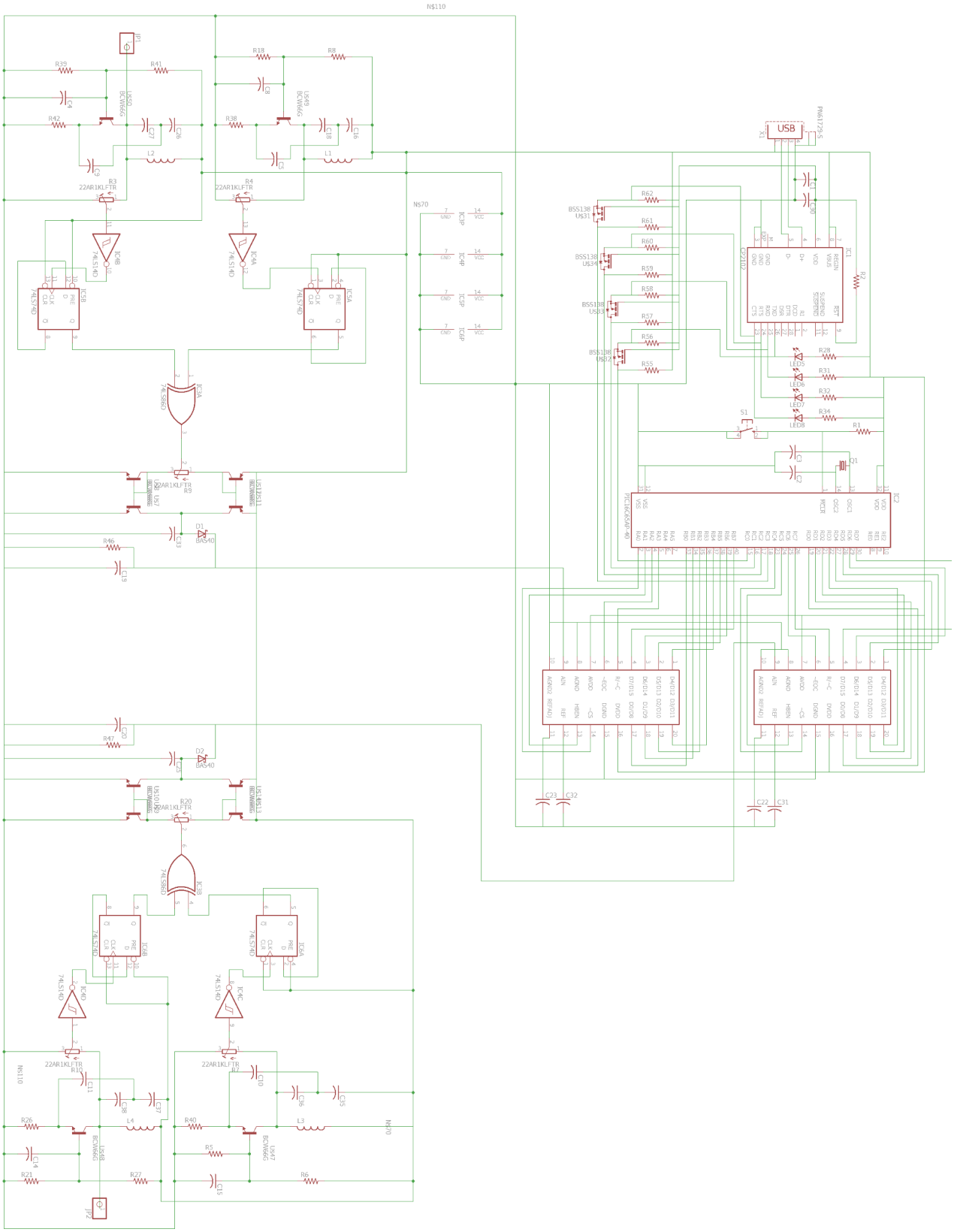
