Project Report

MACMIO- MIDI and Audio Control from Multiple Input and Output

Table Nine - Andrew Shepherd, Charles Lucero, Dalton Morrow, Daniel Kramer

Supervisor: Professor Melinda Piket-May

November 6, 2014

**GEEN 1400-50 – First Year Engineering Projects**

**Final Report**

**MACMIO – MIDI and Audio Control from Multiple Input and Output**

Table Nine - Andrew Shepherd, Charles Lucero, Dalton Morrow, Daniel Kramer

Project Supervisor: Professor Melinda Piket-May

University of Colorado Boulder

December 12, 2014

**Abstract**

The purpose of our project is to create an easy-to-use music player to be implemented into the Imagine Longmont Smart Home. Our client’s name is Stacey. She is in a wheelchair and has limited use of her hands, however she loves music. Our goal was to make a loop box which can play some of her favorite songs as well as allow her to play along to the music.

The final design is an acrylic box with six buttons and a slider. One large button will cycle through five programmed songs. The other five unique buttons will play sound effects using either MIDI signals or an Arduino-based synthesizer code, depending on the setup of the box. The slider will change the sounds that the five buttons make. The volume of the music and sounds is controlled by a proximity sensor, meaning that Stacey will be able to control the volume by moving her hand closer to or further from the box. The sound will be sent through a headphone/line output jack. We have also implemented a plug-in for removable flash storage.

The MACMIO (Mack Mee-oh) has three primary design components: circuitry, coding, and the box itself. Our circuitry uses two Arduino Redboards, an MP3 Trigger, a power-supply, and the control components. The circuit is programmed to play songs continuously, as well as sound effects based upon the button inputs.

**Table of Contents**

Contents

[List of Figures v](#_Toc406165271)

[1.0 Introduction 1](#_Toc406165272)

[2.0 Background Information 3](#_Toc406165273)

[3.0 Design 4](#_Toc406165274)

[4.0 Circuitry 7](#_Toc406165276)

[5.0 Coding 10](#_Toc406165277)

[5.1 Mozzi Board 10](#_Toc406165279)

[5.2 Non-Mozzi Board 10](#_Toc406165281)

[5.3 MP3 Trigger 11](#_Toc406165283)

[6.0 Packaging 12](#_Toc406165284)

[6.1 Components 12](#_Toc406165285)

[6.2 Layout 13](#_Toc406165286)

[7.0 Recommendations 14](#_Toc406165288)

[8.0 Conclusion 15](#_Toc406165289)

[Appendix A- Budget 17](#_Toc406165290)

[Appendix B- Solidworks 18](#_Toc406165291)

[Appendix C- Code 19](#_Toc406165292)

## List of Figures

[Figure 1: Arduino Redboard 9](#_Toc406165813)

[Figure 2: Gantt Chart October-November 11](#_Toc406165814)

[Figure 3: Gantt Chart November 11](#_Toc406165815)

[Figure 4: Gantt Chart November-December 11](#_Toc406165816)

[Figure 5: Circuitry Schematic 14](#_Toc406165817)

[Figure 6: MACMIO Solidworks 17](#_Toc406165818)

[Figure 7: MACMIO Interface 20](#_Toc406165819)

## 1.0 Introduction

Table Nine was given this project by Imagine! Colorado, a company which helps people who have developmental disabilities. They gave us a client whose needs we had to cater to as well as a set of design requirements which had to be met in the final product. The remaining features of the project were open to our own creative interpretation. After receiving the requirements we set about deciding what we wanted the MACMIO to do as well as how it was to accomplish its various tasks. We had to consider what user controls we would implement, what circuit boards would allow for us to produce sounds, and what kind of electronics would be necessary to make all the components run. Our design process consisted of three general sections: circuitry, coding, and packaging. These will be covered later in the report and will show what kinds of decisions we had to make in order to produce a functioning music box.

Our group worked very well together in making the MACMIO. Each member made important contributions to the project by finding a particular role and fulfilling it. Andrew is very experienced with circuitry and electronics. He made sure that all components were powered properly and handled the overall schematic of our design. Charles was the only member with enough programming experience to make our Arduinos work properly. The coding of the two Redboards was essential to the functionality of the MACMIO and it was very complex. Dalton made contributions to the project by handling the logistics and other general problem-solving; this included the budget and the human aspect of the design. Finally, Danny was in charge of initial design and packaging. He has worked with sound-based technology before, so he knew how we could begin to go about constructing a device with the requirements we were given. He also did an excellent job with the Solidworks designs used to make the container. Table Nine was an effective team; each member had a job which they knew how to do and worked very hard at in order to create the MACMIO.

## 2.0 Background Information

The design requirements were derived from the needs of our client, Stacy. Stacy has limited motor capability and movement and is confined to a wheelchair. She can control her hands but only to an extent. Stacey also loves music, though her impairments prevent her from taking up traditional instruments. Our goal was to make some sort of instrument that she can use to play along to her favourite artists. Stacy lives in the Longmont Smarthome with others like her; this project will cater to their uses as well. This means we had to make the box functional but very easy to use. The basic design requirements were suggested to us as follows:

We were required to have a single large button to change the music track. In addition, there had to be other controls which could produce sound effects. These buttons must be large and easy to press so someone like Stacy will not have trouble actuating them. Songs must be uploadable via a USB port or SD card. Also, the buttons pressed must send generic MIDI signals via USB for computer interface. The volume control must be adjustable by a non-tactile proximity sensor of some sort. This way Stacy can more easily control the output volume of the device.

