DESIGN DOCUMENT

EE 491 | DEC 1503

ABSTRACT

A technical outline of the modular audio mixer for our client, Jay Becker, with the help of our advisor, Josh Bertram. This document covers the System Design, the Detailed Design, and the Design Documents.

Team Leader: Clayton Hawken
Team Concept Holder: Charles Stobbie
Team Webmaster: Brian West
Team Communication Leader: Deborah Baeder

December 8th, 2015

Table of Contents

INTRODUCTION

Problem Statement

Our client, Jay Becker, has many different audio sources that he listens to. He only has one set of speakers by his computer, and needs a way to connect all of these audio sources with the speakers. His current setup consists of a 3.5mm audio source going into his computer, and the combined output goes out through his sound card to the speakers, a "daisy chain" configuration. This is not ideal because it requires that the computer to be on in order to use the audio from the 3.5m.

Problem Solution and Deliverables

To meet the problem of multiple audio sources, we will design an audio mixer to combine and control individual inputs, including 3.5mm audio and Bluetooth. The hardware will consist of a PCB that will be divided into 3 main circuits. The mixer will be the circuit that mixes the audio signals together into a single output. The power supply will provide power to all the ICs on the PCB, as well as the raspberry pi. The relay driving circuit will provide power for energizing the relays. Since the raspberry pi GPIO is incapable of providing enough current, we need this auxiliary circuit. These circuits will be fabricated onto a printed circuit board, and then have specific components linked to the exterior. The mixer's interface has dials on the top of the enclosure that allow the user to adjust the volume of each channel, and then view the respective channel level on the LCD screen. A power toggle switch activates the board, as well as a single red LED above the toggle itself. Furthermore, our audio mixer will include a single C program file to handle user inputs and adjust the output accordingly.

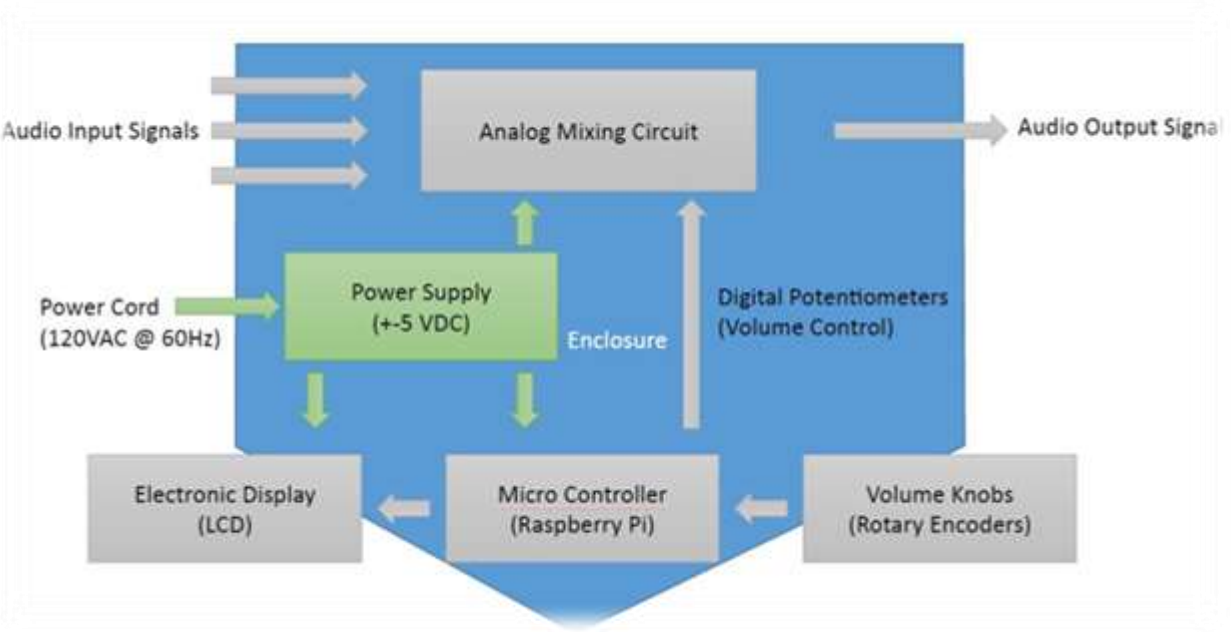
***May 2015 Deliverables**: Have one functioning, prototype mixer, with knowledge of issues and improvement ideas for the final build.*

***December 2015 Deliverables**: Fully-functioning prototype that meets all requirements (including Bluetooth accessible). Solving any problems that happened from the May prototype.*

SYSTEM LEVEL DESIGN

Functional Decomposition and Systems Analysis

Our mixer must be capable of mixing two analog 3.5 mm inputs and Bluetooth audio, and outputting the mixed signal to a single analog 3.5 mm analog output. The mixer must be able to handle a stereo signal. It also must be capable of driving headphones. Input and Output channels will need individual volume control and current volume information needs to be displayed to the user.



We chose a Raspberry Pi as the main controller for the digital circuitry. For volume knobs we chose Rotary Encoders (Position is not stored, Clockwise and Counterclockwise rotation is recorded digitally.) A PCB containing our power supply and analog mixing circuit is housed in our enclosure. The analog mixing is done with digitally controlled potentiometers so that the volume can be adjusted digitally from the Raspberry Pi. An electronic display is also controlled by the Raspberry Pi.

REVISED DESIGN SPECIFICATIONS

User Interface

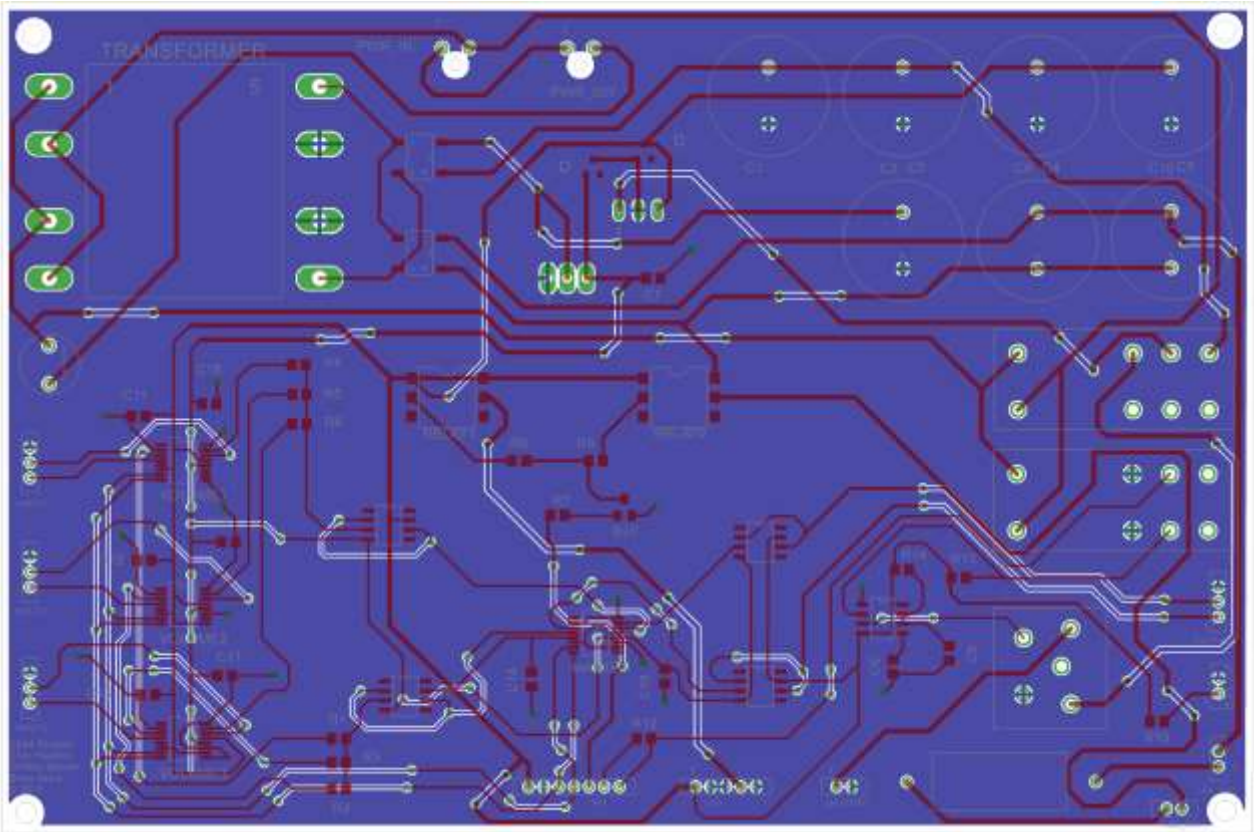
Multiple sources must be routed to a master channel. Two input channels are given an analog audio jack on the left side of the enclosure. The third input channel is wireless Bluetooth audio so there is no audio jack, however it still has a volume knob. One master volume dial can control all three input channels to one output channel. The LCD screen has visual indicators, as well as a numerical indicators for levels from 0 to 99%. The dials rotate both clockwise, and counterclockwise. This behavior will directly affect the amplitude of the analog sound signal, causing an increase or a decrease. The Raspberry Pi will store the latest level and remember upon shut down. The volume knobs also have a muting function from pushing down. The LCD screen will allow for four character lines of information. The dimensions of the screen are about 4x3 inches, and will have white text on a blue background. The screen will be placed center-top of the enclosure. The LCD screen will toggle off the backlight after no user input for a short duration. The power switch toggle is a simple up/down device that illuminates a small indicator LED on/off dependent upon power status.

