950,1000,1050,1100,1150,1200,1250,1300,1350, //85-94
1400,1450,1500,1550,1650,1750,1850,1950,2050,2150, //95-104
2250,2350,2450,2550,2650,2750,2850,2950,3050,3150, //105-114
3250,3350,3450,3550,3650,3750,3850,3950,4050,4200, //115-124
4350,4500,4650,4800,4950,5100,5250,5400,5550,5700, //125-134
5850,6000,6150,6300,6450,6600,6750,6900,7050,7200, //135-144
7350,7500,7650,7800,7950,8100,8250,8400,8550,8700, //145-154
8850,9000,9150,9300,9450,9600,9750,9900,11000,12100, //155-164
13000,14000,15000,16000,17000,18000,19000,20000, //165-173
};
//int limiterIndex = 41; // used to initiate limiters to -0 dB via dBLookup for a full scale output range.

// Each _loc variable stores the location in progmem of first byte

/* DSP Program Data */
#define PROGRAM_SIZE_IC_1 2560
#define PROGRAM_SIZE_IC_2 512 // Number of Prog registers to write during boot-up

#define PROGRAM_ADDR_IC_1 1024

ADI_REG_TYPE Program_Data_IC_1[PROGRAM_SIZE_IC_1] = {
0x00, 0x00, 0x00, 0x00, 0x01,

```


0x00, 0x00, 0x00, 0xE8, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x00, 0x08, 0x00, 0xE8, 0x01,
0x00, 0x02, 0x00, 0x20, 0x01,
0x00, 0x10, 0x00, 0xE2, 0x01,
0x00, 0x0A, 0x00, 0x20, 0x01,
0x00, 0x18, 0x00, 0xE2, 0x01,
0xFF, 0xF2, 0x01, 0x20, 0x01,
0x00, 0x21, 0x08, 0x22, 0x41,
0x00, 0x40, 0x00, 0xE2, 0x01,
0x00, 0x31, 0x08, 0x20, 0x01,
0x00, 0x21, 0x08, 0x34, 0x01,
0x00, 0x42, 0x02, 0x22, 0x01,
0x00, 0x28, 0x00, 0xE2, 0x01,
0x00, 0x28, 0x00, 0xC0, 0x01,
0x00, 0x38, 0x00, 0xF2, 0x01,
0x00, 0x17, 0xFF, 0x20, 0x01,
0x00, 0x58, 0x00, 0xE2, 0x01,
0x00, 0x1F, 0xFF, 0x20, 0x01,
0x00, 0x70, 0x00, 0xE2, 0x01,
0x00, 0x82, 0x06, 0x20, 0x01,
0x00, 0x7A, 0x07, 0x22, 0x01,
0x00, 0x5A, 0x03, 0x22, 0x01,
0x00, 0x52, 0x04, 0x22, 0x01,
0x00, 0x4A, 0x05, 0x22, 0x01,
0x00, 0x88, 0x00, 0xE2, 0x01,
0x00, 0x9A, 0x06, 0x20, 0x01,
0x00, 0x92, 0x07, 0x22, 0x01,
0x00, 0x72, 0x03, 0x22, 0x01,
0x00, 0x6A, 0x04, 0x22, 0x01,
0x00, 0x62, 0x05, 0x22, 0x01,
0x00, 0xA0, 0x00, 0xE2, 0x01,
0x00, 0xB2, 0x0B, 0x20, 0x01,
0x00, 0xAA, 0x0C, 0x22, 0x01,
0x00, 0x8A, 0x08, 0x22, 0x01,
0x00, 0x82, 0x09, 0x22, 0x01,
0x00, 0x7A, 0x0A, 0x22, 0x01,
0x00, 0xB8, 0x00, 0xE2, 0x01,
0x00, 0xCA, 0x0B, 0x20, 0x01,
0x00, 0xC2, 0x0C, 0x22, 0x01,
0x00, 0xA2, 0x08, 0x22, 0x01,
0x00, 0x9A, 0x09, 0x22, 0x01,
0x00, 0x92, 0x0A, 0x22, 0x01,
0x00, 0xD0, 0x00, 0xE2, 0x01,
0x00, 0xE2, 0x10, 0x20, 0x01,
0x00, 0xDA, 0x11, 0x22, 0x01,
0x00, 0xBA, 0x0D, 0x22, 0x01,
0x00, 0xB2, 0x0E, 0x22, 0x01,
0x00, 0xAA, 0x0F, 0x22, 0x01,
0x00, 0xE8, 0x00, 0xE2, 0x01,
0x00, 0xFA, 0x10, 0x20, 0x01,
0x00, 0xF2, 0x11, 0x22, 0x01,
0x00, 0xD2, 0x0D, 0x22, 0x01,
0x00, 0xCA, 0x0E, 0x22, 0x01,
0x00, 0xC2, 0x0F, 0x22, 0x01,
0x01, 0x00, 0x00, 0xE2, 0x01,
0x01, 0x12, 0x15, 0x20, 0x01,
0x01, 0x0A, 0x16, 0x22, 0x01,
0x00, 0xEA, 0x12, 0x22, 0x01,
0x00, 0xE2, 0x13, 0x22, 0x01,
0x00, 0xDA, 0x14, 0x22, 0x01,

0x01, 0x18, 0x00, 0xE2, 0x01,
0x01, 0x2A, 0x15, 0x20, 0x01,
0x01, 0x22, 0x16, 0x22, 0x01,
