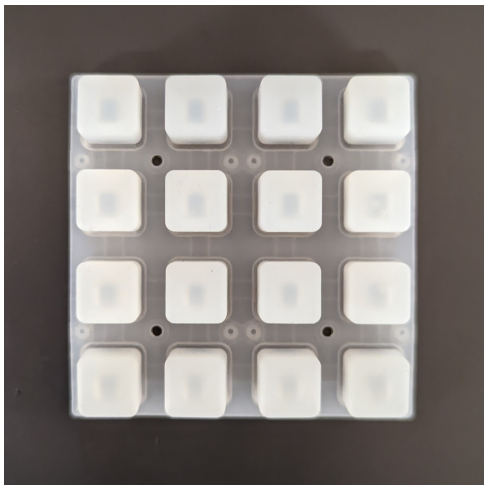
Week 7

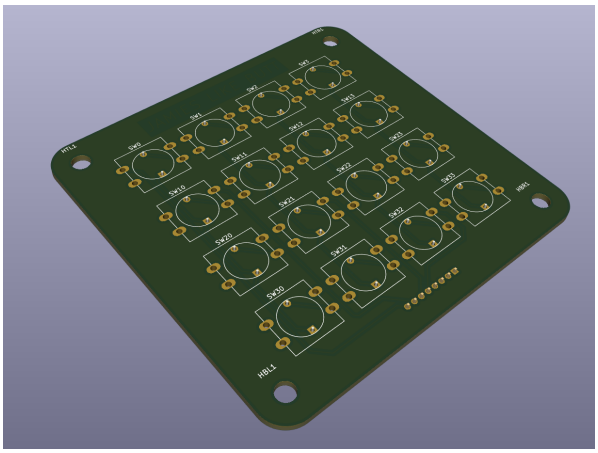
This week

Hardware Design:

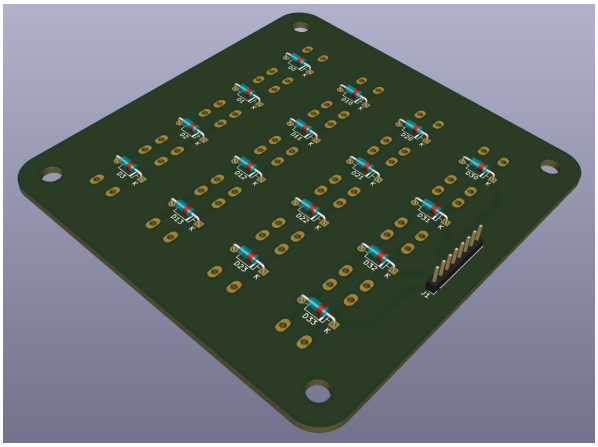
In order to use the Sparkfun button matrix PCB mentioned in previous progress reports, we would have to order extra proprietary parts from Sparkfun directly. In order to circumvent this, I quickly design a button matrix PCB to work with an existing silicone cover that we discovered in the lab.



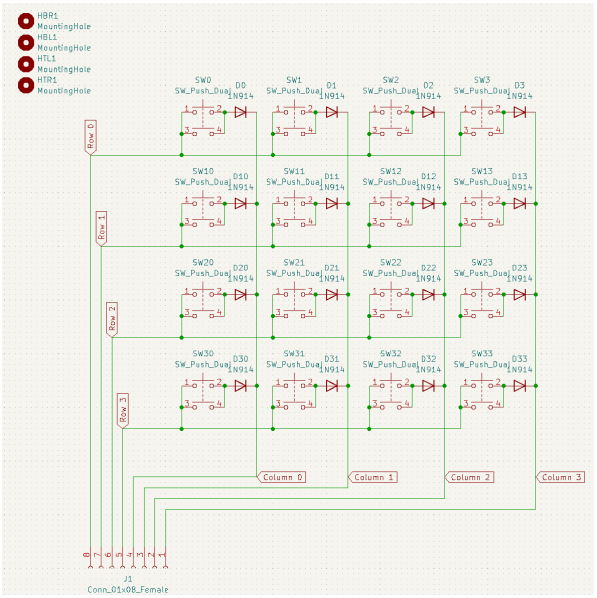
This meant measuring the silicone cover and design the PCB to fit accurately with it. The silicone pad interfaces with 16 12mm push buttons, so we used that footprint in our design:



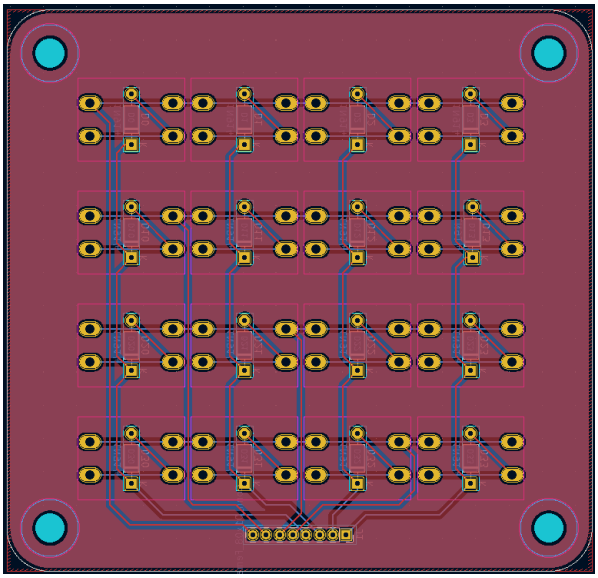
Front Side of Matrix



Back Side of Matrix



Schematic



Routes

It is a little excessive to design our own button matrix for this project, but it will allow us to reliably eliminate ghosting, and gave me a little practice using KiCad.

Software:

This week I upgraded the `UartDecoder` class to make it simpler to get information from the MCU. `UartDecoder::readSerial()` now sends a “request” for the UART message. In reality, this is just a simple 0x01 that gets sent so the MCU knows to send data. An example of how to use this object is below:

```
#ifdef TESTDECODER
// TEST MAIN FOR DEBUGGING
int main(int argc, char ** argv){
  string deviceStr = "/dev/ttyUSB0";
  UartDecoder uart = UartDecoder(deviceStr);
    if(uart.serial_port == 0){
        cout << "Problem Setting Up Serial Port" << endl;
        return 1;
    }
    int messagei = 0;
    for(;;){
        uart.readSerial();
        printf("New Message %d\n",messagei);
        printf("Current Button: %d\n", uart.getButton());
        printf("Current Bounce: %d\n", uart.getBounce());
        messagei++;
    }
    return 0;
}
#endif
```

Additionally, this week for software I started to prototype a `Projector` class to abstract the drawing functions called by the game loop. The first library I tried to use was OpenCV purely because we already included it in our build. This involved using `cv::imshow()` to show a matrix in a popup window, and modifying the `cv::mat` object when drawing to the screen. I created simple initialization functions and a square drawing function. The declaration is as such:

```
void Projector::renderSquare(int x, int y, int w, int h, float r, float g, float b)
```

With r, g and b being the color values of the box. The box is also initialized as such:

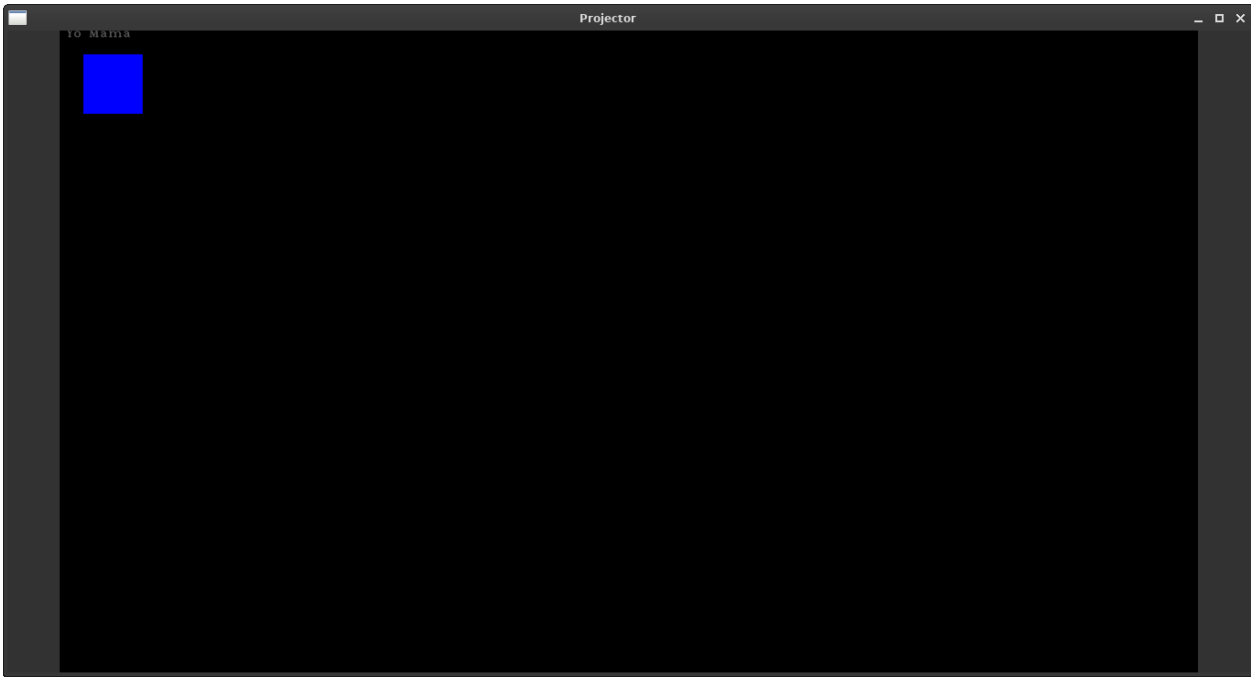
```
Projector::Projector(int h, int w)
```

With the color of the mat being set to black automatically. Below is an example call to `Projector` and the output of the window.

```
#define TEST_PROJECTOR
#ifdef TEST_PROJECTOR
#include "UartDecoder.hpp"
#include <string>
```

```
int main(int argc, char ** argv){
  Projector proj(1080,1920);
  std::string uartDeviceStr = "/dev/ttyUSB0";
  UartDecoder uart(uartDeviceStr);
  proj.renderSquare(40,40,100,100,255,0,0);
  std::string text = "Yo Mama";
  proj.writeText(text,1,10,10,100,100,100);
  proj.redraw();
  cv::waitKey();

}
#endif
```



`renderSquare` automatically cuts off squares that are not in bounds of the window. This will allow us to show simple graphics through the projector based on messages sent from the MCU.

`writeText` will write text to the screen as well cutting off at the edges

Next Week

Hardware:

This coming week we will order our first PCB, and our button matrix PCB. We will assemble and test it as soon as it arrives.

Software:

The UART decoder class is finished, so for next week I will be working on the Projector class and Ball Detector class. The ball detection has taken a little bit of a back seat the past two weeks as we are focusing on hardware and getting a working prototype from table to projector. But, as we finalize designs and wait for PCB shipments we will have more leeway to start working on ball detection again.

For now we will continue to use OpenCV for displaying graphics in the Projector Class. The next two things that I need to test, are putting text in a Mat and displaying a window in full screen. These two features are important for creating a fully fledged UI displayed on the table. If OpenCV does not offer all the features we want we will try another graphics library such as Xlib.

I will first try to lay the framework for the ball detector class by creating some simple declarations detailing the functionality we need. Then I will attempt to implement contour detection, similar to what we did with the depth camera. The ball detector class will continuously update a local x and y variable that details the location of the ball. This could also be a buffer, if we wanted to get a rough idea of the direction the ball is traveling.

The main Game Logic class will call all other classes. The game logic class will create two threads: one for handling ball detection, and one for everything else. The latter will poll the MCU for data at a predefined rate, read a ball position from the camera, update game states, and display to the projector every game loop. Ball detection will be running simultaneously, always collecting ball position data. This means that the position variables set by the Ball Detector class, will have to be atomic, so we don’t run into any collisions if two threads try to access the same memory location.

Presentation:

With our midterm presentation approaching, we will be detailing the design decisions made throughout the entire PCB. We will incorporate the comments from the presentation into our design, and iterate. A lot of next week will be spent preparing for and practicing this presentation.