Printed Circuit Board

To house the components for the mixer, we will create a printed circuit board. This board will combine the three analog circuits into a single assembly. The raspberry pi, however, will be on a separate board. The board will contain a mixture of through-hole and surface mount components. A wires in the mixer will be routed through the Printed Circuit Board (PCB). That is, the components mounted on the enclosure will have wires going to the PCB, and then there were be a PCB output interfacing with the raspberry pi. This will be much neater than having the wires go from the components directly to the raspberry pi.

We will design the board in a program called Cadsoft Eagle. This is a free program that will allow us a lot of freedom in designing the PCB. The main reason why we chose Eagle is because it has an extensive

user community, with many tutorials. Since none of the members of our group has any experience in PCB design, we will need to rely heavily on internet tutorials in order to design our PCB. Once we have our PCB designed, we will pick a manufacturer in order to fabricate it. We will consult with Lee, the electronics technician, in order to determine which vendor will best suit our needs. The PCB print is included below.



The schematics for each section are included in Appendix.

Mixer

This circuit will mix the audio signals together. It employs digital potentiometers to attenuate the input signals. There are digital potentiometers in total: one for each input channel and one for the master volume. Each potentiometer IC is actually a double potentiometer, because the mixer uses 2 channel stereo audio. The values of the potentiometer will be dictated by the raspberry pi. The potentiometers

will communicate via I2C with the raspberry pi. The circuit contains two voltage buffers to ensure that there is no signal loss between the mixing stage and the master volume stage. There are two additional voltage buffers connected to the output so that the mixer is capable of driving low impedance loads, such as headphones. Each voltage buffer consists of a simple 8 pin operational amplifier IC with the output connected to the inverting input. A schematic of the circuit is shown below.

Circuit Schematics are located within the Appendix on page 24 and on:

Safe Shutdown on page 24

Power Supply on page 25

Mixing on page 26

Relay Drivers on page 27

External Interfaces on page 28

Power Supply

Since we are creating an active mixer, there will be several components that need a power supply in order to operate. As such, the PCB will contain an on-board power supply that will convert 120VAC mains electricity into low voltage DC that will work with the active components. The raspberry pi requires a stable 5V power supply to operate. The LCD, rotary encoders, and digital potentiometers also operate on 5V. The op amp ICs require both a positive and negative supply rail, but the exact value of voltage is up to the user. It makes the most sense to use 5V for the positive rail on the op amps, since all other components also use 5V. Thus, we will use -5V for the negative rail for symmetry purposes. This will provide more than enough headroom for the input signal, to make sure that there is no saturation. The input from any 3.5 mm source should be no larger than 1.5V or so.

The power supply consists of a transformer, bridge rectifier, and filter capacitors. There are also a linear voltage regulator to make sure that the negative voltage stays constant. The positive 5v uses a Buck converter instead of a linear regulator due to the power draw of the Raspberry Pi and back lighting.

Since the mixer contains a microcontroller and other digital components, the supply will need to be very consistent. The supply also contains a pair of reverse biased diodes to protect the regulator ICs in case of short circuit. There is a red indicator LED mounted on the enclosure to indicate that the mixer is powered on.

Relay Drivers

The 3.3V GPIO pins on the raspberry pi are incapable of providing enough current to energize a solid state relay. Therefore, we will need to create a circuit that will switch the relay on and off. The circuit will need one relay for the LCD and backlighting. We will use a Raspberry Pi GPIO pin in order to switch the relay. The circuit consists of a bipolar transistor biased in saturation to act as a switch, thus switching on and off the energizing current through the input of the solid state relays. The solid state relays use an LED in the input to energize the switching, so the circuit also contains some current limiting resistors to ensure that no more than 15 mA runs through the inputs of the relays.

Back-up Battery

The backup battery circuit switches the Raspberry Pi to battery power whenever the mains power has been removed. This is necessary because the Raspberry Pi requires a safe shutdown. The circuit outputs a logical high when mains power is connected, and low when it is not. The Raspberry Pi uses this to trigger a shutdown event. Once the circuit switches to backup power, the 555 timer will switch off the battery power after 15 seconds.

Additional Pin Map

The following images outline the intended usage per Raspberry Pi B+ Pin Mapping for the digital potentiometers, the rotary encoders, the LCD screen, and the Raspberry Pi itself. Many of these connections were not included in the PCB as they were component to component.

Raspberry Pi and LCD on page 29

Digital Potentiometer on page 30

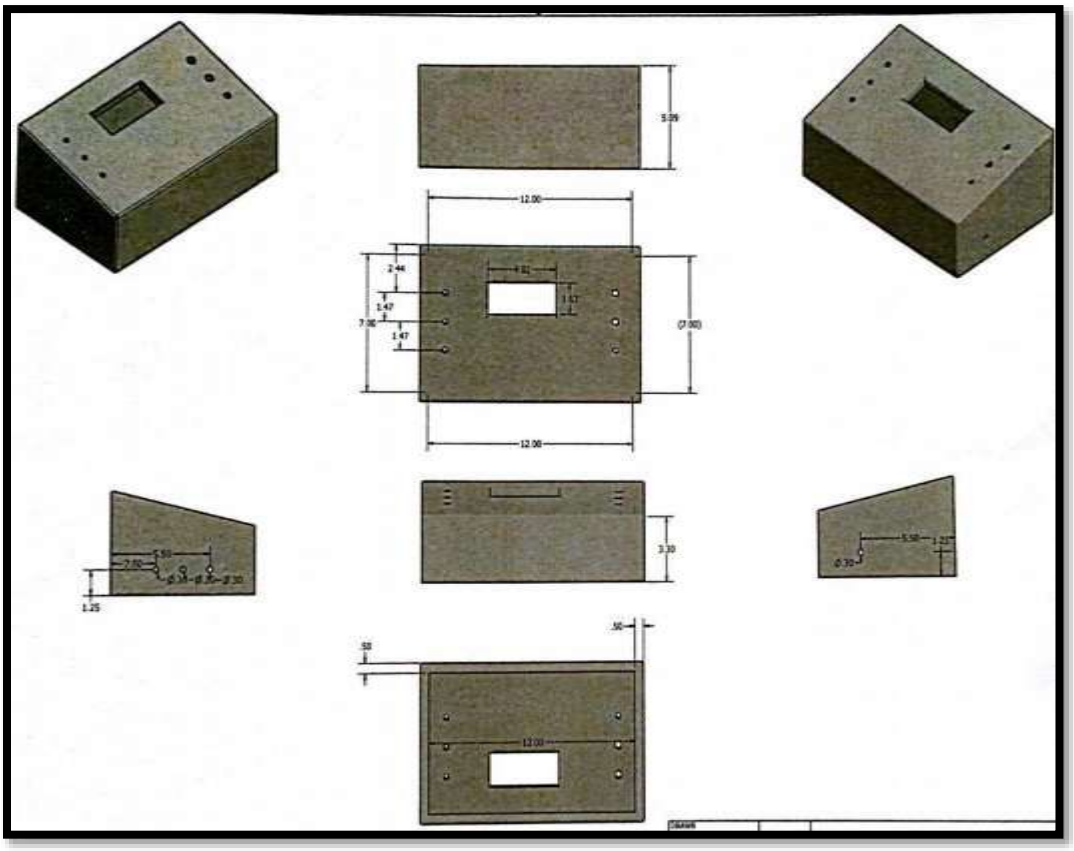
Rotary Encoder on page 31

Mechanical - Enclosure

The following images convey the measurements and specifications to build our first enclosure. The enclosure was drawn using Autodesk Inventor. This enclosure required the ability to hold auxiliary jacks, rotary encoders, an LCD, a power LED, a power switch, and provide an outlet for the power cable. After fabricating the first prototypes' enclosure, our team was able to construct and fit most components, with the exception of the power LED, and the power cable port. The LCD screen was also slightly loose within the top face of the enclosure.

Over time, our team found that the enclosure did not need to be so wide, and resulted in roughly the following dimensions: 5" x 7" x 8" (H x W x L).