0x01, 0x02, 0x12, 0x22, 0x01,
0x00, 0xFA, 0x13, 0x22, 0x01,
0x00, 0xF2, 0x14, 0x22, 0x01,
0x01, 0x30, 0x00, 0xE2, 0x01,
0x01, 0x42, 0x1A, 0x20, 0x01,
0x01, 0x3A, 0x1B, 0x22, 0x01,
0x01, 0x1A, 0x17, 0x22, 0x01,
0x01, 0x12, 0x18, 0x22, 0x01,
0x01, 0x0A, 0x19, 0x22, 0x01,
0x01, 0x48, 0x00, 0xE2, 0x01,
0x01, 0x5A, 0x1A, 0x20, 0x01,
0x01, 0x52, 0x1B, 0x22, 0x01,
0x01, 0x32, 0x17, 0x22, 0x01,
0x01, 0x2A, 0x18, 0x22, 0x01,
0x01, 0x22, 0x19, 0x22, 0x01,
0x01, 0x60, 0x00, 0xE2, 0x01,
0x01, 0x72, 0x1F, 0x20, 0x01,
0x01, 0x6A, 0x20, 0x22, 0x01,
0x01, 0x4A, 0x1C, 0x22, 0x01,
0x01, 0x42, 0x1D, 0x22, 0x01,
0x01, 0x3A, 0x1E, 0x22, 0x01,
0x01, 0x78, 0x00, 0xE2, 0x01,
0x01, 0x8A, 0x1F, 0x20, 0x01,
0x01, 0x82, 0x20, 0x22, 0x01,
0x01, 0x62, 0x1C, 0x22, 0x01,
0x01, 0x5A, 0x1D, 0x22, 0x01,
0x01, 0x52, 0x1E, 0x22, 0x01,
0x01, 0x90, 0x00, 0xE2, 0x01,
0x01, 0xA2, 0x24, 0x20, 0x01,
0x01, 0x9A, 0x25, 0x22, 0x01,
0x01, 0x7A, 0x21, 0x22, 0x01,
0x01, 0x72, 0x22, 0x22, 0x01,
0x01, 0x6A, 0x23, 0x22, 0x01,
0x01, 0xA8, 0x00, 0xE2, 0x01,
0x01, 0xBA, 0x24, 0x20, 0x01,
0x01, 0xB2, 0x25, 0x22, 0x01,
0x01, 0x92, 0x21, 0x22, 0x01,
0x01, 0x8A, 0x22, 0x22, 0x01,
0x01, 0x82, 0x23, 0x22, 0x01,
0x01, 0xC0, 0x00, 0xE2, 0x01,
0x01, 0xD2, 0x29, 0x20, 0x01,
0x01, 0xCA, 0x2A, 0x22, 0x01,
0x01, 0xAA, 0x26, 0x22, 0x01,
0x01, 0xA2, 0x27, 0x22, 0x01,
0x01, 0x9A, 0x28, 0x22, 0x01,
0x01, 0xD8, 0x00, 0xE2, 0x01,
0x01, 0xEA, 0x29, 0x20, 0x01,
0x01, 0xE2, 0x2A, 0x22, 0x01,
0x01, 0xC2, 0x26, 0x22, 0x01,
0x01, 0xBA, 0x27, 0x22, 0x01,
0x01, 0xB2, 0x28, 0x22, 0x01,
0x01, 0xF0, 0x00, 0xE2, 0x01,
0x02, 0x02, 0x2E, 0x20, 0x01,
0x01, 0xFA, 0x2F, 0x22, 0x01,
0x01, 0xDA, 0x2B, 0x22, 0x01,
0x01, 0xD2, 0x2C, 0x22, 0x01,
0x01, 0xCA, 0x2D, 0x22, 0x01,
0x02, 0x08, 0x00, 0xE2, 0x01,

0x02, 0x1A, 0x2E, 0x20, 0x01,
0x02, 0x12, 0x2F, 0x22, 0x01,
0x01, 0xF2, 0x2B, 0x22, 0x01,
0x01, 0xEA, 0x2C, 0x22, 0x01,
0x01, 0xE2, 0x2D, 0x22, 0x01,
0x02, 0x20, 0x00, 0xE2, 0x01,
0x02, 0x32, 0x33, 0x20, 0x01,
0x02, 0x2A, 0x34, 0x22, 0x01,
0x02, 0x0A, 0x30, 0x22, 0x01,
0x02, 0x02, 0x31, 0x22, 0x01,
0x01, 0xFA, 0x32, 0x22, 0x01,
0x02, 0x38, 0x00, 0xE2, 0x01,
0x02, 0x4A, 0x33, 0x20, 0x01,
0x02, 0x42, 0x34, 0x22, 0x01,
0x02, 0x22, 0x30, 0x22, 0x01,
0x02, 0x1A, 0x31, 0x22, 0x01,
0x02, 0x12, 0x32, 0x22, 0x01,
0x02, 0x50, 0x00, 0xE2, 0x01,
0x02, 0x3A, 0x35, 0x20, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x02, 0x70, 0x00, 0xE2, 0x01,
0x02, 0x9A, 0x38, 0x20, 0x01,
0x02, 0x92, 0x3A, 0x22, 0x01,
0x02, 0x82, 0x38, 0x34, 0x01,
0x02, 0x7A, 0x3A, 0x22, 0x01,
0x02, 0x72, 0x36, 0x22, 0x01,
0x02, 0x6A, 0x37, 0x22, 0x01,
0x02, 0x62, 0x39, 0x22, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x02, 0x88, 0x00, 0xE2, 0x01,
0x02, 0xA0, 0x00, 0xF2, 0x01,
0x02, 0xCA, 0x3D, 0x20, 0x01,
0x02, 0xC2, 0x3F, 0x22, 0x01,
0x02, 0xB2, 0x3D, 0x34, 0x01,
0x02, 0xAA, 0x3F, 0x22, 0x01,
0x02, 0x8A, 0x3B, 0x22, 0x01,
0x02, 0x82, 0x3C, 0x22, 0x01,