## 3.0 Design

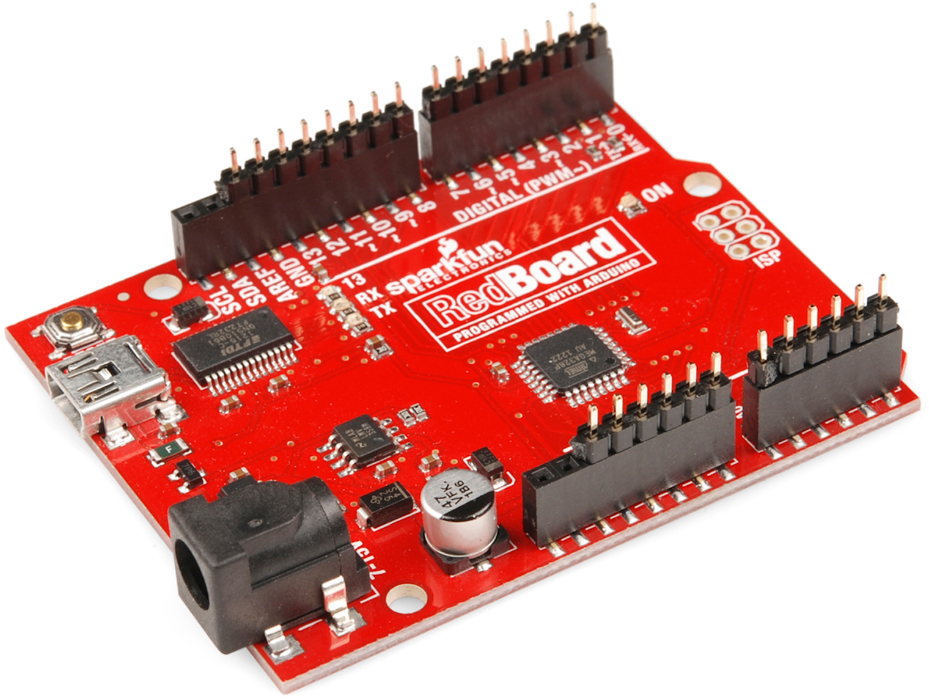
Initially, we were given our design requirements and a simple sketch of the general design of the final product. We then started designing the circuitry and all of the internal components. We had to make sound effects through musical code called MIDI and through internally produced sounds. To do this we decided to use Arduino RedBoards, circuit boards with pins that can be treated as inputs or outputs and programmed to do specific tasks. 

Figure : Arduino Redboard

Our first step was to output MIDI signal from the Arduino to the computer. In order to do this we programmed the Arduino to take a digital input from a button and turn an LED light on and off. From there, we were essentially able to replace the LED with a MIDI message. MIDI is a serial protocol that sends a message to software on a PC computer. The software on the PC can then be mapped so that the MIDI messages it receives will trigger any effect or sound.

Once we had MIDI figured out, we moved on to the onboard synthesizer effects known as MOZZI. MOZZI is a library of synthesizer effects created by Tim Barrass and we used it under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. We used several effects from the MOZZI library and designed the circuitry and programming so that the sound effects could be played over an MP3 track that was playing.

After we had all of our circuitry and programming completed we moved on to building and constructing the packaging and troubleshooting. The design and the construction of the packaging was relatively straight forward and once we got all of the electronics into place and everything connected we were able to spend about two weeks on troubleshooting and perfecting all aspects of the MACMIO. We ran into quite a few issues once we started to try to get everything to work all at the same time and almost all of our time as we got closer to the project deadline was spent troubleshooting and fixing the problems that we had.

Throughout the design process we had a schedule that we used to keep us on track (Fig 1). We also scheduled client meetings throughout the design process so that we could make sure that we were building the MACMIO exactly as our client wanted. The schedule helped us budget our time and we had enough time at the end to fix every problem that we had.

## **Gantt Chart 1.JPG**

Figure : Gantt chart October-November

## **Gantt Chart 2.JPG**

Figure : Gantt chart November

## **Gantt Chart 3.JPG**

Figure : Gantt chart November-December

## 4.0 Circuitry

The button panel has six buttons, five of them produce synthesized sounds and MIDI signals simultaneously. The sixth button, the big red one, will trigger the MP3 player. The button array is routed to each Arduino and the MP3 shield. The synthesized sounds from the MOZZI Arduino are fed into a low-pass audio filter to remove most of the high-frequency PWM hash on the signal. The output of the filter is fed into an audio mixer.

The MIDI Arduino will, in addition to the buttons, also have two analog inputs for a slider and a proximity sensor. The Arduino will convert the analog signal from the proximity sensor to a digital PWM output to control a servo which will turn the master volume pot on the mixer. The slider potentiometer will control the MIDI notes sent to the computer. A MIDI to USB converter comes out of the back of the MACMIO to interface with the computer.

The mixer will combine the signals of the MP3 player and the synthesizer Arduino. The mixer output will be a ⅛” phone jack for Hi-Fi or computer speakers to patch into. The mixer requires a regulated 9 volt supply. To generate this, the 12 volt master supply is fed into an LM317-based adjustable regulator trimmed out to 9 volts at the output. Basically, the 12 volt supply is fed into a chip configured to drop the voltage to 9 volts.