Autodesk Inventor Dimension Views - Version 1.0

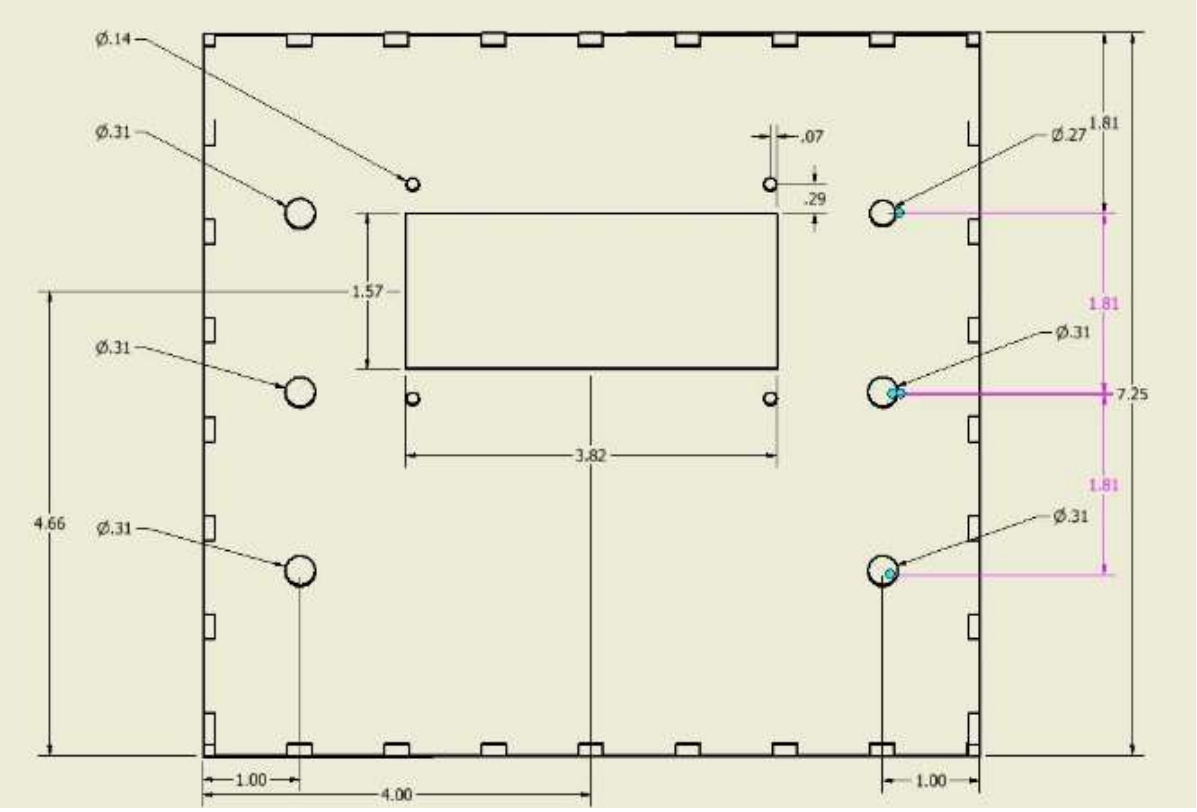


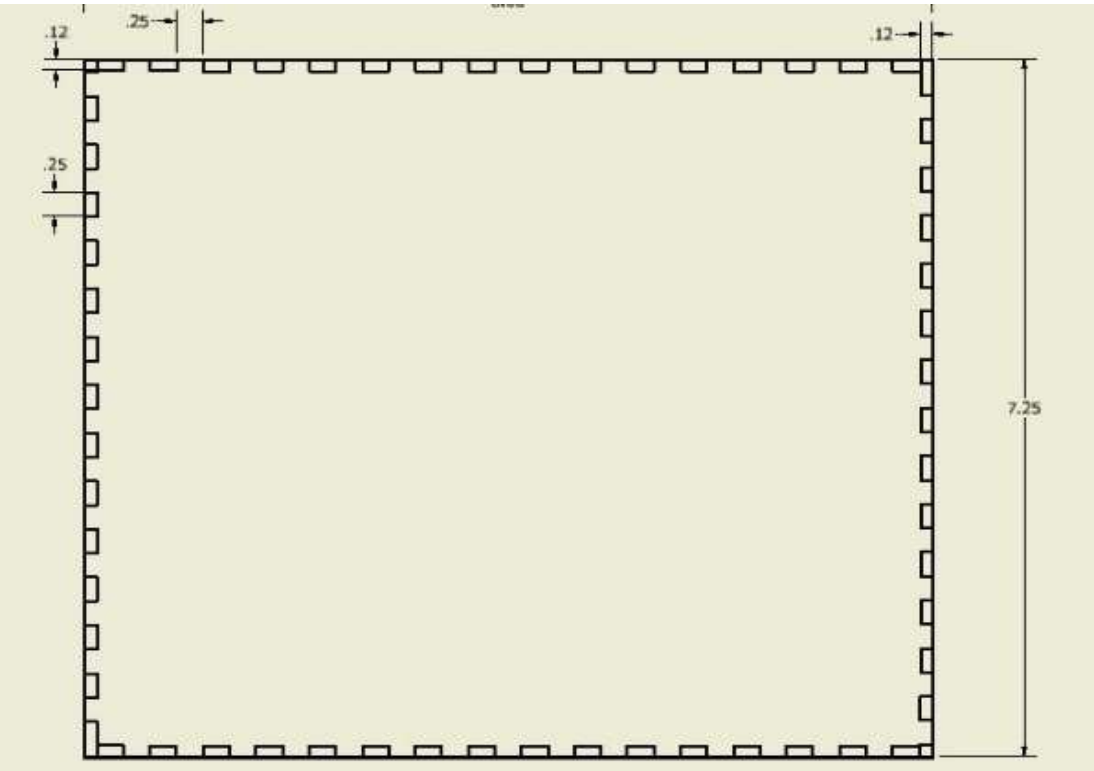
Autodesk Inventor Dimension Views - Version 2.0

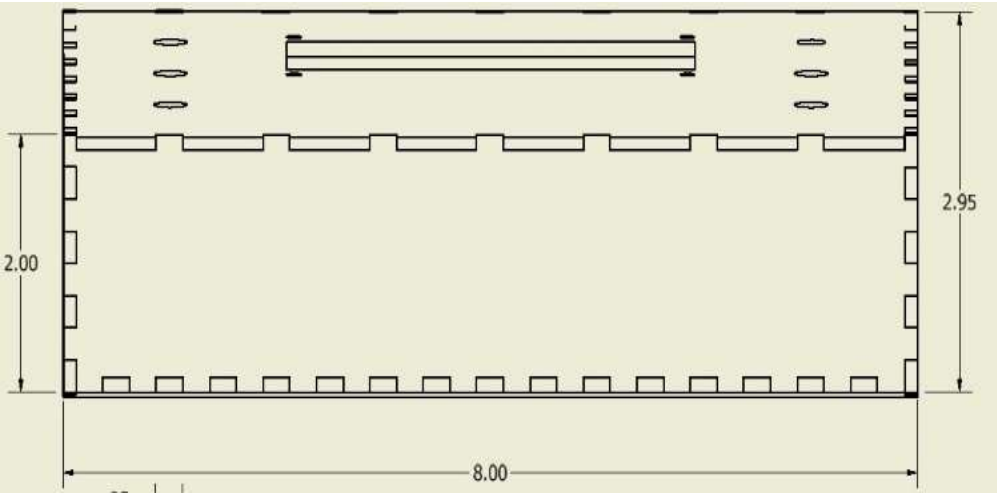
This version took into account removal of one input auxiliary port, a backlighting switch, power jack, and LCD screws for stability. The components' fit do not have any error except for the power jack, where the height measurement was misunderstood as too large.