0x02, 0x7A, 0x3E, 0x22, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x02, 0xB8, 0x00, 0xE2, 0x01,
0x03, 0xF8, 0x00, 0xE2, 0x01,
0x02, 0xD0, 0x00, 0xF2, 0x01,
0x02, 0x39, 0x08, 0x20, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x02, 0xE8, 0x00, 0xE2, 0x01,
0x03, 0x12, 0x42, 0x20, 0x01,
0x03, 0x0A, 0x44, 0x22, 0x01,
0x02, 0xFA, 0x42, 0x34, 0x01,
0x02, 0xF2, 0x44, 0x22, 0x01,
0x02, 0xEA, 0x40, 0x22, 0x01,
0x02, 0xE2, 0x41, 0x22, 0x01,
0x02, 0xDA, 0x43, 0x22, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x03, 0x00, 0x00, 0xE2, 0x01,
0x03, 0x18, 0x00, 0xF2, 0x01,
0x03, 0x42, 0x47, 0x20, 0x01,
0x03, 0x3A, 0x49, 0x22, 0x01,
0x03, 0x2A, 0x47, 0x34, 0x01,
0x03, 0x22, 0x49, 0x22, 0x01,
0x03, 0x02, 0x45, 0x22, 0x01,
0x02, 0xFA, 0x46, 0x22, 0x01,

0x02, 0xF2, 0x48, 0x22, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x03, 0x30, 0x00, 0xE2, 0x01,
0x02, 0x58, 0x00, 0xE2, 0x01,
0x03, 0x48, 0x00, 0xF2, 0x01,
0x02, 0x52, 0x4A, 0x20, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x03, 0x68, 0x00, 0xE2, 0x01,
0x03, 0x7A, 0x4D, 0x20, 0x01,
0x03, 0x72, 0x4F, 0x22, 0x01,
0x03, 0x6A, 0x4B, 0x22, 0x01,
0x03, 0x62, 0x4C, 0x22, 0x01,
0x03, 0x5A, 0x4E, 0x22, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x03, 0x80, 0x00, 0xE2, 0x01,
0x03, 0x92, 0x52, 0x20, 0x01,
0x03, 0x8A, 0x54, 0x22, 0x01,
0x03, 0x82, 0x50, 0x22, 0x01,
0x03, 0x7A, 0x51, 0x22, 0x01,
0x03, 0x72, 0x53, 0x22, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x03, 0x98, 0x00, 0xE2, 0x01,
0x04, 0x10, 0x00, 0xE2, 0x01,
0x02, 0x51, 0x08, 0x20, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x03, 0xB0, 0x00, 0xE2, 0x01,
0x03, 0xC2, 0x57, 0x20, 0x01,
0x03, 0xBA, 0x59, 0x22, 0x01,
0x03, 0xB2, 0x55, 0x22, 0x01,
0x03, 0xAA, 0x56, 0x22, 0x01,
0x03, 0xA2, 0x58, 0x22, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x03, 0xC8, 0x00, 0xE2, 0x01,
0x03, 0xDA, 0x5C, 0x20, 0x01,
0x03, 0xD2, 0x5E, 0x22, 0x01,
0x03, 0xCA, 0x5A, 0x22, 0x01,
0x03, 0xC2, 0x5B, 0x22, 0x01,
0x03, 0xBA, 0x5D, 0x22, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x03, 0xE0, 0x00, 0xE2, 0x01,
0x03, 0x50, 0x00, 0xE2, 0x01,
0x02, 0x59, 0x08, 0x20, 0x01,
0xFF, 0x80, 0x00, 0x02, 0x01,
0x03, 0x51, 0x08, 0x20, 0x01,
0xFF, 0x78, 0x00, 0x02, 0x01,
0x04, 0x22, 0x62, 0x20, 0x01,
0x04, 0x1A, 0x63, 0x22, 0x01,
0x03, 0xFA, 0x5F, 0x22, 0x01,
0x03, 0xF2, 0x60, 0x22, 0x01,
0x03, 0xEA, 0x61, 0x22, 0x01,
0x04, 0x28, 0x00, 0xE2, 0x01,
0x04, 0x3A, 0x62, 0x20, 0x01,
0x04, 0x32, 0x63, 0x22, 0x01,
0x04, 0x12, 0x5F, 0x22, 0x01,
0x04, 0x0A, 0x60, 0x22, 0x01,
0x04, 0x02, 0x61, 0x22, 0x01,
0x04, 0x40, 0x00, 0xE2, 0x01,
0x04, 0x42, 0x64, 0x20, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x04, 0x48, 0x00, 0xE2, 0x01,
0x04, 0x42, 0x64, 0x20, 0x01,

0x00, 0x00, 0x00, 0x00, 0x01,
0x04, 0x50, 0x00, 0xE2, 0x01,
0x04, 0x4A, 0x65, 0x20, 0x01,
0x04, 0x58, 0x00, 0xE2, 0x01,
0x04, 0x52, 0x66, 0x20, 0x01,
0x04, 0x60, 0x00, 0xE2, 0x01,