Each Arduino will be powered directly from the 12 volt supply. The built-in regulator will drop the voltage to 5 volts to power themselves as well the MP3 trigger and the buttons. A tricolor RGB LED strip will be powered directly off of the 12 volt supply. We have a tricolor LED strip because then multiple colours can be generated. The MIDI Arduino will control its color via three of its PWM (Pulse Width Modulation) output pins each buffered by power transistors (Fig. 2).

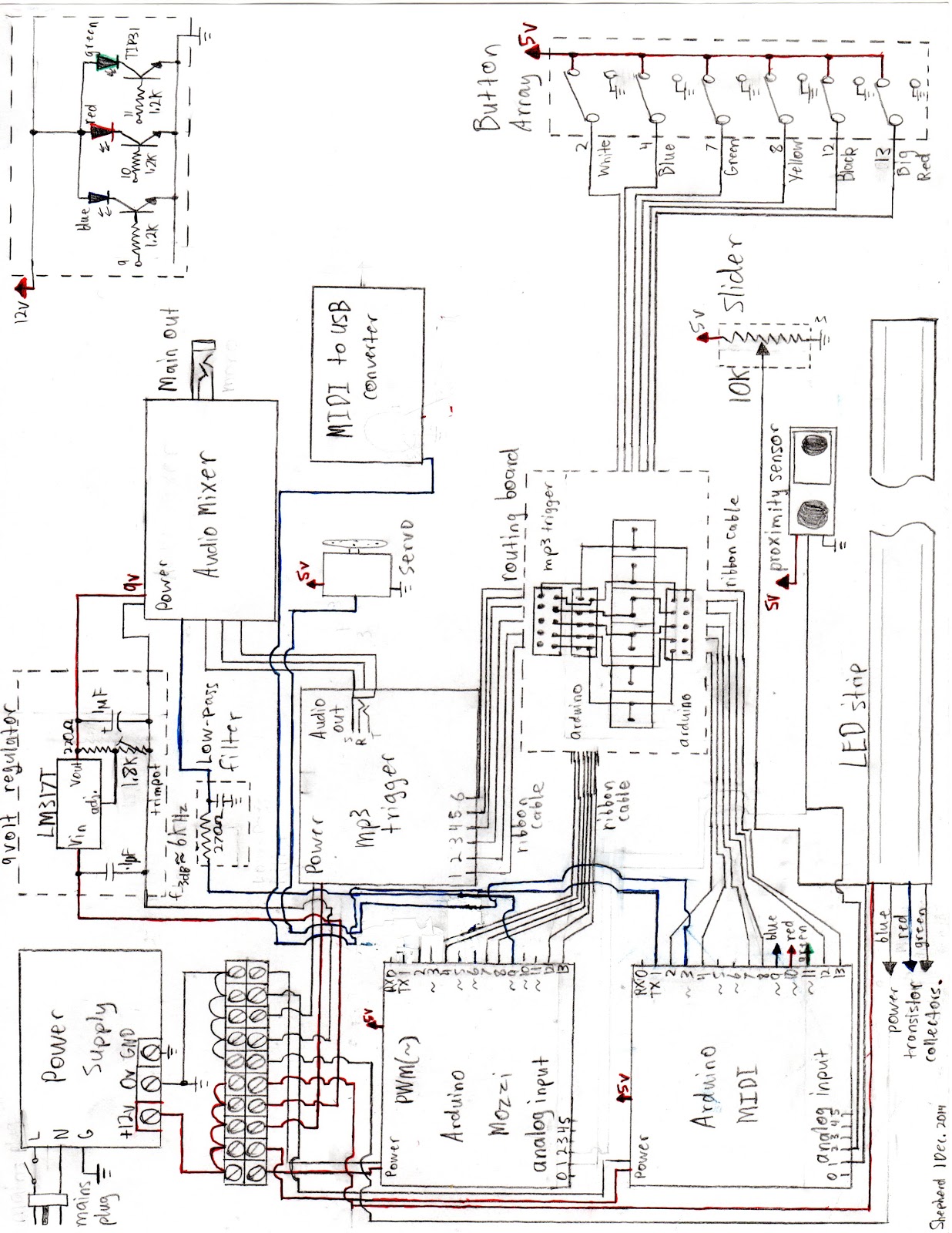


Figure : Circuitry Schematic

## 5.0 Coding

### In order to get the box to produce sounds, we programmed two Arduino RedBoards and an MP3 Trigger.  One RedBoard is completely dedicated to using MOZZI libraries in order to create synthesized overlays that will play from the box itself.  The second RedBoard is responsible for sending MIDI triggers to a computer and controlling the box’s volume.  The MP3 trigger is responsible for playing various songs.

### 5.1 MOZZI Board

### In order to create synthesized overlays, we found and utilized a set of open-source Arduino libraries named MOZZI.  We chose five sounds that we liked and altered their respective codes so that they would all fit into one Arduino sketch file.  The result is a file that, when utilized with our system, creates different sounds depending on which button is being pressed.  Due to the size of this one file almost taking up all of the allotted space per file, we were required to use a second board in order to give the box all of its necessary features.