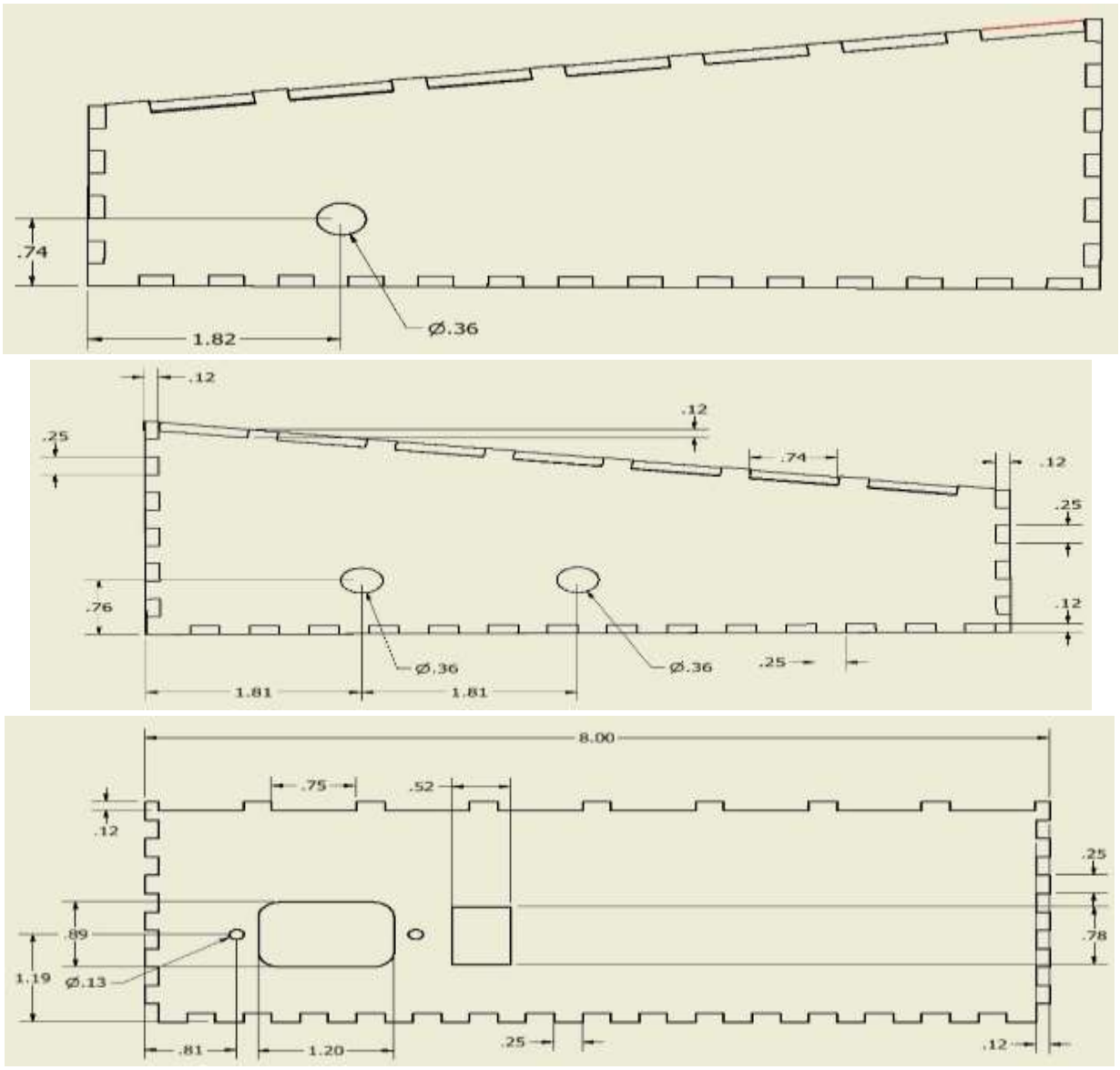
All components otherwise fit the enclosure and provide user-friendly controls for the mixer. Once again, our team found that the enclosure did not need to be so tall, and resulted in roughly these following dimensions: 3" x 7.25" x 8" (H x W x L).

Views: Top Face, Bottom Face, Front View, Right Face, Left Face, Back Face.




.12
.25
.25
.12
7.25


2.00
2.95
8.00



Mechanical – Enclosure Fabrication

The enclosure was fabricated at the College of Design, using a laser printer on 0.118” acrylic material. The 12” x 12” acrylic panels were shipped from Amazon.com at a cost of $5 dollars each. By placing the Inventor drawings of three faces each on two panels, our team was able to accomplish a total manufacturing cost of roughly $25 dollars. Using a handful of super glue and screws, the components were able to fit snug within the designed enclosure.

Other options had been reaching out to Protocase.com or making a 3D-printed enclosure from a lab within Howe Hall on the Iowa State University campus. Protocase.com would have delivered a professional galvanized steel or plastic case, as prescribed by an Inventor drawing as well, but would have increased the overall production cost. Similarly, having an enclosure printed on a 3D-printer would have been conveniently located on campus, but much too expensive for the size of the enclosure our project required. Finally, after training on the laser printer, the enclosure was produced within one hour on campus.

This translucent, blue acrylic material is lightweight, aesthetically appealing, and allows for a soft glow of light given from the inside of the enclosure. This light comes from a small edge-lit strip of LEDs adhered to the inner perimeter of the enclosure. This lighting addition was added for aesthetic appeal, at no extra cost to the team besides partial power draw.

Software – Overview

The use of the Raspberry Pi allows us more functionality in our project so that we can continue to add different inputs and features to our mixer without much change to the project overall. Since the Raspberry Pi supports a full Linux operating system, the programming for this project must be different compared to a microcontroller with no operating system. Running our file on startup, our program will need to be run whenever the power is cycled on the mixer. This is more difficult than a microcontroller since our program must be run when the operating system is booting up. The figure below demonstrates the program's logic flow of our usage of interrupts and loops.

Programming Concept on page 32

CHALLENGES

- Raspberry pi Polling/Interrupt response time may be inadequate for smooth user input. Additional Programing was required in the Linux kernel to ensure that no input was missed.

- Raspberry PI did not shut down correctly when powered off. Battery Backup circuit was required for proper state-saving in the Raspberry Pi in the event of an unexpected power off.
- 256 Linear step potentiometers only provide 15 3dB steps, more accurate for human hearing.
- Large number of wires required wiring system. Molex connectors were used for component connections not included in the PCB.
- Cross talk between digital and analog lines in PCB provided routing challenges, especially with space requirements in our small mixer enclosure.
- Our team's first PCB design in semester one, unforeseen spacing, assembly and routing problems occurred. Most were correct in our second build for December

TESTING

Our primary test to ensure that the prototype audio mixer is valid will be the frequency response. Using software and equipment provided by Iowa State we set up an efficient logarithmic sweep of the audible frequencies. Our only benchmark for acceptable signal loss is comparing with production datasheets, or going by our own hearing. But generally we want a linear, minimal delayed response, with constant amplitude at all frequencies in the audible range. The following demonstrate possible testing situations:

- Applying various sinewave to input to measure sinewave amplitude output for each input for audible frequencies around 50 Hz to 20 kHz, while ensuring a flat response curve.
- Testing for noise of sample audio files directly through the speakers vs through mixer to speakers.
- Ensuring that power cycling the device does not cause the programs to crash and lose functionality.

APPENDIX I: CODE

Code Explanation

Included below are some snippets of code that will explain the various software functions of our audio mixer. This is the most significant code of the device since it handles the user input.

Rotary Encoder Interrupt

```
void encoderInterruptCH2(void)
{
      if(push2 == 0)
      {
            digitalWrite(LCDDIM, HIGH);
            int mostSignificant = digitalRead(CH2N1);
            int leastSignificant = digitalRead(CH2N2);

            current2 = (mostSignificant << 1) | leastSignificant;
            int sum = (last2 << 2) | current2;
            if(sum == 0b1101 || sum == 0b0100 || sum == 0b0010 || sum ==
0b1011)
                  value2++;
            if(sum == 0b1110 || sum == 0b0111 || sum == 0b0001 || sum ==
0b1000)
                  value2--;
            last2 = current2;
      }
      timeFlag = 1;
}
```

Each rotary encoder is linked to an interrupt function that is called whenever a user adjusts the knob. This function, shown above for channel two input, calculates whether the knob has been rotated clockwise or counterclockwise using bitwise operations. We can then use current and last volume variables to determine if we need to adjust the digital potentiometers as the user desires. This can only

be done if the channel has not been muted, thus the reason for a preliminary if statement. We also want to make sure the LCD screen is lit up if the user is interacting with the mixer, hence writing the LCD dim pin to high. Finally we set a flag to indicate user input so the time can be recorded later in the code to test if the user has stopped interacting with the mixer.

Push Button Interrupt

```
void pushButtonInterruptMaster(void)
{
	digitalWrite(LCDDIM, HIGH);
	if(pushM)
	{
		pushM = 0;
		newVolumeM = volAtMuteM;
	}
	else
	{
		pushM = 1;
		volAtMuteM = newVolumeM;
		newVolumeM = 0;
	}
	timeFlag = 1;
}
```

Must like the rotary encoder interrupt, we have an interrupt for if the user presses down on the knob indicating they want to mute that channel. Once again we want to make sure the LCD screen is lit up when the user is interacting with the device. We then simply need to check if the user is muting the channel, or unmuting it, hence the simple if-else case. Finally we set the flag high again for capturing the time of user input.

Processing volume changes

```
void processVolumeChanges(int channelNumber){

	//Temporary variables
```

```
	int i;
	char volumeToSend = 255;
	char buf2[2]; //For the volume numbers.

		if(channelNumber == 1){

			pastVolume1 = newVolume1;
			newVolume1 = value1 / 4;
			//If the user turns the volume up at full volume, then no
changes
			if(newVolume1 > 33)
			{
				newVolume1 = 33;
				value1 = 132;
			}
			//If the user turns the volume down at no volume, then no
changes
			if(newVolume1 < 0)
			{
				newVolume1 = 0;
				value1 = 0;
			}
			lastValue1 = value1;
			lcdPosition(lcdHandle, 5, 1);
			//if the volume is a single digit, shows the leading zero
along with the singles digit
			if(newVolume1 < 4)
			{
				sprintf(buf2, "0%d", newVolume1 * 3);
				lcdPuts(lcdHandle, buf2);
			}
			//prints all the double digit volume values
			else
```

```
        {
            sprintf(buf2, "%d", newVolume1 * 3);
            lcdPuts(lcdHandle, buf2);
        }
        //Sends the new volume to both registers in dig. pot.
        //Simply have to just send the value now
        char temp[] = {VOLR1,linearArray[newVolume1]};
        char temp2[] = {VOLR2,linearArray[newVolume1]};
        write(CH1BYTE, temp, 2);
        write(CH1BYTE, temp2, 2);
    }
```

Whenever the user makes a change with the volume knob, the interrupt is first called to process quickly what action they are making. Soon after a user defined function is called which is shown above. This is done because interrupts must execute quickly so there is a less chance of a step of the rotary encoder being missed. This function takes in the channel number that is being changed and properly adjusts the digital potentiometer for that channel and also the LCD screen. We have two cases where we must not rely on the user turning the knob properly. One is if the user keeps turning the volume down when the volume is already at zero. We want to make sure that our variables do not go into the negatives as negative volumes do not exist. The same occurs for max volume as we can only output a maximum volume so the user must not be able to obtain any higher volumes. We then adjust the volume number on the screen for the respective channel and adjust the digital potentiometer for that channel by writing to it over I2C.