0x04, 0x2A, 0x67, 0x20, 0x01,
0x04, 0x5A, 0x67, 0x22, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x04, 0x68, 0x00, 0xE2, 0x01,
0x04, 0x61, 0x08, 0x20, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x04, 0xC8, 0x00, 0xF0, 0x01,
0xFF, 0xE9, 0x08, 0x22, 0x01,
0x04, 0xC0, 0x00, 0xF2, 0x01,
0x04, 0xB1, 0x08, 0x20, 0x01,
0x04, 0xB2, 0x74, 0x22, 0x41,
0x04, 0xC2, 0x74, 0x22, 0x01,
0x04, 0xA1, 0x08, 0x34, 0x01,
0x04, 0xA2, 0x74, 0x22, 0x41,
0x04, 0xCA, 0x74, 0x22, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x04, 0xA8, 0x00, 0xE2, 0x01,
0x04, 0xB8, 0x00, 0xF2, 0x01,
0x04, 0xD0, 0x00, 0xE2, 0x01,
0xFF, 0xF2, 0x70, 0x40, 0x01,
0x04, 0xD1, 0x08, 0x20, 0x09,
0x04, 0xD2, 0x68, 0x20, 0x01,
0xFF, 0xF2, 0x69, 0x22, 0x01,
0x04, 0x70, 0x00, 0xE2, 0x01,
0x04, 0xD2, 0x6A, 0x20, 0x01,
0xFF, 0xF2, 0x6B, 0x22, 0x01,
0x04, 0x70, 0x00, 0xE2, 0x23,
0xFF, 0xF2, 0x71, 0x40, 0x01,
0x04, 0xD1, 0x08, 0x20, 0x09,
0x04, 0xD2, 0x6C, 0x20, 0x01,
0xFF, 0xF2, 0x6D, 0x22, 0x01,
0x04, 0x70, 0x00, 0xE2, 0x23,
0xFF, 0xF2, 0x72, 0x40, 0x01,
0x04, 0xD1, 0x08, 0x20, 0x09,
0x04, 0xD2, 0x6E, 0x20, 0x01,
0xFF, 0xF2, 0x6F, 0x22, 0x01,
0x04, 0x70, 0x00, 0xE2, 0x23,
0x00, 0x00, 0x00, 0x00, 0x01,
0x04, 0x70, 0x00, 0xC0, 0x01,
0x04, 0xD7, 0xFF, 0x20, 0x41,
0xFF, 0xF1, 0x07, 0x22, 0x01,
0x04, 0x70, 0x00, 0xE2, 0x01,
0x04, 0x77, 0xFF, 0x20, 0x01,
0x04, 0x70, 0x00, 0xE2, 0x01,
0x04, 0x70, 0x00, 0xC0, 0x01,
0x04, 0x70, 0x00, 0xC0, 0x01,
0x04, 0xD7, 0xFF, 0x20, 0x41,
0xFF, 0xF1, 0x07, 0x22, 0x01,
0x04, 0x70, 0x00, 0xE2, 0x01,
0x04, 0x77, 0xFF, 0x20, 0x01,

0x04, 0x70, 0x00, 0xE2, 0x01,
0x04, 0x70, 0x00, 0xC0, 0x01,
0x00, 0x02, 0x73, 0xA0, 0x01,
0xFF, 0xE7, 0xFF, 0x20, 0x01,
0x04, 0x98, 0x00, 0xE2, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x04, 0x99, 0x08, 0x40, 0x01,
0xFF, 0xF1, 0x08, 0x20, 0x09,
0x00, 0x00, 0x00, 0x00, 0x01,
0x04, 0x98, 0x00, 0xE2, 0x23,
0x00, 0x00, 0x00, 0x00, 0x01,
0x04, 0x99, 0x08, 0x40, 0x01,
0x04, 0x91, 0x08, 0x20, 0x09,
0x04, 0x9A, 0x75, 0x20, 0x01,
0x04, 0x92, 0x75, 0x22, 0x41,
0x04, 0x91, 0x08, 0x22, 0x01,
0x04, 0x98, 0x00, 0xE2, 0x23,
0x00, 0x00, 0x00, 0x00, 0x01,
0x04, 0x99, 0x08, 0x20, 0x01,
0x04, 0x88, 0x00, 0xE2, 0x01,
0xFF, 0xF2, 0x76, 0x22, 0x49,
0xFF, 0xF1, 0x08, 0x20, 0x01,
0xFF, 0xE9, 0x08, 0x20, 0x25,
0x04, 0x80, 0x00, 0xE2, 0x01,
0x04, 0x98, 0x00, 0xC0, 0x01,
0x04, 0x67, 0xFF, 0x20, 0x01,
0x04, 0x78, 0x00, 0xE2, 0x01,
0x04, 0x69, 0x08, 0x20, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x04, 0xC8, 0x00, 0xF0, 0x01,
0xFF, 0xE9, 0x08, 0x22, 0x01,
0x04, 0xC0, 0x00, 0xF2, 0x01,
0x05, 0x11, 0x08, 0x20, 0x01,
0x05, 0x12, 0x83, 0x22, 0x41,
0x04, 0xC2, 0x83, 0x22, 0x01,
0x05, 0x01, 0x08, 0x34, 0x01,
0x05, 0x02, 0x83, 0x22, 0x41,
0x04, 0xCA, 0x83, 0x22, 0x01,
0x00, 0x00, 0x00, 0x00, 0x01,
0x05, 0x08, 0x00, 0xE2, 0x01,
0x05, 0x18, 0x00, 0xF2, 0x01,
0x04, 0xD0, 0x00, 0xE2, 0x01,
0xFF, 0xF2, 0x7F, 0x40, 0x01,
0x04, 0xD1, 0x08, 0x20, 0x09,
0x04, 0xD2, 0x77, 0x20, 0x01,