### 5.2 Non-Mozzi Board

### The second board has three main jobs.  First, it must send a MIDI signal.  In order to do this, we found an open-source code that allows any Arduino Board to send MIDI signal.  We then implemented it in to the sketch for the second board so that when a button is pressed or when the slider is adjusted the sent signal will change accordingly.  We also wrote in code that when the infrared sensor gets certain readings will change a servo’s position in order to manipulate the volume of the speaker.

### 5.3 MP3 Trigger

The MP3 trigger plays MP3 files off of a Micro SD card at the push of a button. The MP3 files are loaded onto the SD card along with a simple initialization file then tells the device what to do when a specific trigger is shorted. How the board works is there are 18 trigger pins and when one of them is shorted that track number plays or the board does the action specified in the initialization file. We wrote the initialization file so that pressing the large red button on the MACMIO would play a song and then skip to the next song on the card each time it is pressed. The MP3 Trigger is also attached to an access panel so that the end user can take out the SD card and change the tracks that are on it.

## 6.0 Packaging

The MACMIO is constructed with white colored acrylic bonded together using high strength plastic welder epoxy. The chassis provides the platform that all the electrical components are attached to. The acrylic pieces were designed using SolidWorks CAD software and those CAD files were then used to laser cut the individual pieces. There are six pieces that make up the rectangular box that is 18 inches long by 6 inches wide by 4.5 inches high (Fig. 2). The pieces are joined at the corners with a dovetail joint, small notches that interlock with each other to provide a secure joint.

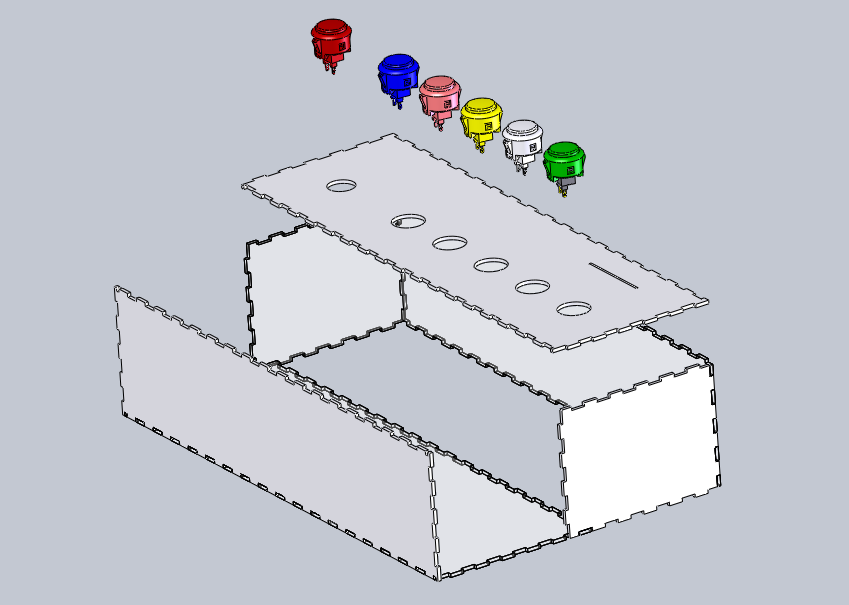


Figure : MACMIO Solidworks

### 6.1 Components

Each electrical component is bolted to the acrylic chassis using ½ inch bolts and nylon insert lock nuts. There is an access panel in the bottom of the box that allows the user to access the MP3 player circuit board and change the flash storage. The access panel is held in place by two magnets inside the box that attach to two metal L-brackets on the panel itself. The MP3 player circuit board is then bolted to the access panel and the attached wiring keeps it from being pulled out too far. The six buttons on the top of the box thread into large holes in the acrylic and are secured by plastic nuts. The proximity sensor and slider are also attached with screws through the top of the box. Inside the box there is a servo that moves a volume knob on the mixer circuitry based on the position of the users hand in relation to the proximity sensor. The servo is attached to the chassis with long standoff spacers in line with the volume knob. A timing belt is then attached between the servo and the volume knob.

### 6.2 Layout

### The inside of the box is laid out with the power supply on the far right side and all mains power cables running to a terminal strip next to the power supply. The main power switch is located on the front of the box for easy user access and is directly above the power supply. In the center the two Arduino RedBoards are bolted to the chassis and the access panel with the MP3 player circuit board is on the far left. On the top of the box all six buttons are near the front edge to promote ease of use for the user and the centers of the buttons are in line to be aesthetically pleasing. The sider is located just behind the buttons and the proximity sensor is located next to the slider so that it is not accidentally triggered when the user is pressing the buttons on the front edge. On the front face of the box is also the audio jack for headphones or speakers and the USB port for MIDI connection to a PC is on the back next to the power cable.

## 7.0 Recommendations

When building the MACMIO we encountered several problems involving circuitry and coding. This was mainly due to the relative complexity of the system we used.  One such problem was that when we were assembling the project, our circuit boards stopped working.  We determined that the cause of this was that we had unregulated 12 volts running through the boards.  This compromised the circuitry in the boards, making them unusable.  Our advice would be to make sure that all voltage to the boards be regulated by feeding it through the DC jack on each board.  We also advise making sure that each of the pin contacts are insulated so as to prevent shorts that could also brick the boards.

Once the electronics were deemed functional, we had to troubleshoot our code. It was very long due to the complexity of using synthesizer code. What we wanted to do with our project was too much for one Arduino to handle, which is why we incorporated two of them in our project.  Instead of using Arduino, we advise using a board along the lines of a Raspberry Pi; something with lots of processing power which is typically used for music.