Infinite loop and timing

```
/** Infinite loop to run our code. All setup should be complete. **/
    for (;;)
    {
        //Shut down right away if the wall power has been removed
        if(wallPowerRemovedFlag == 1)
        {
            system("sudo shutdown -h now");
```

```
}

if(timeFlag == 1)
{
      timeOld = time(0);
      tOld = localtime(&timeOld);
      timeFlag = 0;
}

tim = time(0);
t = localtime(&tim);

//Find out if the user has been inactive for 10 seconds
if(t->tm_sec <= tOld->tm_sec)
{
      int seconds = tOld->tm_sec - t->tm_sec;
      //If so then dim the LCD screen
      if(seconds >= 10)
      {
            digitalWrite(LCDDIM, LOW);
      }
}
else
{
      int seconds = t->tm_sec - tOld->tm_sec;
      //If so then dim the LCD screen
      if(seconds >= 10)
      {
            digitalWrite(LCDDIM, LOW);
      }
}
```

At the beginning of this code is the point in our program where the code returns to and never exits. This allows the user to have a functioning device whenever they desire to use it, as it will always be

ready for user input. Next we have a flag variable that is triggered by an interrupt that tells us whether the device has been removed from a stable power supply and to our backup battery supply. If it has been switched to our backup battery then the device needs to instantly shutdown to ensure that a clean reboot will occur when the device is plugged back into a stable source. Next is the code that occurs whenever a user interacts with the device as was discussed before. The time is taken at that exact moment so we know when the last moment of user input has occurred. Next the current time is taken so we can continually compare it to the last input time, and if 10 seconds have expired then the LCD screen will go dim.

Volume adjustments

```
/**************************************************************************
             * If volume M control has been turned successfully in either
direction *

**************************************************************************/
            if((valueM - lastValueM >= 4 || valueM - lastValueM <= -4) && pushM
== 0)
            {
                  processVolumeChanges(4);
            }


            //if the encoder has been pushed, output a double zero the screen
and mute
            if(push1)
            {
                  lcdPosition(lcdHandle, 5, 1);
                  lcdPuts(lcdHandle, "00");
                  //Tell dig. pot. to "shutdown" both registers (will store
value)
                  write(CH1BYTE, shutdownR1, 2);
                  write(CH1BYTE, shutdownR2, 2);
```

```
            }
            else
            {
                  //Tell dig. pot. to "turn on" both registers (will store
value)
                  write(CH1BYTE, turnOnR1, 2);
                  write(CH1BYTE, turnOnR2, 2);
                  unmuteThenPrintOld(1);
            }
```

This code is what executes after the interrupt has been processed. As you can see, the first if statement determines if the master volume knob has been turned successfully and if so, calls the process function we defined above. The second part of the code, the if-else statements, determine if the user has pushed the knob to mute or unmute and performs the proper operation of the digital potentiometers and adjusts the LCD screen as necessary. The unmute function is a simple function that restores the value of the channel that was once muted onto the LCD screen.

Startup script

Important to the device is a startup script that executes when the device is powered on. This script allows the Raspberry Pi to boot up, but then execute our C program as soon as possible so the user can begin interfacing with the device.

APPENDIX II: USER GUIDE

1. Connect the A/C power cord to the jack in the back left corner of the mixer.
2. Switch the toggle on the top face in the upwards direction. A small LED light should turn on. If it does not, there is a problem with the power circuit.
3. Plug in one to two 3.5 mm auxiliary cords to the left-hand side of the mixer, or connect your Bluetooth-enabled device. This works for both Apple and Android.

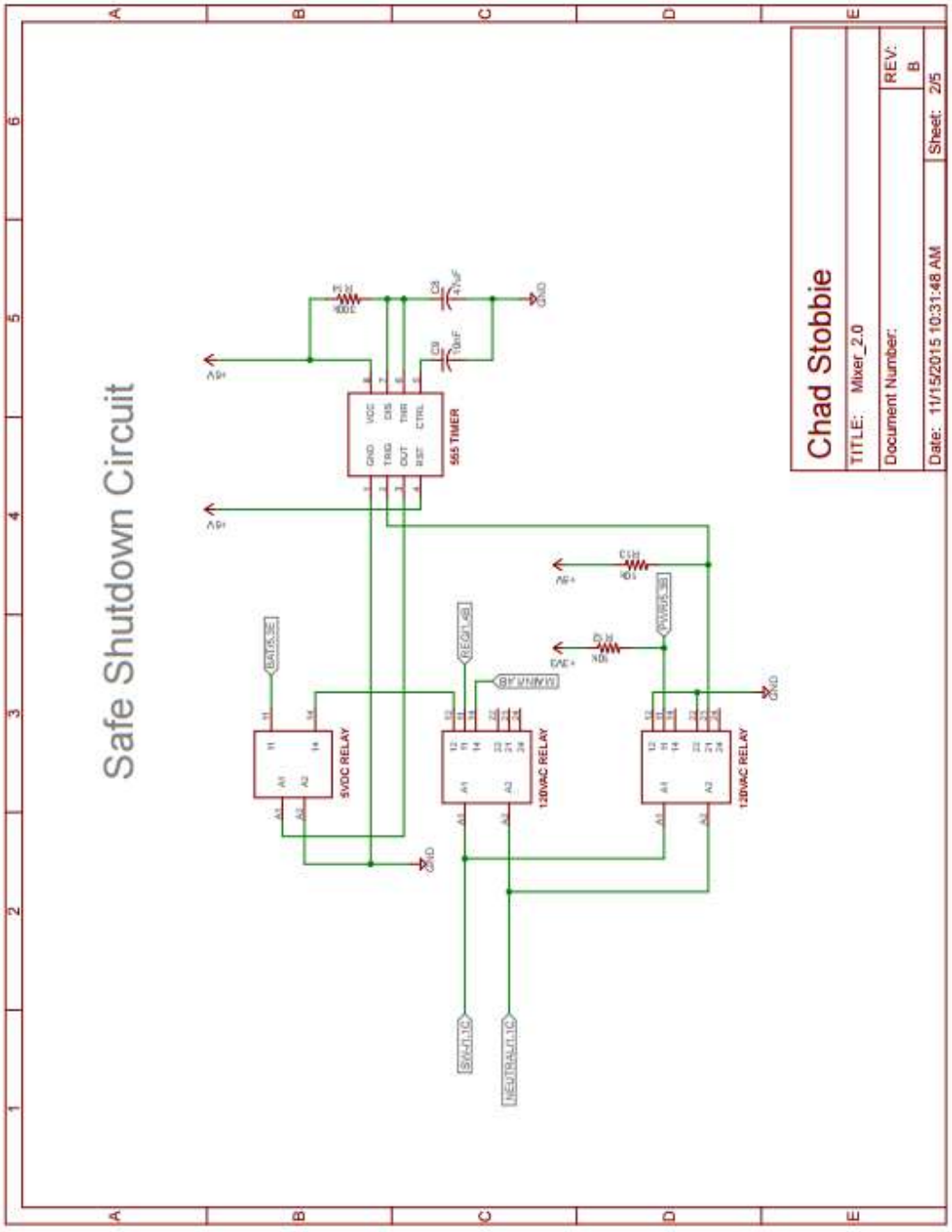
4. Connect the output to speakers via another 3.5 mm auxiliary cable; the port is located on the right-hand side of the mixer.
5. The LCD screen should already be activated and read “DEC 1503 MIXER”, along with the various channels’ volume levels.
6. Using the LCD screen for live feedback, adjust the volume knobs to the desired level /or mute the input channels completely by pressing firmly on the respective knob.
7. The knob on the right-hand side of the mixer will change the overall volume.
8. Enjoy the tunes!