0xFF, 0xF2, 0x78, 0x22, 0x01,
0x04, 0x70, 0x00, 0xE2, 0x01,
0x04, 0xD2, 0x79, 0x20, 0x01,
0xFF, 0xF2, 0x7A, 0x22, 0x01,
0x04, 0x70, 0x00, 0xE2, 0x23,
0xFF, 0xF2, 0x80, 0x40, 0x01,
0x04, 0xD1, 0x08, 0x20, 0x09,
0x04, 0xD2, 0x7B, 0x20, 0x01,
0xFF, 0xF2, 0x7C, 0x22, 0x01,
0x04, 0x70, 0x00, 0xE2, 0x23,
0xFF, 0xF2, 0x81, 0x40, 0x01,
0x04, 0xD1, 0x08, 0x20, 0x09,
0x04, 0xD2, 0x7D, 0x20, 0x01,
0xFF, 0xF2, 0x7E, 0x22, 0x01,
0x04, 0x70, 0x00, 0xE2, 0x23,
0x00, 0x00, 0x00, 0x00, 0x01,

0x00, 0x00, 0x00, 0x00,
0x00, 0x80, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x80, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x80, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x80, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x80, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x80, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x80, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
0x00, 0x80, 0x00, 0x00,
0x00, 0x07, 0x09, 0x79,
0x00, 0x0E, 0x12, 0xF1,
0x00, 0x9E, 0x36, 0x9F,
0x00, 0x07, 0x09, 0x79,
0x0F, 0xC5, 0xA3, 0x7F,
0x00, 0x07, 0x09, 0x79,
0x00, 0x0E, 0x12, 0xF1,
0x00, 0x9E, 0x36, 0x9F,
0x00, 0x07, 0x09, 0x79,
0x0F, 0xC5, 0xA3, 0x7F,
0x00, 0x56, 0x24, 0xC8,
0x0F, 0x53, 0xB6, 0x70,
0x00, 0x9E, 0x36, 0x9F,
0x00, 0x56, 0x24, 0xC8,
0x0F, 0xC5, 0xA3, 0x7F,
0x00, 0x56, 0x24, 0xC8,
0x0F, 0x53, 0xB6, 0x70,
0x00, 0x9E, 0x36, 0x9F,
0x00, 0x56, 0x24, 0xC8,
0x0F, 0xC5, 0xA3, 0x7F,
0x00, 0x80, 0x00, 0x00,
0x00, 0x07, 0x09, 0x79,
0x00, 0x0E, 0x12, 0xF1,
0x00, 0x9E, 0x36, 0x9F,
0x00, 0x07, 0x09, 0x79,
0x0F, 0xC5, 0xA3, 0x7F,
0x00, 0x07, 0x09, 0x79,
0x00, 0x0E, 0x12, 0xF1,
0x00, 0x9E, 0x36, 0x9F,
0x00, 0x07, 0x09, 0x79,
0x0F, 0xC5, 0xA3, 0x7F,
0x00, 0x56, 0x24, 0xC8,
0x0F, 0x53, 0xB6, 0x70,
0x00, 0x9E, 0x36, 0x9F,

of 4 bytes

```
// Set the hardware registers controlling GPIO, clock polarity, memory type, serial input type, power core configuration and more  
SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_COREREGISTER_IC_1_ADDR , R3_HWCONFIGURATION_IC_1_SIZE,  
R3_HWCONFIGURATION_IC_1_Default_loc );//24 bytes  
// Set 0x1D to the core control register. Enables CR, ADCs, DACs, allowing audio to pass through. 2 LSB bits control sampling rate of  
device (in this case 01 = 96 kHz  
SIGMA_WRITE_REGISTER_BLOCK( DEVICE_ADDR_IC_1, REG_COREREGISTER_IC_1_ADDR, REG_COREREGISTER_IC_1_BYTE,  
R4_COREREGISTER_IC_1_Default_loc );// 2 bytes  
}
```

#endif

Appendix C: Market Analysis

Commercial alternatives

Within the DSP Amplifier category there are numerous stereo amplifiers in the pro, and prosumer categories offered by companies including: K-array, Behringer, and Crown Audio (Harman International). Other systems dedicated to installed audio, contractor or 70V systems, and touring sound line arrays as well as those previously mentioned above are ill-suited for the average consumer looking to better their home audio experience.