The final problem we encountered was difficulty in closing our box.  This was a result of having not enough space, too many wires, and not planning where every component would go.  We advise making the encasement for the circuitry and components after connecting all of the components together and fitting them together.

## 8.0 Conclusion

The MACMIO consists of three major components, each denoted by the four circuit boards. The first Arduino works as a synthesizer programmed with MOZZI code. When the arcade buttons are pressed preset sounds are synthesized and outputted. The second Arduino is the non-Mozzi controller. It has three functions: it outputs MIDI trigger signals, controls the LEDs, and controls the volume. The third component is an MP3 trigger which is activated by the big red button only. Its job is to play songs which can be changed by opening the access port on the bottom of the box and removing the micro SD card from the MP3 trigger. The fourth component is an audio mixer that combines the MP3 audio and the Mozzi audio together. This module also serves as a volume control and as the audio output.



Figure : MACMIO Interface

The result of this array of boards and components is two methods by which to make music with the MACMIO: internal music and sound effects or external MIDI notes. This computer interface capability enables the user to have more sound variety.

Although we encountered problems and unforeseen difficulties, making this project was fun.  At each step of the design process we were required to use all of our technical and problem solving skills.  This would include determining what components we needed, how they would come together, and how we would get them to work.  Accomplishing this would prove to be both mentally and physically taxing, but in the end, when everything worked, we knew that it was well worth the effort.

This box was designed for ease of use so Stacy and others like her will be able to interact directly with their music. Our hope is that Imagine! Colorado and Stacey will find the MACMIO to be fun and useful in their upcoming music center.

## Appendix A- Budget

Arduino Redboard $19.95 X2

Replacement Redboard $19.95

MP3 Trigger $49.95

MIDI to USB $6.29

MIDI to USB Interface Cable $5.19

Large Red Button $5.95

Pushbutton Switches $1.95X5

100 ohm Resistors X6

Crimp Wire Connectors

Infrared Proximity Sensor $13.95

10k Slider Potentiometer $2.95

Servo

Hub Mount – Quad $5.99

Shaft – D-Shaft $1.49

 Servo Mount – Servo Plate B $6.99

   Timing Pulley – Shaft Mount (16T, 0.25” Bore) $8.95

   Aluminum Channel – 3.0” $3.99

   Servo Shaft Coupler – Hitec Standard $4.99

   Ball Bearing – Flanged (1/4” Bore, ½” OD) $2.19

Switching 12 Volt Power Supply $16.95

Tricolor LED Strip (RGB) $9.95

Mains Missile Switch $3.90

Audio Mixer: $22.39

Shrink Tubing - Free

SD Card $5.93

USB Connector $12.34

Timing Belt $7.05

Acrylic - $25.95

Poster $45

Misc Bolts etc. $13.30

L-Brackets and Connectors $8.68

Apoxy $5.99

Electrical Tape $0.99

Miscellaneous  $60

Arduino Headers

Chassis Mount Terminal Block

PC Mount Terminal Block

Wiring

Ribbon Cable w/headers

Perfboard

Mains Cable

¼” Male Stereo Connectors

⅛” Female Mono Connector

⅛” Male Stereo Connector

Audio Low-Pass Filter:

270 ohm Resistor

.1uF Capacitor

9 volt Power Regulator:

LM317T Adjustable Regulator

.1uF cap

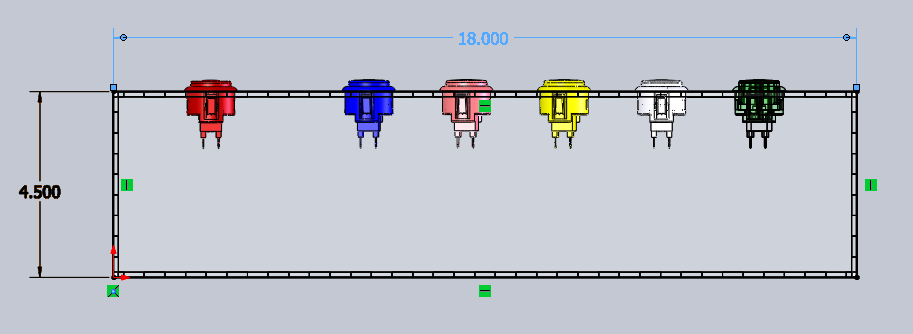
1uF cap

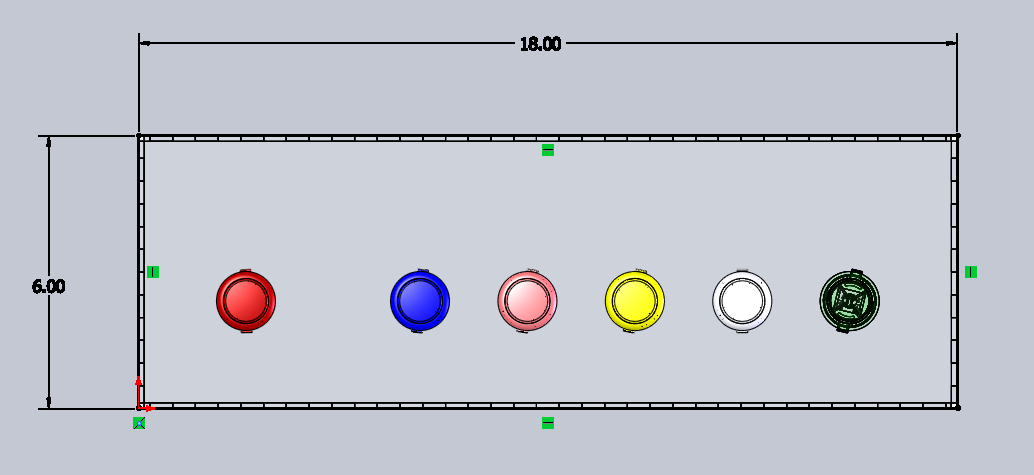
220 ohm Resistor

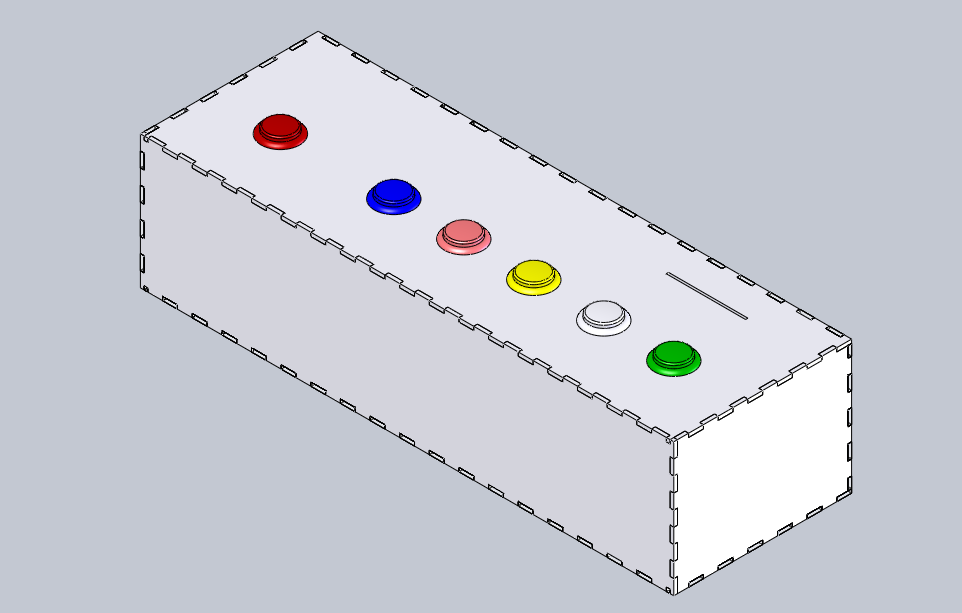
1.8k Trimpot

**Total $426.89**

Appendix B- Solidworks

****

****

****

## Appendix C- Code

Mozzi Arduino Code

//////////////////////////////////Sinewave/////////////////////////////////////

#include <MozziGuts.h>

#include <Oscil.h>

#include <tables/cos8192\_int8.h>

Oscil <COS8192\_NUM\_CELLS, AUDIO\_RATE> aSin(COS8192\_DATA);

///////////////////////////////////////////////////////////////////////////////

//////////////////////////////////Line\_Gliss///////////////////////////////////

#include <MozziGuts.h>

#include <Line.h>

#include <Oscil.h>

#include <tables/triangle\_warm8192\_int8.h>

#include <mozzi\_midi.h>

Oscil <TRIANGLE\_WARM8192\_NUM\_CELLS, AUDIO\_RATE> aTriangle(TRIANGLE\_WARM8192\_DATA);

Line <long> aGliss;

byte lo\_note = 24;

byte hi\_note = 36;

long audio\_steps\_per\_gliss = AUDIO\_RATE / 4;

long control\_steps\_per\_gliss = CONTROL\_RATE / 4;

int counter = 0;

byte gliss\_offset = 0;

byte gliss\_offset\_step = 2;

byte gliss\_offset\_max = 36;

///////////////////////////////////////////////////////////////////////////////

//////////////////////////////////PWM\_Phasing//////////////////////////////////

#include <MozziGuts.h>

#include <Phasor.h>

Phasor <AUDIO\_RATE> aPhasor1;

Phasor <AUDIO\_RATE> aPhasor2;

float freq = 55.f;

///////////////////////////////////////////////////////////////////////////////

///////////////////////////////////AMsynth/////////////////////////////////////

#include <MozziGuts.h>

#include <Oscil.h>

#include <tables/COS8192\_int8.h>

#include <mozzi\_fixmath.h>

#include <EventDelay.h>

#include <mozzi\_rand.h>

#include <mozzi\_midi.h>

Oscil<COS8192\_NUM\_CELLS, AUDIO\_RATE> aCarrier(COS8192\_DATA);

Oscil<COS8192\_NUM\_CELLS, AUDIO\_RATE> aModulator(COS8192\_DATA);

Oscil<COS8192\_NUM\_CELLS, AUDIO\_RATE> aModDepth(COS8192\_DATA);

EventDelay kNoteChangeDelay;

Q8n8 ratio;

Q24n8 carrier\_freq;

Q24n8 mod\_freq;

Q8n0 octave\_start\_note = 42;

///////////////////////////////////////////////////////////////////////////////

/////////////////////////////////////ReverbTank////////////////////////////////

#include <MozziGuts.h>

#include <ReverbTank.h>

#include <Oscil.h>

#include <tables/cos8192\_int8.h>

#include <tables/envelop2048\_uint8.h>

ReverbTank reverb;