APPENDIX III: CONCLUSION

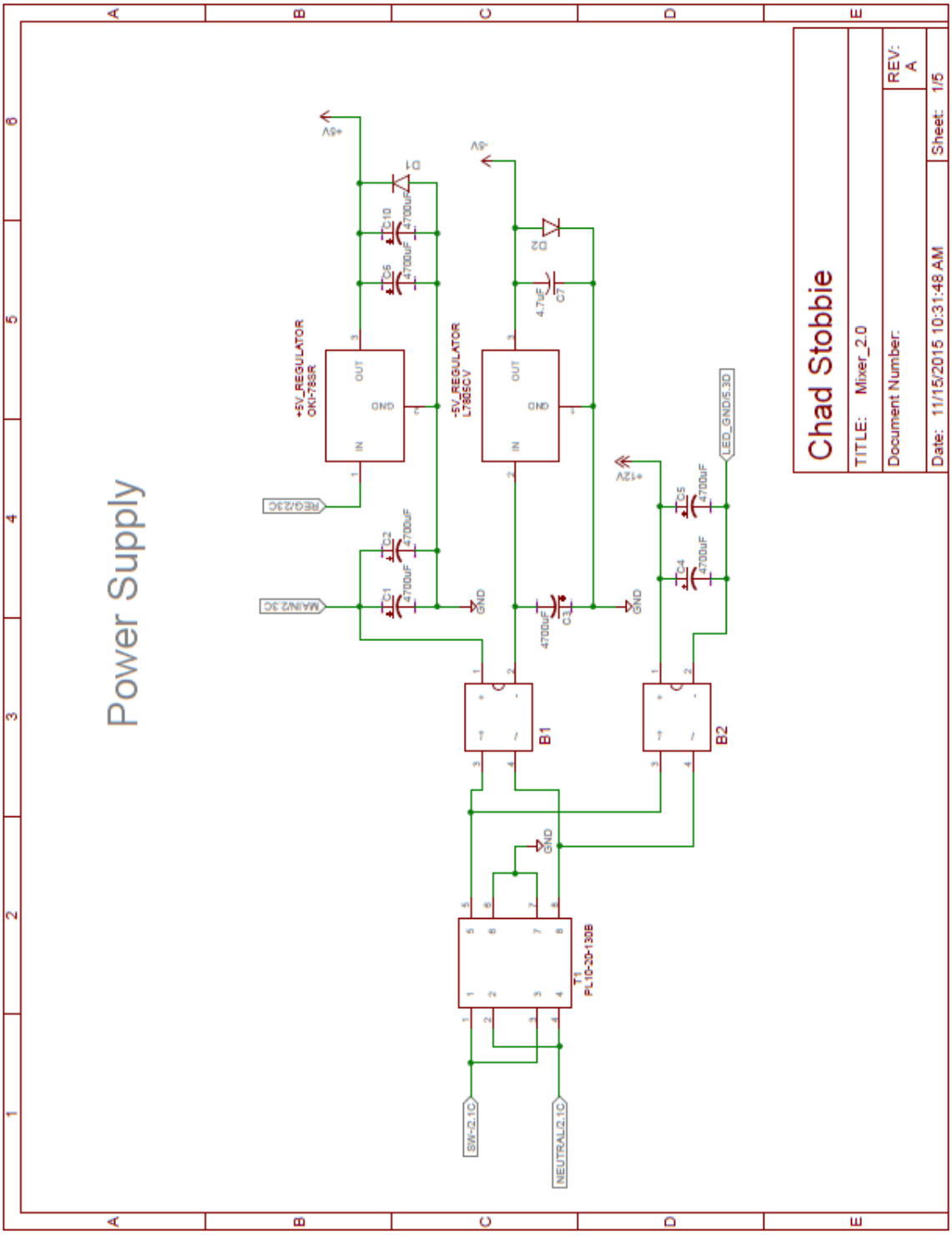
We were able to learn an incredible amount about audio components, hardware/PCB design, embedded Linux systems programming, project management and teamwork. We were able to produced two working prototype mixers. For December we were able to improve upon our May design with higher fidelity components, increased efficiency power supply, smaller and more compact enclosure, condensed, readable programming, and a simple, intuitive user interface. As well as solving some problems such as cable management, data corruption on shutdown and functioning Bluetooth audio.