Primarily most of these systems come in rack mount units and are designed to be robust and functional above aesthetics. An additional grievance is that most of these systems are two-channel high power units and that need 3 separate units to power the system. The more power needed, the more expensive the system.

The approach that many enthusiasts take is to use dsp software: Room Equalizer Wizard, or one that is provided with their hardware, in combination with a calibrated microphone and an enclosed DSP unit such as a miniDSP to characterize and correct the frequency response in both time and frequency. This is significantly more cost effective than the most affordable commercially available rackmount solution: the Behringer DCX2496, however, considerable wiring is required to interface with multiple amplifiers and the output level may not match the input sensitivity of the amplifiers being driven.

There are numerous DSP chips and development kits available through Texas Instruments utilizing both Purepath (a studio graphical development environment) and C-based programmability options sold as prototype board/chip level devices. Setting up one of these systems requires a fair amount of audio technician experience. Many powered loudspeaker manufacturers have opted to embed similar DSP boards (such as Equator Audio, JBL Pro) in their loudspeakers but none of these solutions are available to the public. The key missing link in the market is the integration of these chips into consumer amplifiers for custom rather than built in use.

This project will incorporate one or more dsp chip(s) with a developer written graphical interface (like the miniDSP) that can be used to accelerate initial setup and provide the computational power for all of the filters and speaker control.

Definition of Market Region:

The project's main competitors are other rivaling single unit 2.1 computer programmable speaker systems. The customers that this project caters to are primarily audio hobbyists looking to obtain a hi-fidelity stereo system with an emphasis on user friendliness and simplicity. By combining the key components required to process and amplify a 2.1 system, our product will streamline the 2.1 audio experience for the user and provide maximum cost savings over competing multi-component systems. The cost of this system combined with the interface focus on ease of use make this DSP amplifier system much more appealing to cost conscious and computer inexperienced audio hobbyists alike compared to more advanced audio dsp systems.

Notable Audio Industry Competitors:



Dayton Audio is a major provider of in the home and commercial speakers, drivers, and speaker building accessories. Selling rivaling products such as subwoofer plate amplifiers and 2.1 plate amplifiers, this company is a serious contender in the audio industry.



Yung International is an international audio sourcing and manufacturing company based in China/Taiwan. For the last twenty five years, they have provided quality audio products, most notably their subwoofer amplifiers ranging from 100-500 Watts.



Behringer is an international company that provides quality, affordable, audio equipment. The equipment they sell caters specifically to the music industry, with some emphasis on home audio system. A notable rivaling product they offer are audio amplifiers with DSP.



Peavey is one of the largest audio equipment manufacturers nationwide and is located in Meridian, Mississippi. Since 1964, they have pioneered amplifier technology in the audio industry. Most notably, they have pioneered power amplifier technology by reducing weight and increasing output power, for affordable cost.



JBL Audio by HARMAN is a company that provides every form of audio product under the sun. Established in the late 1920's, this company has led the charge from the beginning of the electronic audio industry. Their most notable competing products to our Audio-DSP amplifier are their home theatre systems.



M-Audio is a provider of two channel USB audio interface solutions that compete with our audio DSP unit. While not as large as other competing companies, their products are unique and provide quality audio tailored to recording professionals.

MiniDSP is provider of Digital Signal Processing Platforms for audio applications. They also make single and dual channel plate amplifiers.

Size of Market:

The total market for consumer electronics is over 209 billion USD [13] (according to 2013 study by CEA). Focusing on audio electronics Factoring in inflation [14], the market in 2013 for home audio products hovers around 1.7 billion USD [15]. Due to its size, there is a significant ability to profit in this region of the market.

