**Initial Architecture Document**

**Team Number:** 23

**Team Members:**

Dominic D’Attilio, Alexander Wilhelm, Andrew Brown, Cole Stinson, Gage Burmaster

**Project Name:**

Prysm

**Project Synopsis:**

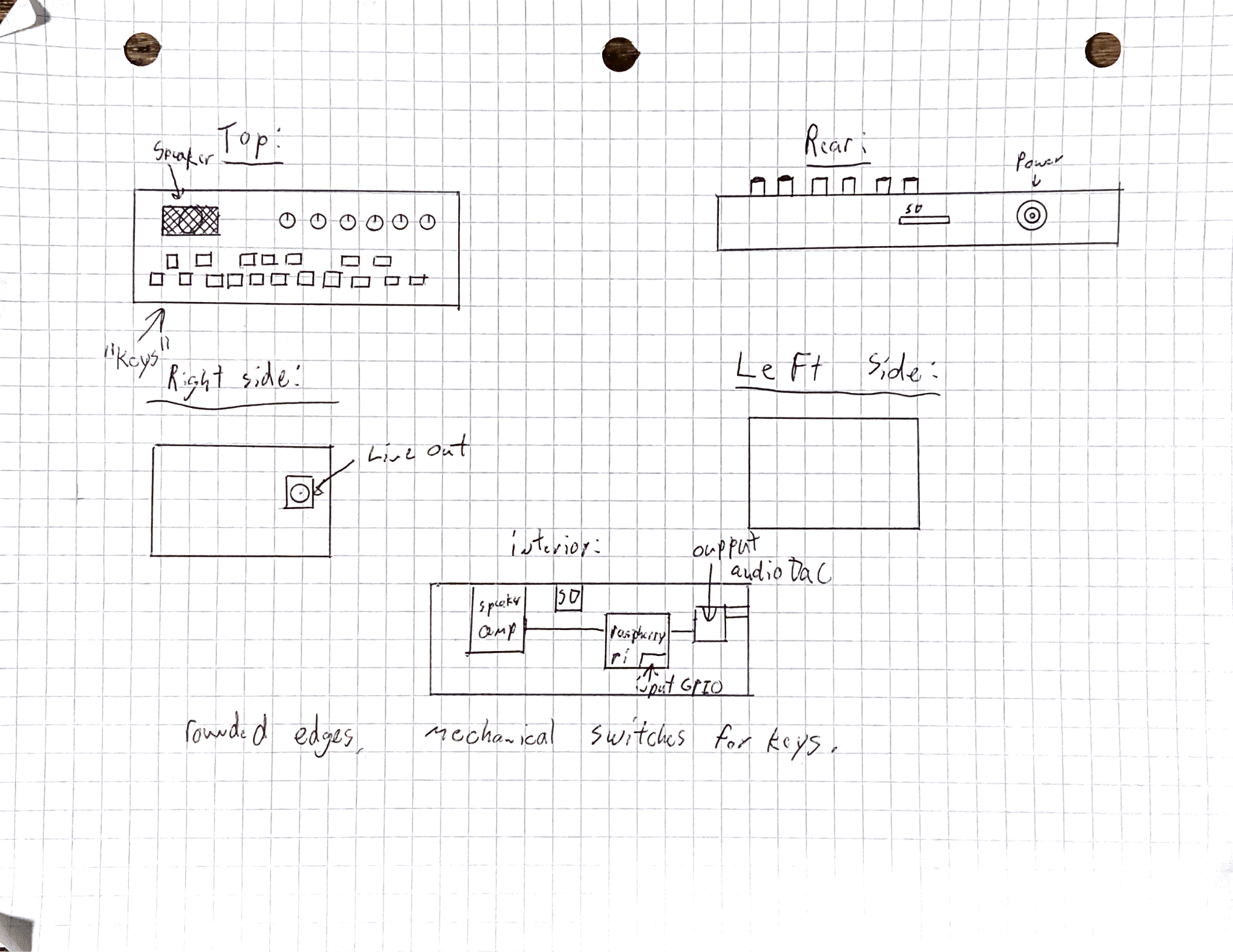
A hardware synthesizer with a companion desktop application that allows you to change the synth wave table and upload it to your synthesizer.