APPENDIX IV: SCHEMATICS

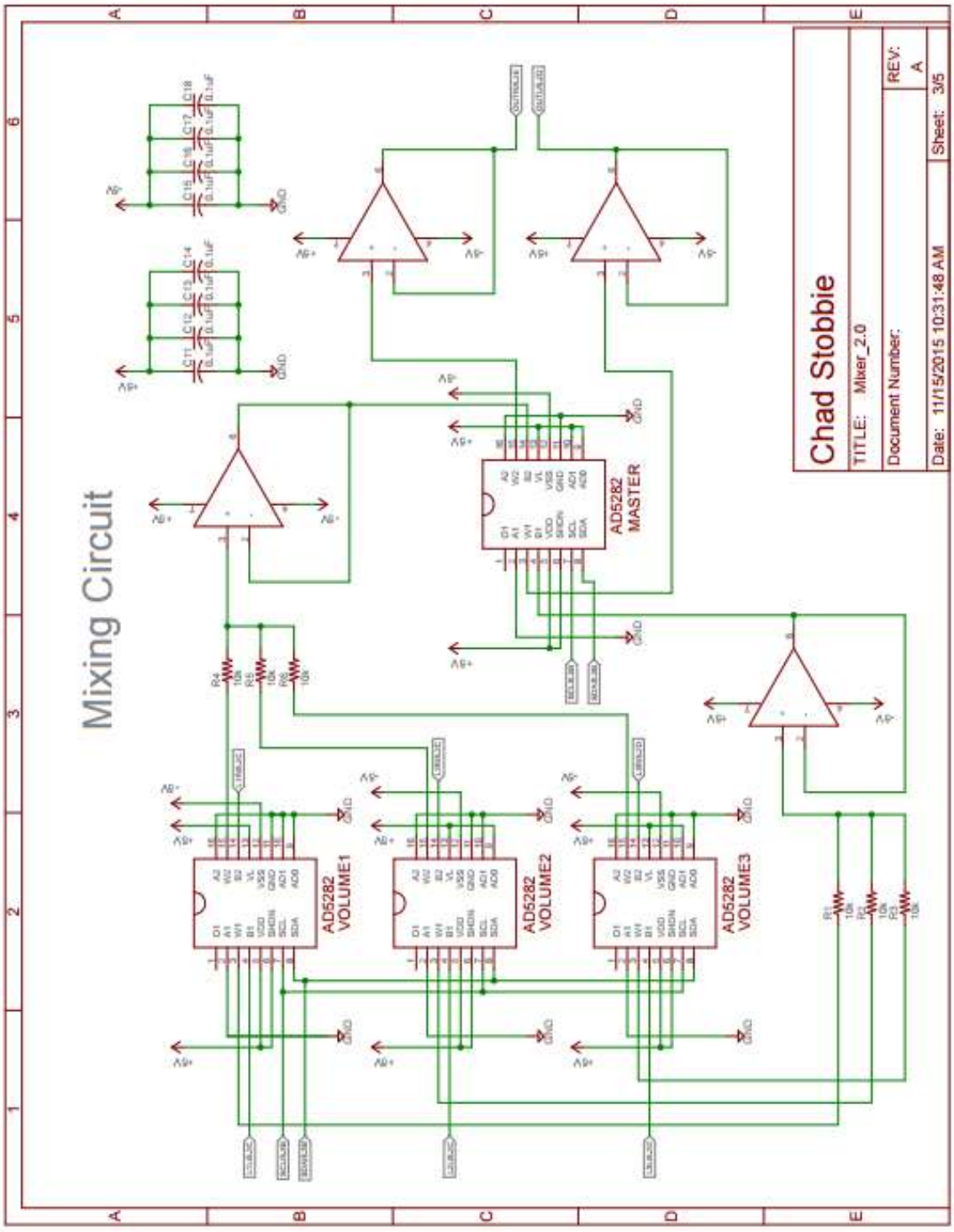
Safe Shutdown Schematic



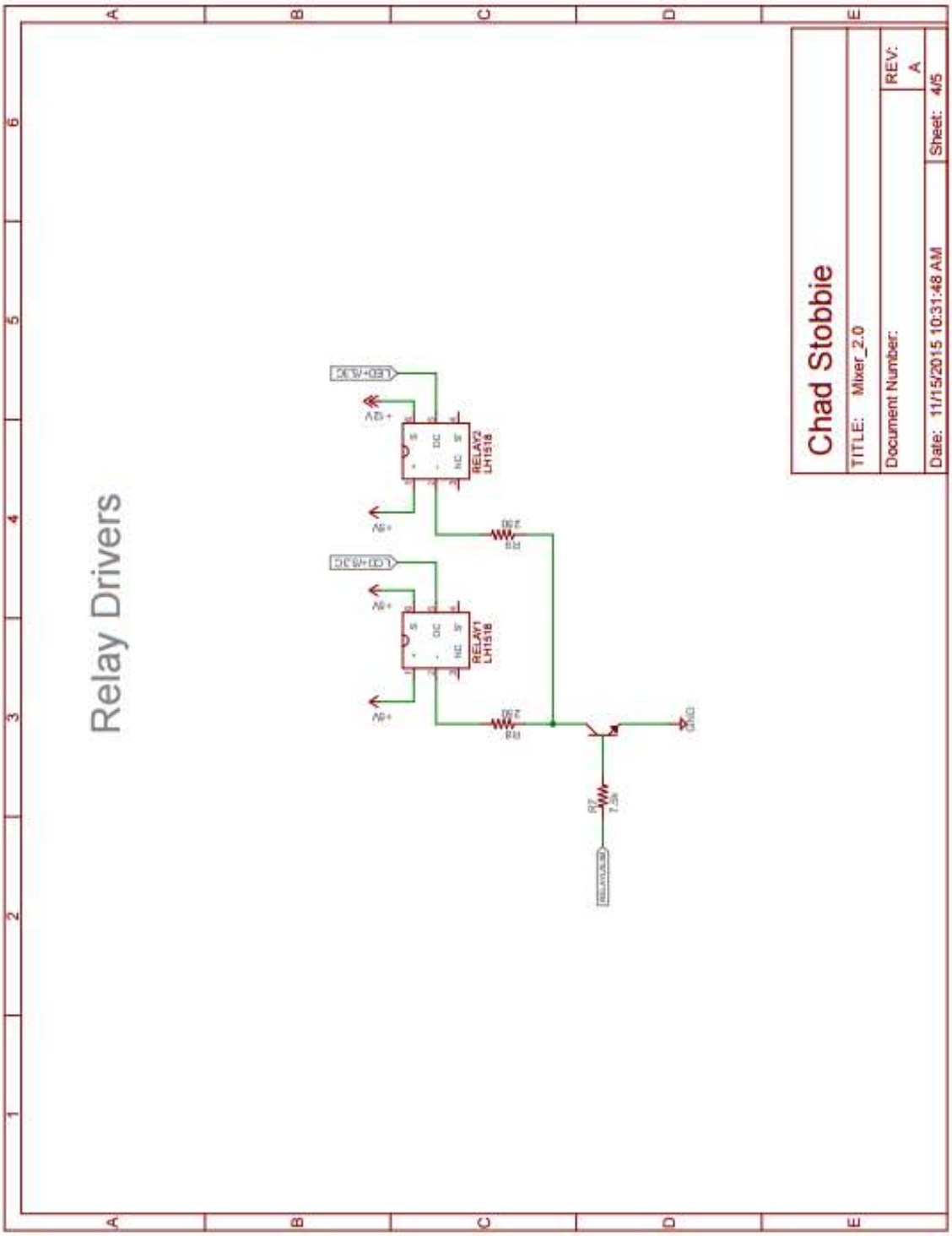
Power Supply Schematic



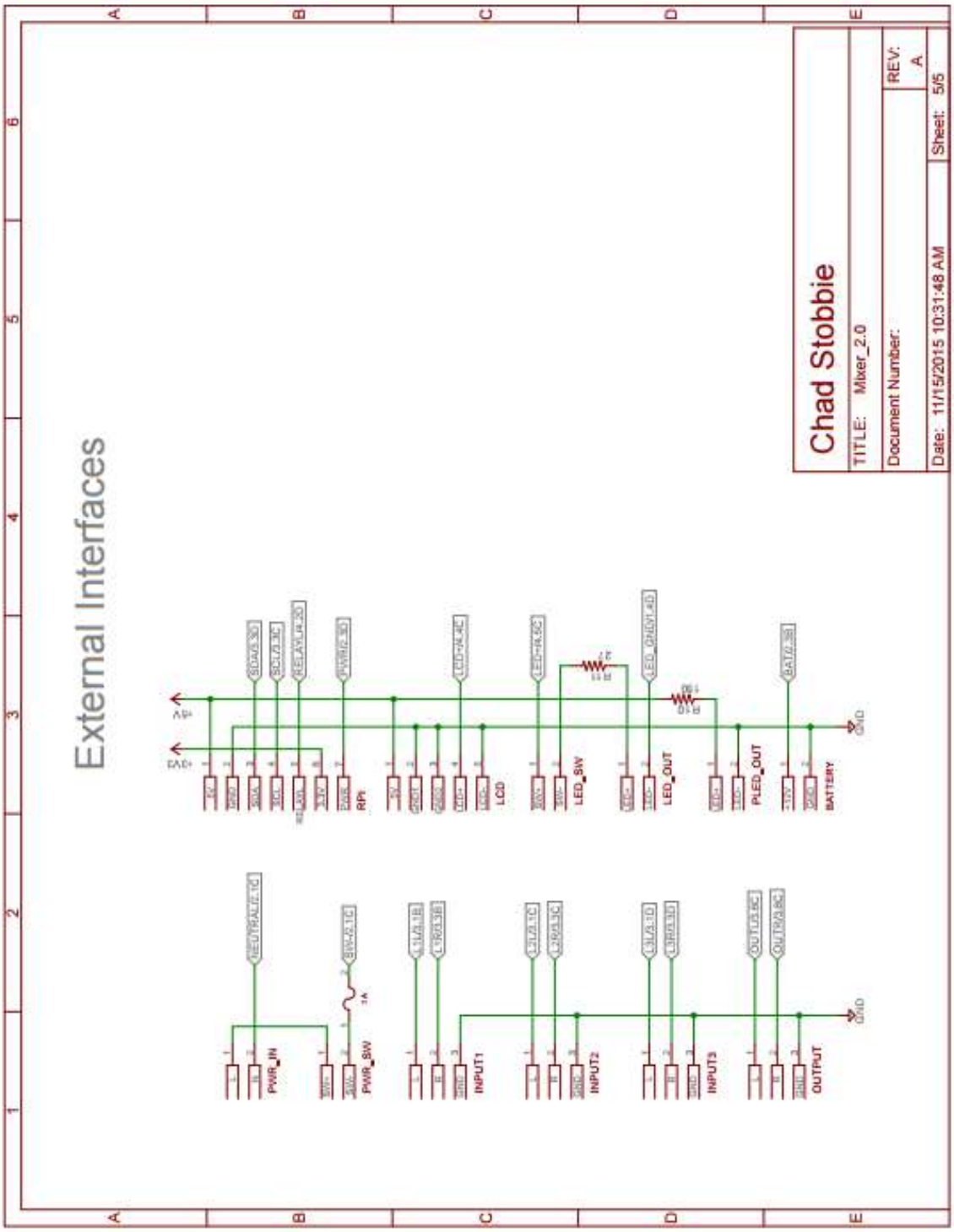
Mixing Circuit Schematic



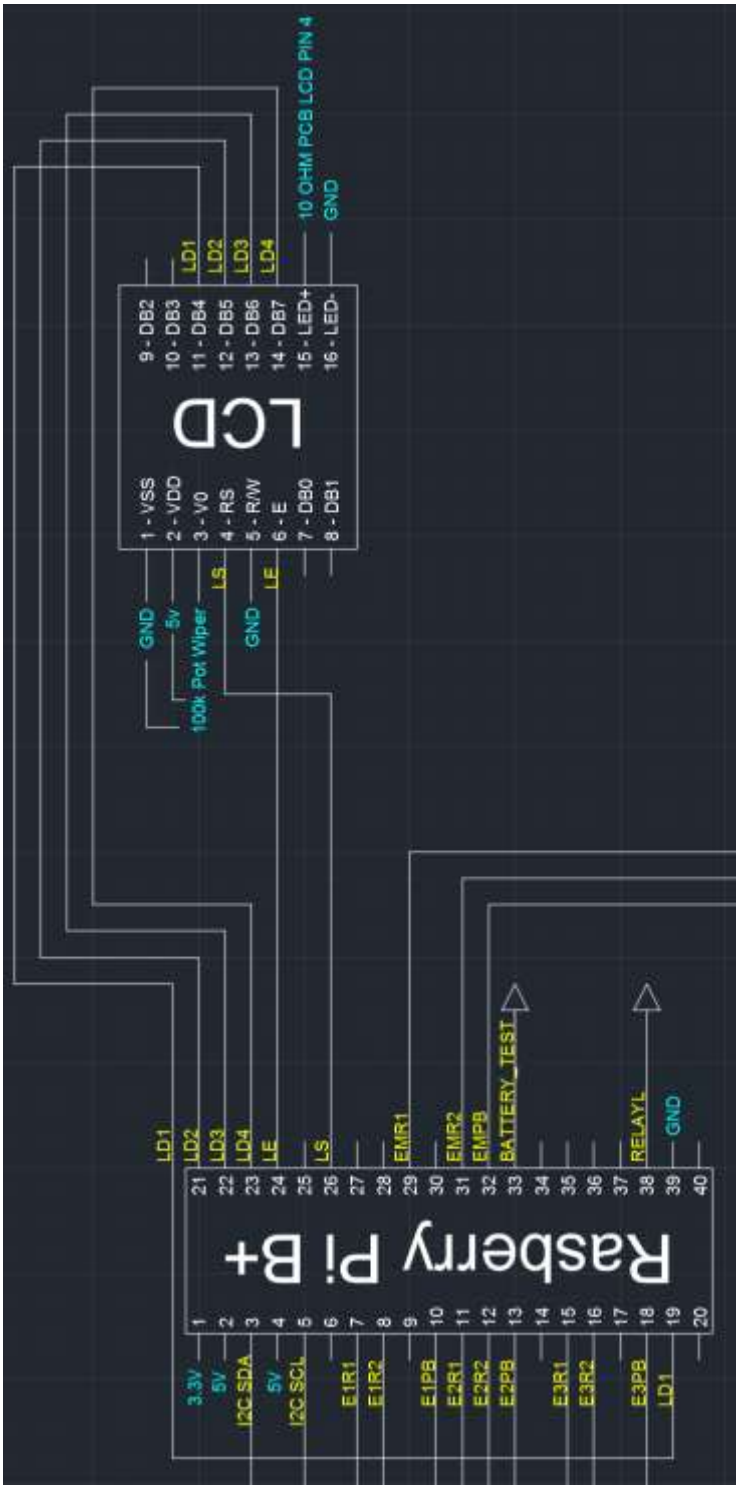