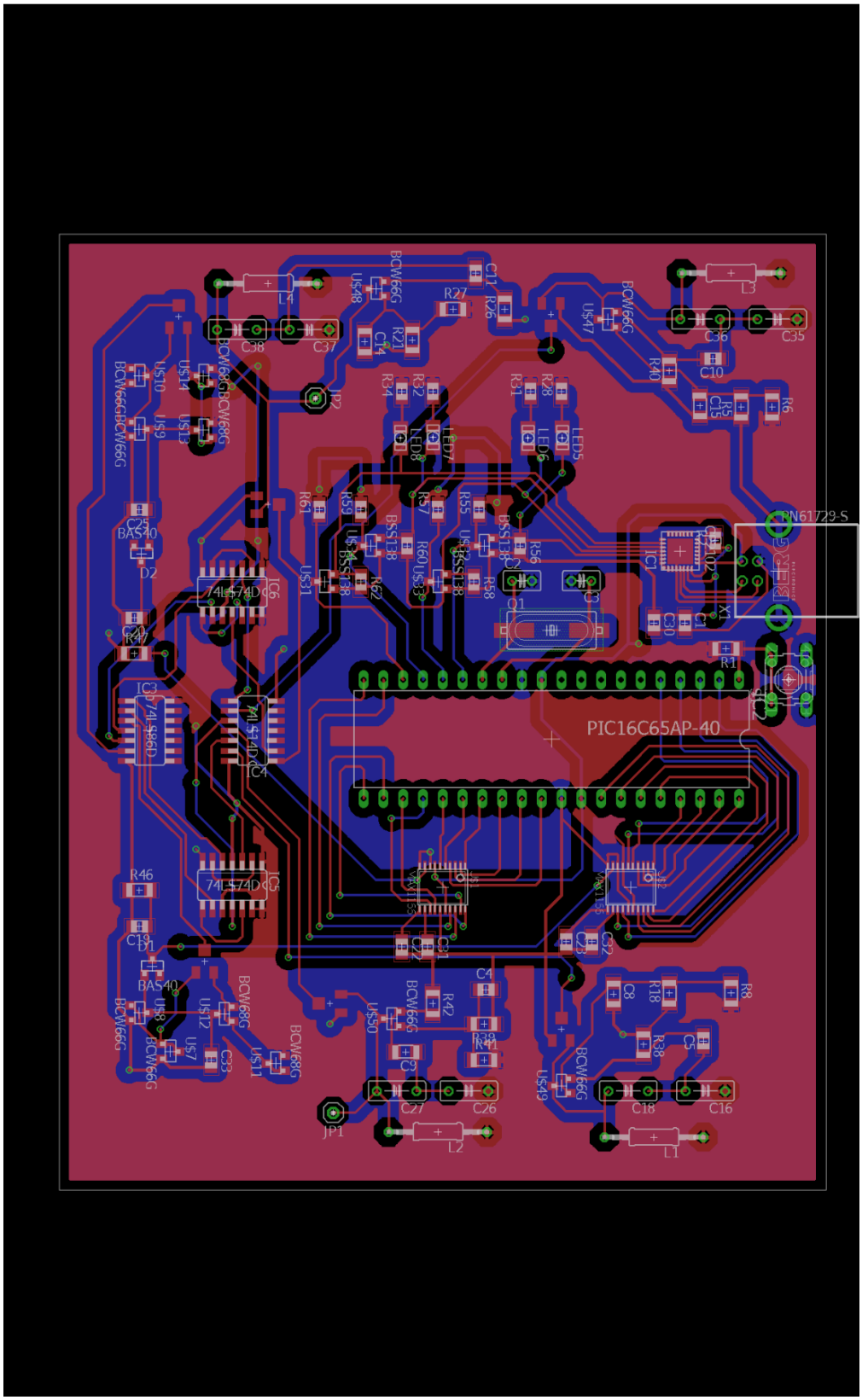
Appendix
Full board schematic:



PCB image:


PIC16C65AP-40
74LS74D
BAS40
BCW66G
BCW68G
BSS138
RN61729-S

Audio engine code:

```
/*  Copyright (C) 2019 Daniel Olivas Hernandez
 *
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.

 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.

 *  You should have received a copy of the GNU General Public License
 *  along with this program.  If not, see <https://www.gnu.org/licenses/>.
 */

#include <stdio.h>
#include <stdlib.h>
#include <vector>

#include <stk/Stk.h>
#include <stk/RtWvOut.h>
#include <stk/SineWave.h>
#include <stk/FileRead.h>

#include <gainput/gainput.h>
#include <X11/Xlib.h>
#include <X11/Xatom.h>
#include <GL/glx.h>

#define MIN(a,b) (((a)<(b))?(a):(b))

enum Button
{
    MouseX,
    MouseY
};

const char* windowName = "frequency mouse control";
const int width = 1600;
const int height = 900;

// values for overtone amplitudes gained through unscientific means
const double attenuation[7] = {1, .3162, .3162, .1, .1, .03162, .03162};
const double normalization[8] = {1, 2, 2.3162, 2.6324, 2.7324, 2.8324, 2.86402,
2.89564};

inline Window setupglx(Display * xDisplay)
{
    static int attributeListDbl[] = {GLX_RGBA, GLX_DOUBLEBUFFER, GLX_RED_SIZE,
        1, GLX_GREEN_SIZE, 1, GLX_BLUE_SIZE, 1, None};
```

```
    Window root = DefaultRootWindow(xDisplay);

    XVisualInfo* vi = glXChooseVisual(xDisplay, DefaultScreen(xDisplay),
attributeListDbl);
    assert(vi);

    GLXContext context = glXCreateContext(xDisplay, vi, 0, GL_TRUE);

    Colormap cmap = XCreateColormap(xDisplay, root, vi->visual, AllocNone);

    XSetWindowAttributes swa;
    swa.colormap = cmap;
    swa.event_mask = ExposureMask
        | KeyPressMask | KeyReleaseMask
        | PointerMotionMask | ButtonPressMask | ButtonReleaseMask;

    Window xWindow = XCreateWindow(
            xDisplay, root,
            0, 0, width, height, 0,
            CopyFromParent, InputOutput,
            CopyFromParent, CWEventMask,
            &swa
            );

    glXMakeCurrent(xDisplay, xWindow, context);

    XSetWindowAttributes xattr;
    xattr.override_redirect = False;
    XChangeWindowAttributes(xDisplay, xWindow, CWOverrideRedirect, &xattr);

    XMapWindow(xDisplay, xWindow);
    XStoreName(xDisplay, xWindow, windowName);
    return xWindow;
}

int main()
{
    // setting up x window
    Display* xDisplay = XOpenDisplay(0);
    if (xDisplay == 0)
    {
        std::cerr << "Cannot connect to X server." << std::endl;
        return -1;
    }
    Window xWindow = setupglx(xDisplay);

    // setting up gainput
    gainput::InputManager manager;
    const gainput::DeviceId mouseId =
manager.CreateDevice<gainput::InputDeviceMouse>();

    gainput::InputMap map(manager);
```

```
    // add way to use MouseX and MouseY
    map.MapFloat(MouseX, mouseId, gainput::MouseAxisX);
    map.MapFloat(MouseY, mouseId, gainput::MouseAxisY);

    manager.SetDisplaySize(width, height);

    double freq = 110.0;
    double vol = 1.0;
    stk::Stk::setSampleRate(44100.0);
    stk::RtWvOut output(2, stk::Stk::sampleRate(), 5, stk::RT_BUFFER_SIZE, 20);

    stk::SineWave fundamental;
    // primitive low pass filter that removes any overtones with frequency > 20 kHz
    std::vector<stk::SineWave> overtones (MIN(20000 / freq,7));
    fundamental.setFrequency(freq);
    for (long unsigned int i = 0; i < overtones.size(); i++)
        overtones[i].setFrequency(freq * (i + 2));

    for (int i = 0; i < 1000000; i++)
    {
        // gainput update shit
        manager.Update();
        XEvent event;
        while (XPending(xDisplay))
        {
            XNextEvent(xDisplay, &event);
            manager.HandleEvent(event);
        }
        // check mouse state
        if (map.GetFloatDelta(MouseX) != 0.0)
        {
            // get mouse position
            double new_freq = freq + map.GetFloat(MouseX) * 1600;
            fundamental.setFrequency(new_freq);
            for (long unsigned int j = 0; j < overtones.size(); j++)
                overtones[j].setFrequency(new_freq * (j + 2));
        }
        if (map.GetFloatDelta(MouseY) != 0.0)
            vol = map.GetFloat(MouseY);
        double amplitude = fundamental.tick();
        for (long unsigned int j = 0; j < overtones.size(); j++)
            amplitude += overtones[j].tick() * attenuation[j];
        amplitude /= normalization[overtones.size()];
        amplitude *= vol;
        output.tick(amplitude);
    }

    // clear x shit
    XDestroyWindow(xDisplay, xWindow);
    XCloseDisplay(xDisplay);

    return 0;
}
```

Device driver code:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdint.h>

#include <iostream>
#include <chrono>

int main(int argc, char * argv[])
{
    if (argc != 2)
    {
        printf("usage: ./test [device file path]\n");
        return -1;
    }

    int fd = open(argv[1], O_RDWR);

    if (fd == -1)
    {
        printf("file open failed\n");
        return -1;
    }

    uint8_t highByteR = 0; // top ADC, right side of PCB
    uint8_t lowByteR = 0;
    double voltageR = 0.0;
    uint8_t highByteL = 0; // bottom ADC, left side of PCB
    uint8_t lowByteL = 0;
    double voltageL = 0.0;
    uint8_t dummy = 10; // dummy write to init

    // time measurement shit
    auto start = std::chrono::steady_clock::now();

    write(fd, &dummy, 1); // init ADC read by transmitting byte
    read(fd, &lowByteR, 1);
    read(fd, &highByteR, 1);
    read(fd, &lowByteL, 1);
    read(fd, &highByteL, 1);

    // other time point
    auto end = std::chrono::steady_clock::now();
    std::cout << "transmission time: " <<
           std::chrono::duration_cast<std::chrono::microseconds>(end -
start).count() << std::endl;

    voltageR = (double)highByteR * 256 + (double)lowByteR;
    voltageR *= (4.096 / 65535.0);
```

```
    voltageL = (double)highByteL * 256 + (double)lowByteL;
    voltageL *= (4.096 / 65535.0);

    printf("voltage level from top ADC, right of PCB: %f\n", voltageR);
    printf("voltage level from bottom ADC, left of PCB: %f\n", voltageL);

    close(fd);

    return 0;
}
```