Oscil <COS8192\_NUM\_CELLS, AUDIO\_RATE> bCarrier(COS8192\_DATA);

Oscil <COS8192\_NUM\_CELLS, AUDIO\_RATE> bModulator(COS8192\_DATA);

Oscil <COS8192\_NUM\_CELLS, AUDIO\_RATE> aModWidth(COS8192\_DATA);

Oscil <COS8192\_NUM\_CELLS, CONTROL\_RATE> kModFreq1(COS8192\_DATA);

Oscil <COS8192\_NUM\_CELLS, CONTROL\_RATE> kModFreq2(COS8192\_DATA);

Oscil <ENVELOP2048\_NUM\_CELLS, AUDIO\_RATE> aEnvelop(ENVELOP2048\_DATA);

///////////////////////////////////////////////////////////////////////////////

/\*

Above is the inclusion of files, tables, and libraries from the MOZZI library that each sound requires.

These are referrenced later in order to create synthesized effects.

Below is how each sound manipulates the included libraries and tables in order to create the synthesized sounds.

In order for this to work, the main MOZZI library must be downloaded from http://sensorium.github.io/Mozzi/

\*/

#define CONTROL\_RATE 64

void setup(){

startMozzi(CONTROL\_RATE); //All//

randSeed(); //Multiple//

aSin.setFreq(440); //Sinewave//

aPhasor1.setFreq(freq); //PWM\_Phasing//

aPhasor2.setFreq(freq+0.2f); //PWM\_Phasing//

ratio = float\_to\_Q8n8(3.0f); //AMsynth//

kNoteChangeDelay.set(200); //AMsynth//

aModDepth.setFreq(13.f); //AMsynth//

bCarrier.setFreq(55); //ReverbTank//

kModFreq1.setFreq(3.98f); //ReverbTank//

kModFreq2.setFreq(3.31757f); //ReverbTank//

aModWidth.setFreq(2.52434f); //ReverbTank//

aEnvelop.setFreq(9.0f); //ReverbTank//

}

void updateControl(){

if (--counter <= 0){ //Line\_Gliss//

gliss\_offset += gliss\_offset\_step; //Line\_Gliss//

if(gliss\_offset >= gliss\_offset\_max) gliss\_offset=0; //Line\_Gliss//

int start\_freq = mtof(lo\_note+gliss\_offset); //Line\_Gliss//

int end\_freq = mtof(hi\_note+gliss\_offset); //Line\_Gliss//

long gliss\_start = aTriangle.phaseIncFromFreq(start\_freq); //Line\_Gliss//

long gliss\_end = aTriangle.phaseIncFromFreq(end\_freq); //Line\_Gliss//

aGliss.set(gliss\_start, gliss\_end, audio\_steps\_per\_gliss); //Line\_Gliss//

counter = control\_steps\_per\_gliss; //Line\_Gliss//

} //Line\_Gliss//

static Q16n16 last\_note = octave\_start\_note; //AMsynth//

if(kNoteChangeDelay.ready()){ //AMsynth//

if(rand((byte)5)==0){ //AMsynth//

last\_note = 36+(rand((byte)6)\*12); //AMsynth//

} //AMsynth//

if(rand((byte)13)==0){ //AMsynth//

last\_note += 1-rand((byte)3); //AMsynth//

} //AMsynth//

if(rand((byte)5)==0){ //AMsynth//

ratio = ((Q8n8) 1+ rand((byte)5)) <<8; //AMsynth//

} //AMsynth//

if(rand((byte)5)==0){ //AMsynth//

ratio += rand((byte)255); //AMsynth//

} //AMsynth//

last\_note += 3 \* (1-rand((byte)3)); //AMsynth//

Q16n16 midi\_note = Q8n0\_to\_Q16n16(last\_note); //AMsynth//

carrier\_freq = Q16n16\_to\_Q24n8(Q16n16\_mtof(midi\_note)); //AMsynth//

mod\_freq = (carrier\_freq \* ratio)>>8; //AMsynth//

aCarrier.setFreq\_Q24n8(carrier\_freq); //AMsynth//

aModulator.setFreq\_Q24n8(mod\_freq); //AMsynth//

kNoteChangeDelay.start(); //AMsynth//

} //AMsynth//

bModulator.setFreq(277.0f + 0.4313f\*kModFreq1.next() + kModFreq2.next()); //ReverbTank//

}

int updateAudio(){

//The while statements determine what sound gets played when pressing a button.

while ((digitalRead(2) == LOW) ) { //Sinewave//

return aSin.next(); //Sinewave//

} //Sinewave//

while (digitalRead(4) == LOW) { //Line\_Gliss//

aTriangle.setPhaseInc(aGliss.next()); //Line\_Gliss//

return aTriangle.next(); //Line\_Gliss//

} //Line\_Gliss//

while (digitalRead(7) == LOW) { //PWM\_Phasing//

char asig1 = (char)(aPhasor1.next()>>24); //PWM\_Phasing//

char asig2 = (char)(aPhasor2.next()>>24); //PWM\_Phasing//

return ((int)asig1-asig2)/2; //PWM\_Phasing//

} //PWM\_Phasing//

while (digitalRead(8) == LOW) { //AMsynth//

long mod = (128u+ aModulator.next()) \* ((byte)128+ aModDepth.next()); //AMsynth//

int out = (mod \* aCarrier.next())>>16; //AMsynth//

return out; //AMsynth//

} //AMsynth//

while (digitalRead(12) == LOW) { //ReverbTank//

int synth = bCarrier.phMod((int)bModulator.next()\*(150u+aModWidth.next())); //ReverbTank//

synth \*= (byte)aEnvelop.next(); //ReverbTank//

synth >>= 8; //ReverbTank//

int arev = reverb.next(synth); //ReverbTank//

return synth + (arev>>3); //ReverbTank//

} //ReverbTank//

return 0; //General//

}

void loop(){

audioHook(); //All// // This just ouptuts the sounds out of digital pin 9.