**Project Description:**

Our project, Prysm, will be a hardware synthesizer with customizable sounds using wavetable synthesis. (to be continued) Our method of wave table synthesis will allow for the oscillation of waves and morphing of sounds to allow for a more natural sound from the synthesizer. The wavetable generator (the companion desktop app) will load the generated tables onto an SD and allow our synthesizer to read these and generate sounds to the users liking. We decided to use an SD card for transfer because it is small, easily accessible, and allows us to store multiple wave tables on the device. Instead of using a flash drive or portable usb drive. The SD card will be stored safely inside the device so the user cannot snap it off or knock the reader out of place. We thought that using a wired connection, ie USB or Micro USB, from a computer to the main board may be an option but storing it on an external device for quick access seemed to be the better option. If we stored the wave table on the main board, the idea was to just use the single wave table only, and to change that table, the user needed to upload a new table from their computer. One of the main contributors for this decision was that the user could store multiple wave tables and easily delete and manage the files on their computer’s native file exploring system. Being able to store multiple wave tables at one time opens up new doors and allows for expansions.

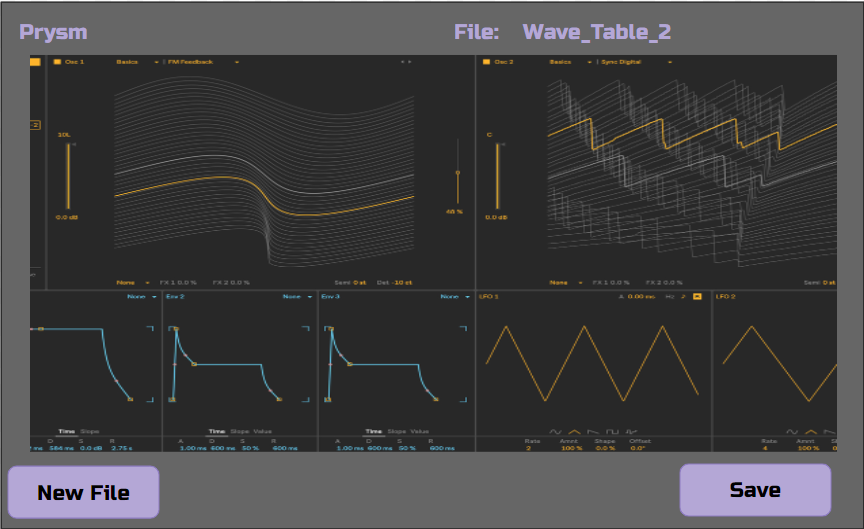
The synthesizer will be built using a Raspberry Pi 4, as well as additional expansion cards to help with sound production and to better the user experience. Using a Raspberry Pi 4 allows us to program our synthesizer in C++. Our decision of C++ originated from the idea of latency because to have an accurate machine that mimics an instrument, there can be little to no latency from when a button is pushed to a sound made. In addition to our Raspberry Pi 4, we will be using multiple expansion cards to allow for faster processing and more flexibility. Our plan is to dedicate an expansion card to record and capture different key presses and relay that information back to the main board. We have decided that using mechanical key switches will achieve the best press and feel for our model. These will resemble the ones that are used for keyboards on computers. An issue of latency does arise, but we decided it was best to allow for more keys with the expansion and add a minimal amount of latency rather than limit ourselves on the amount of buttons we can add to the Raspberry Pi 4. This will allow us to simplify how we program individual parts of the board, allowing for each part to have its own domain. The keys expansion card will funnel all traffic from the keys seamlessly to the main board, mimicking a keyboard, and allow the board to not have to worry about if a key is being pressed, it will focus on the important features. In addition to the keys on the synthesizer, we will also have a knob to control volume and a 3.5mm audio line out so that the user can connect the synthesizer to an external speaker or amplifier.