Is Part of this Market Addressable to our Product:

This solution is priced affordably, matching competitor's pricing at around \$500-1000. There are multiple combinations of individual components needed to create such a 2.1 speaker system that are currently being sold (see list of Competitors above). Our combined Audio-DSP amplifier will fit in right with the competing products while offering advantages they lack.

Key Strengths to Leverage that our Competitors Cannot:

The project plans to combine features of several other competitor products (most notably the plate amplifier and a DSP unit) into one unit that runs on higher power speakers than currently available. The application for this product in the audio market is extremely relevant; the Audio-DSP amplifier will take the best of the 2.1 amplifier and the DSP unit to provide quality audio to high power speakers.

What Areas of the Market are Not Well Served:

The main strength of our project also coincides with a portion of the market that is extremely underserved at the moment. There are no other competing solutions that offer all in one computer programmable audio-DSP 2.1 speaker system management. Another feature that is lacking in this market is the ability to provide computer controlled audio-dsp to high power speakers. Most setups are only for computer speaker systems or theatre systems with low power requirements. A digital 2.1 speaker system that could operate on high power speakers would have a unique feature that is unmatched by competing product solutions.

Window of Opportunity:

The biggest window of opportunity in this market is to create a single unit that can provide high quality audio with DSP capabilities to high power speakers. Many various components can be purchased and arduously assembled and tweaked to provide quality audio for a 2.1 speaker system, but there are no available all-in-one systems that cover the entire spectrum. Providing a product that has easy to use digital interfacing is the key to surpassing competing affordable 2.1 systems in the audio market.

How big of an effort would it take to enter the market?

The estimated cost of developing this Audio-DSP Amplifier and entering the market are shown in Table 1.1 below. The bulk of the cost of entering the market comes from marketing the product to potential customers. Smart times to demo the product would coalign with convention dates such as NAMM, Jan 22-25 in Anaheim, the annual AES convention, the NY Audio Show or the Audio Karma Festival to maximize product exposure to potential customers.

Table C.1: Audio-DSP Amplifier Development Costs

Purchase	Cost (\$)	Assumptions & Reasoning
Retail Space	4500/ month	Average between 2000-7000/month in SLO[11]
Engineers	70,000	Assuming team of 2 with 70,000 annual salary per (6 months) [12]
Hiring Process	8,000 total	Includes phone interviews, current employee time spent, and travel expenses, etc.

Prototype	400	Average estimated for electronic components
Testing	2,000	This value comes from 5 times the parts cost of a prototype
Marketing + Demos	198,000	Marketing team of 3 who also demo the product, assuming for 6 months of work at \$11,000 per month (individually).
Miscellaneous	5,000	Failed designs, reworks, meeting time, etc.
Total	298,600	Assuming 6 months of retail

Who are the Key partners to Engage to be Successful?

Texas Instruments is one of the largest semiconductor device design and manufacturing companies in the world. With analog ICs and embedded processors for many electronic applications, most notably audio for this project.

Texas Instruments will be the largest provider of necessary (and affordable) ICs as well as the microcontroller for the project design.

Analog Devices is one of the biggest competitors to Texas Instruments and has a very wide range of audio DSP chips that are highly used in industry and very customizable, working with one of these two giants is key for making a competitive and cost effective end unit.

Parts Express is a provider for audio, video, and speaker components. The power supply unit/power amplifier will be purchased from them.

Who would be key potential customers we would need to contact?

Best Buy is a general purpose provider for a variety of affordable consumer electronics. Since the product price is around \$400 and customer archetype favors new audio-hobbyists, Best Buy is a fantastic vendor for the Audio-DSP amplifier.

Parts Express is another general purpose provider of a wide range of consumer electronics, with a focus specifically through internet transactions. In similar scope to Best Buy, they offer affordable audio products that are in the Audio-DSP Amplifier's price category of around \$400. They would be an ideal vendor for this product.

Amazon is the largest online distributor of products in the world. Specifically for audio electronics, this company would be an effective vendor for our Audio-DSP amplifier, as they would be able to ship the unit to customers across the globe.

Project/Product Description

What is wrong with the present solutions?

All present solutions satisfy one portion of the home audio market but none succeed in terms of combining sufficient 3 or more channels of amplification, programmable DSP, and simplistic user interface at an affordable price. The most common approach when constructing a bi-amplified audio system with digital processing is to purchase a DSP unit, two stereo amplifiers (or receiver), and a powered subwoofer (or dedicated subwoofer amplifier). Other alternatives include receiver with external subwoofer amplifier systems or three channel plate amplifiers with built-in passive crossovers which lack DSP and provide inferior sound quality due to use of cheap passive crossovers.

What is the nature of your proposed solution?

The answer to the problems of simplicity and system price are combining components inside a single chassis and providing the most straightforward connectivity option: stereo RCA input and binding post speaker outputs. Control for the DSP will be implemented through a microcontroller which takes inputs such as filter selection and volume displayed through an LCD display and controlled with rotary encoder, buttons, and capacitive touch control. This cost effective and visually simplified interface will control the filter settings of the DSP unit which allows for stand-alone, fully configurable system.