Relay Driver Circuit Schematic



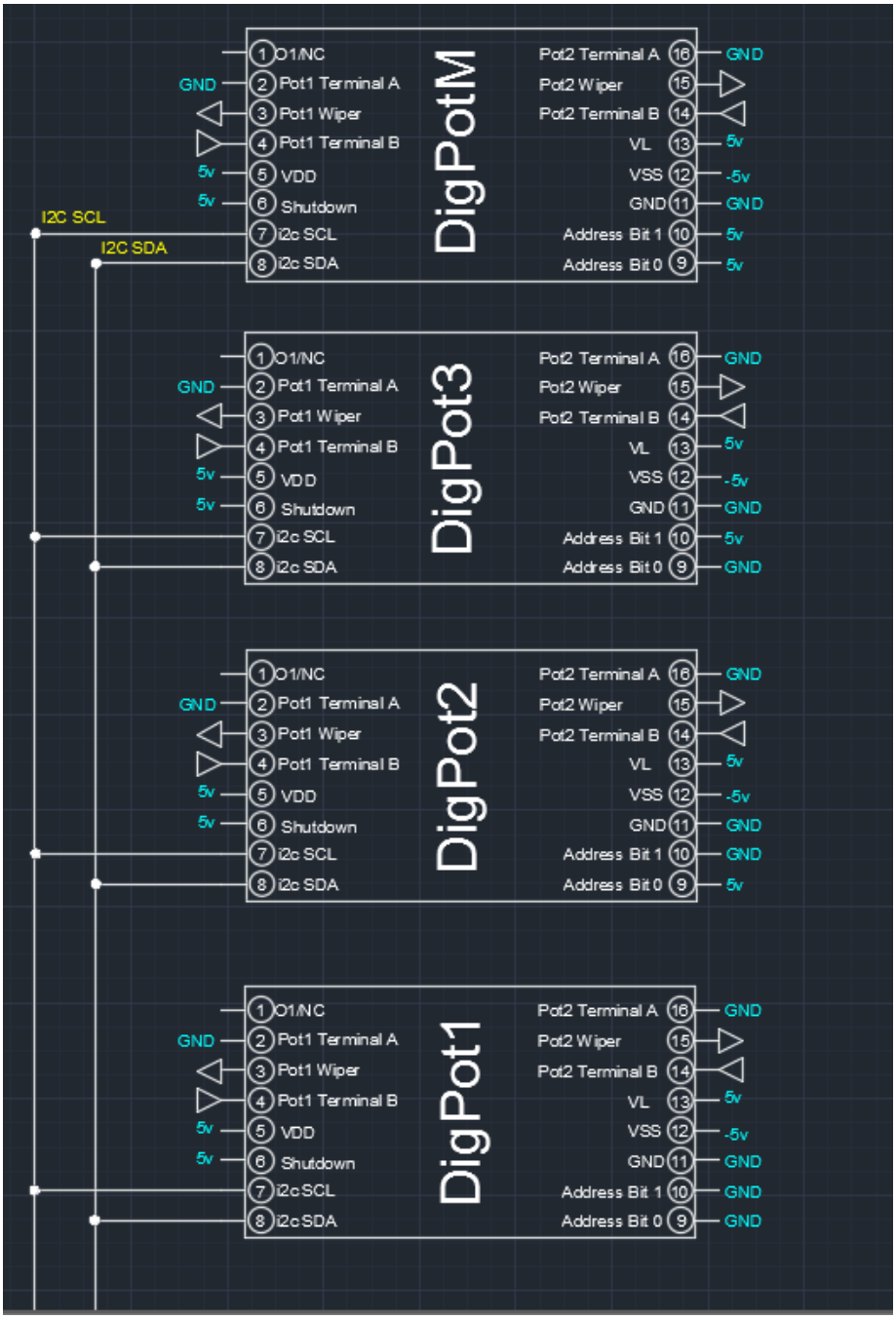
External Interfaces Circuit Schematic



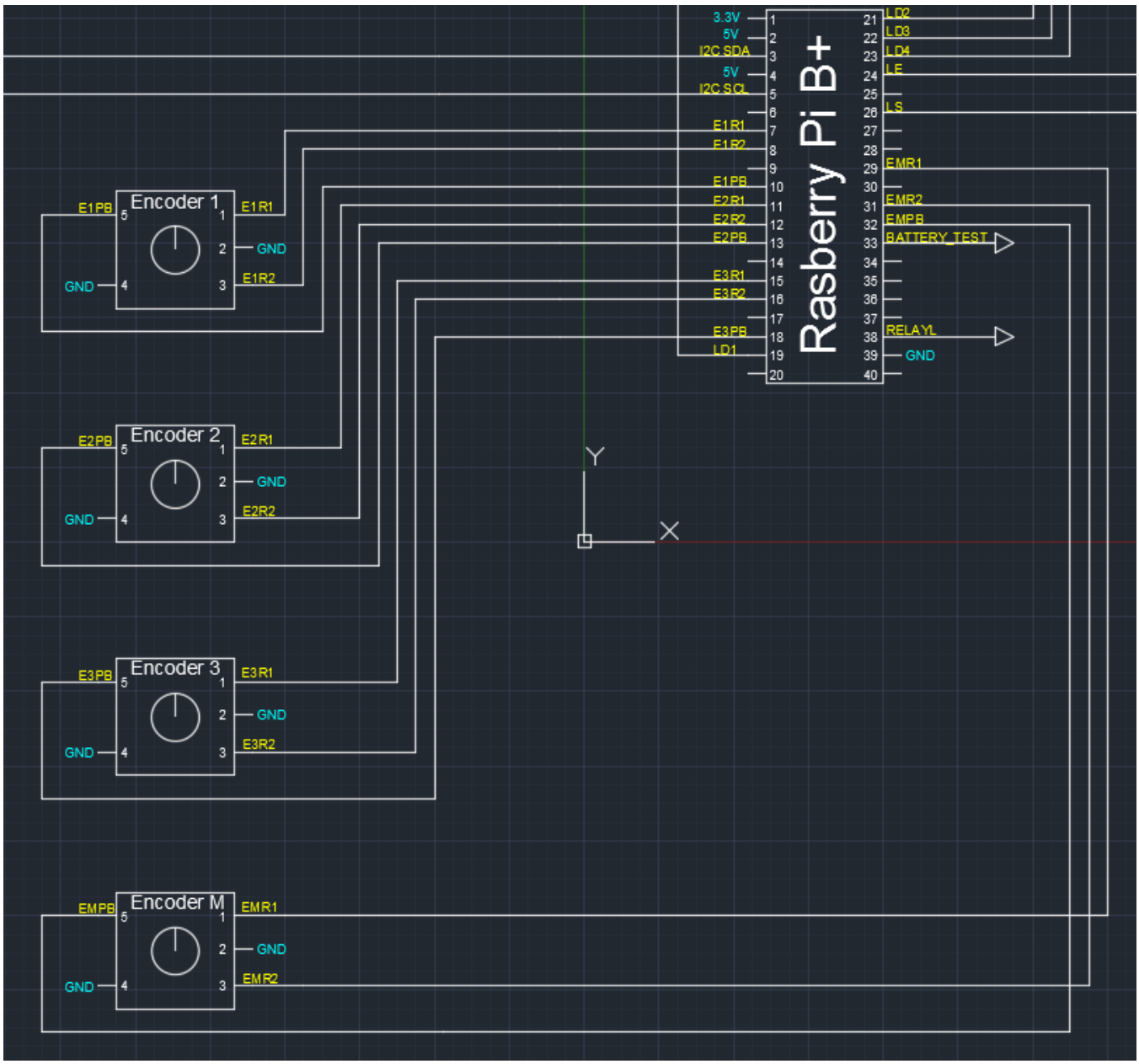
Raspberry Pi and LCD



Digital Potentiometers



Rotary Encoders



Programming Concept