}

Non-Mozzi Code

#include <Servo.h>    //This prepares the Arduino so that it can control a servo.

int IRVal;               //This is what we will save the value from the IR sensor as.

int servoVal = 180;      //This is the initial position of the servo.  It will later be

                          changed by IRVal.

Servo servo1;            //More setup for the servo.

int var;                 //This is what we will save the value from the slider as.

int noteON = 144;        //MIDI command.

int noteOFF = 128;       //MIDI command.

void setup() {          //This just prepares the board for how we will be using the pins.

 Serial.begin(31250);

 pinMode(2,INPUT);

 pinMode(4,INPUT);

 pinMode(7,INPUT);

 pinMode(8,INPUT);

 pinMode(9,OUTPUT);

 pinMode(10,OUTPUT);

 pinMode(11,OUTPUT);

 pinMode(13,INPUT);

}

void loop() {

 servo1.attach(3);        //More servo setup.

 IRVal = analogRead(A2);  //IR sensor setup.

 servo1.write(servoVal);  //Changes position of servo.

 //This next two blocks determine how to change servoVal based on IRVal.

 if (IRVal < 350 && IRVal > 100){

   if (servoVal > 30){

     servoVal = servoVal - 1;

     delay(50);

   }

 }

 if (IRVal < 600 && IRVal > 350){

   if (servoVal < 180){

     servoVal = servoVal + 1;

     delay(50);

   }

 }

 //The next blocks change which color gets lit and which MIDI note to send.

 if (digitalRead(2) == LOW){

   analogWrite(9,0);

   analogWrite(10,0);

   analogWrite(11,0);

   analogWrite(random(9,12),255);

   var = analogRead(A1);

   var = map(var,0,1023,-39,39);

   MIDImessage(noteON, 60 + var, 127);

   delay(300);

   MIDImessage(noteOFF, 0, 0);

 }

 if (digitalRead(4) == LOW){

   analogWrite(9,0);

   analogWrite(10,0);

   analogWrite(11,0);

   analogWrite(random(9,12),255);

   var = analogRead(A1);

   var = map(var,0,1023,-39,39);

   MIDImessage(noteON, 62 + var, 127);

   delay(300);

   MIDImessage(noteOFF, 0, 0);

 }

 if (digitalRead(7) == LOW){

   analogWrite(9,0);

   analogWrite(10,0);

   analogWrite(11,0);

   analogWrite(random(9,12),255);

   var = analogRead(A1);

   var = map(var,0,1023,-39,39);

   MIDImessage(noteON, 64 + var, 127);

   delay(300);

   MIDImessage(noteOFF, 0, 0);

 }

 if (digitalRead(8) == LOW){

   analogWrite(9,0);

   analogWrite(10,0);

   analogWrite(11,0);

   analogWrite(random(9,12),255);

   var = analogRead(A1);

   var = map(var,0,1023,-39,39);

   MIDImessage(noteON, 65 + var, 127);

   delay(300);

   MIDImessage(noteOFF, 0, 0);

 }

 if (digitalRead(12) == LOW){

   analogWrite(9,0);

   analogWrite(10,0);

   analogWrite(11,0);

   analogWrite(random(9,12),255);

   var = analogRead(A1);

   var = map(var,0,1023,-39,39);

   MIDImessage(noteON, 67 + var, 127);

   delay(300);

   MIDImessage(noteOFF, 0, 0);

 }

 if (digitalRead(13) == LOW){

   analogWrite(9,0);

   analogWrite(10,0);

   analogWrite(11,0);

   analogWrite(random(9,12),255);

   var = analogRead(A1);

   var = map(var,0,1023,-39,39);

   MIDImessage(noteON, 60 + var, 127);

   delay(300);

   MIDImessage(noteOFF, 0, 0);

   delay(200);

   MIDImessage(noteON, 62 + var, 127);

   delay(300);

   MIDImessage(noteOFF, 0, 0);

   delay(200);

   MIDImessage(noteON, 64 + var, 127);

   delay(300);

   MIDImessage(noteOFF, 0, 0);

   delay(200);

   MIDImessage(noteON, 65 + var, 127);

   delay(300);

   MIDImessage(noteOFF, 0, 0);

   delay(200);

   MIDImessage(noteON, 67 + var, 127);

   delay(300);

   MIDImessage(noteOFF, 0, 0);

 }

}

//This is how we send the MIDI triggers to the computer.

void MIDImessage(int command, int MIDInote, int MIDIservoVal) {

 Serial.write(command);

 Serial.write(MIDInote);

 Serial.write(MIDIservoVal);

}