In addition to the hardware, we will have a companion desktop app to allow users to edit and customize the synth wavetables. By customizing the wavetables, the user will be able to change the sounds played when he plays the synthesizer. Built using the Vue.JS desktop framework, the app will allow users to customize the wavetables and upload them to an SD card that can be transferred to the synthesizer to then load and play the new sounds. Doing this allows more freedom for the user to fully customize their experience while using our synthesizer. We decided on doing a desktop app because it is the easiest way to both manually edit the wavetables, and transfer these wavetables to the SD card to be read by the Raspberry Pi 4. We decided to use SD cards to transfer the files because it gives the user the ability to load multiple wave tables at once onto the SD card from their computer and then have all the wave tables on the synthesizer. We plan to allow the user to be able to manually swap between the different wave tables that they have stored on the SD card from the synthesizer itself, so the user can use the synthesizer away from their computer.



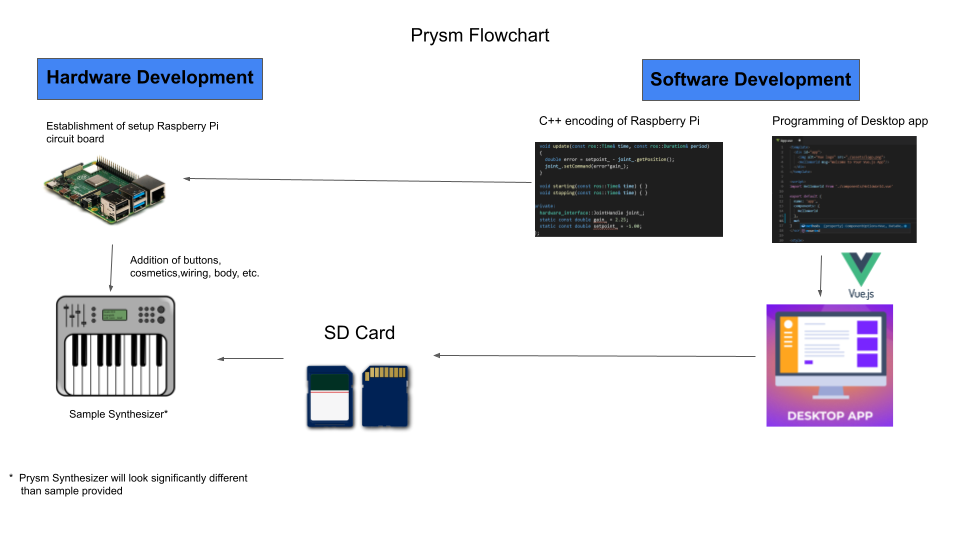
(Fig. 1: Hand-drawn schematic of synthesizer)

This figure shows the initial schematic of our synthesizer. You can see the various pieces of hardware we will have, the keys (buttons), control knobs, and ports for power, audio line out, and the port for the SD card. We wanted the board to resemble a piano, but we also wanted the compact functionality of a sound board. The knobs will also have various functions including volume, reverb, and a few other features to help produce the perfect sound.



(Fig. 2: Prototype of desktop application)

This figure shows an initial prototype of how the desktop app would look and work. You can see the file name you are editing, as well as the various wave tables where you will be able to customize the varying aspects of the wave table. You can clearly see the button where you save your work to the SD card. There will also be a button to create a new file, allowing the user a fresh start to the wave table creation process. The app will be created with Vue.js, allowing for a fast and easy setup.



(Fig. 3: Flowchart of project)

Figure 3 shows the flowchart of how Prysm development will work. It illustrates that we will use a desktop app built in Vue.js to load an SD card with files generated by the user. The SD card will be read by the Raspberry Pi 4 and they will create the synthesizer sound.