Competing Product Solutions:

Table 1.2 on the next page showcases several noteworthy competing product solutions alongside their respective advantages and disadvantages to our proposed Audio-DSP Amplifier.

Table C.2: Competing Product Solutions Comparison

Image	Product Name and Manufacturer	Advantages	Disadvantages
	MiniDSP Kit 2 x In, 4 x Out	<ul style="list-style-type: none"> • Simplified input/output • User friendly USB interface 	<ul style="list-style-type: none"> • No amplification • Low output signal level • Only four output channels
	Dayton Audio SPA250 250 Watt Subwoofer Plate Amplifier	<ul style="list-style-type: none"> • Cost Effective • Easy to use • High Level Inputs for flexibility • Daisy chain capable 	<ul style="list-style-type: none"> • Plate Amplifier form factor • No Processing • Analog In Only
	iNUKE NU1000DSP 1000-Watt Power Amplifier with DSP Control and USB Interface [9]	<ul style="list-style-type: none"> • Cost Effective • Class Leading DSP • High Output Power • Lightweight 	<ul style="list-style-type: none"> • Limited I/O choice • Loud Fan • Only two amp channels, need more than one • Large • Poor high frequency response

	Peavey IPR2™ 2000 DSP Power Amplifier [10]	<ul style="list-style-type: none"> • Class Leading DSP • High Output Power • Flexible I/O • Input Sensitivity Selector 	<ul style="list-style-type: none"> • Rack Mount, large • Very Expensive • Only two amplifier channels
	Dayton Audio MCA2250E 2.1 Channel Class D Plate Amplifier [16]	<ul style="list-style-type: none"> • Cheapest reliable 2.1 Amplifier • Self Contained unit • Powers a complete system • Simple wiring 	<ul style="list-style-type: none"> • No DSP • Low Output Power • Plate mounting scheme limited • Limited flexibility • Attached wires limits

The goal of the project is to simplify the user experience with home/studio speaker systems and reduce cost, so removing superfluous features that inflate the cost of the system is of paramount importance. After comparing the most relevant features of competing solutions, the following marketing requirements are established below to remove features that are unimportant or provide unnecessary complication for home audio applications. See above table 1.2 for a description of competing products' strengths and weaknesses.

Alternative Design Solutions:

The DSP board is the single most expensive hardware component of the total system. The main issue with the design proposed in Figure 3.2 above lies with the ability to interface and control the DSP board with the microcontroller. Once this is achieved, the total system can successfully integrate and operate as a single unit. The following proposed design alternatives are heavily centered on choosing an appropriate DSP board and fleshing the system out to accommodate its specifications.

1) Use MiniDSP 2x8 Unit as DSP Board [6]:

Using a 2x8 MiniDSP unit, we gain access to MiniDSP's custom DSP software interfacing. This software has a well designed GUI capable of easily changing audio settings to meet desired specifications for varying home theatre environments. This software makes the learning curve of system interfacing much easier for the consumer. However, the MiniDSP unit is unable to interface with a microcontroller, and must use computer to USB interfacing to adjust settings on the board. This makes implementing DSP changes inside the unit extremely difficult, as we do not have access to the source

code capable of augmenting the MiniDSP unit to interface with a microcontroller. The MiniDSP unit is also expensive, pricing at an estimate of around \$300.

2) DSP Board Self Fabrication:

By fabricating the DSP board ourselves, we can replace an expensive DSP unit from a provider (such as MiniDSP) with a custom board design centered around an inexpensive Texas Instruments DSP chip. A custom tailored DSP board would certainly allow for microcontroller interfacing, having the necessary input output requirements (2x6), and all necessary processing capabilities. The manufacturing cost of a self fabricated prototype DSP board is around \$50, in comparison to the MiniDSP (2x8) at \$300. This cuts down prototyping costs drastically, and would be the optimal solution given enough time and/or additional assistance with this project. Unfortunately, the design and fabrication of a custom DSP board requires enough man-hours to be its own senior project.

3)Use Existing Analog DSP Board [19]:

If an existing Analog DSP Board is purchased from a provider, an ideal board that suits our system's I/O requirements can be obtained. A 2x4 DSP board capable of microcontroller interfacing can be purchased from a provider to meet system specifications. The learning curve for the custom DSP board software is steeper than the MiniDSP software, but can be adjusted through microcontroller interfacing without the use of a computer. The board cost is estimated at \$600. There is an additional cost of \$70 for a separate hardware component to connect the DSP board to the microcontroller (to be purchased from the board supplier).

Figures C.1 and C.2 below showcase visual representations of cost comparisons as well as ease of implementation comparison.

Figure C.1: DSP Board Cost Comparison

Figure C.2: Estimation of Design Method Ease of Implementation

*Estimation of ability to implement is based on a scale of 1(most difficult) to 10(easiest).

Although purchasing an existing analog board is the most expensive method, it has the most reliable chance of an effective system implementation for a prototype. If this system was designed by a company, the fabrication of the DSP board would be completed in-house. Since the goal of this project is to produce a functioning prototype under a limited timeframe, the third design choice is the most desirable method.