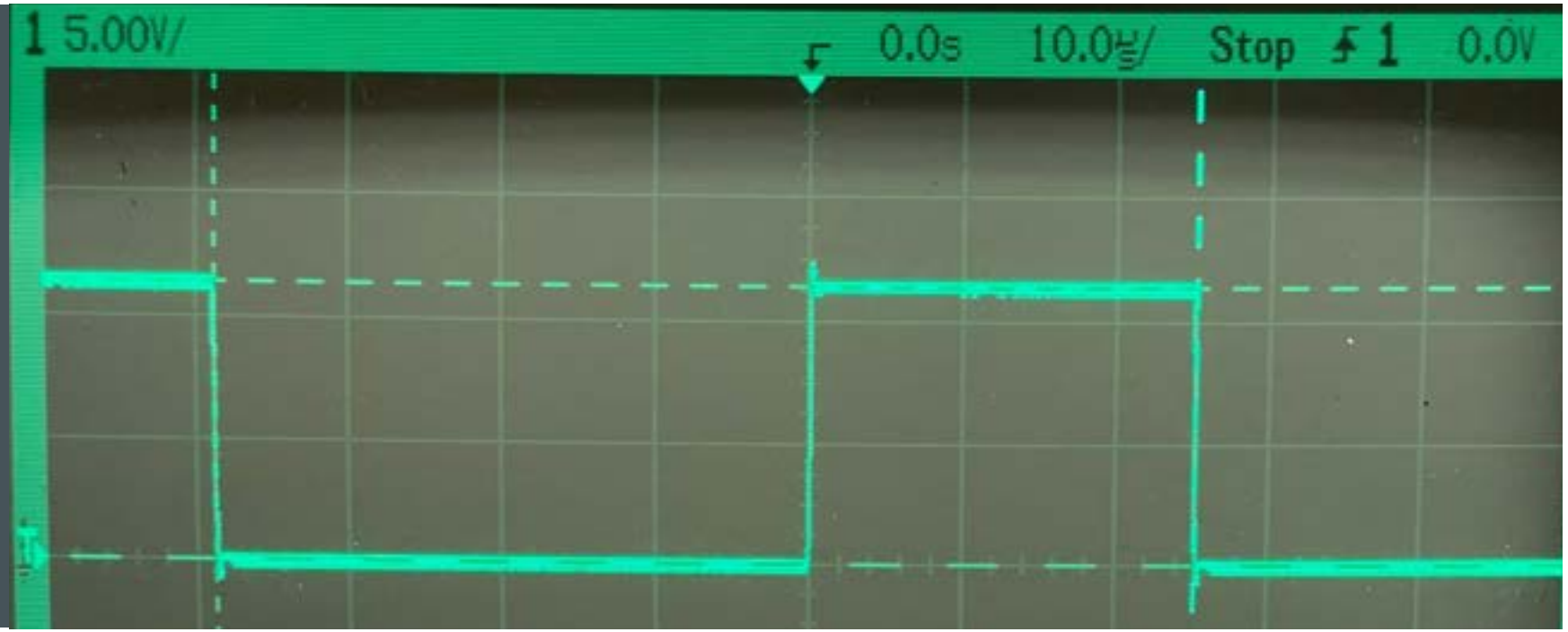
RAHEEL REHMATULLAH

PROJECT PORTFOLIO

October 2023

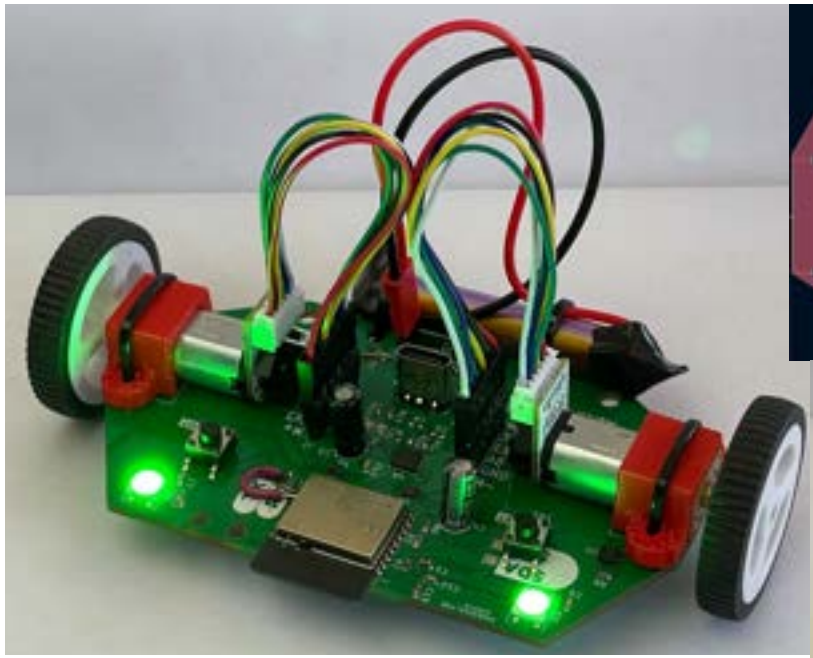


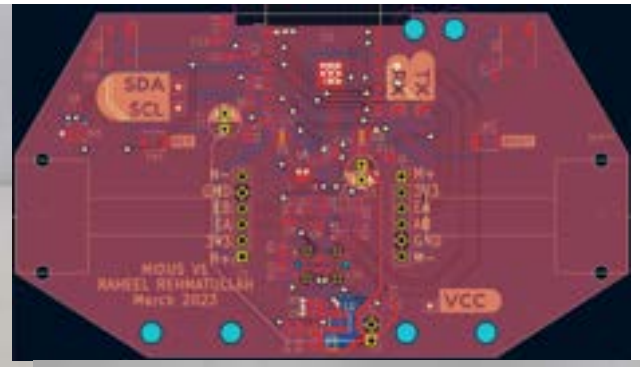
M1-E 'DEMONSTRATOR' (May 2023) SLO PROPULSION TECHNOLOGY

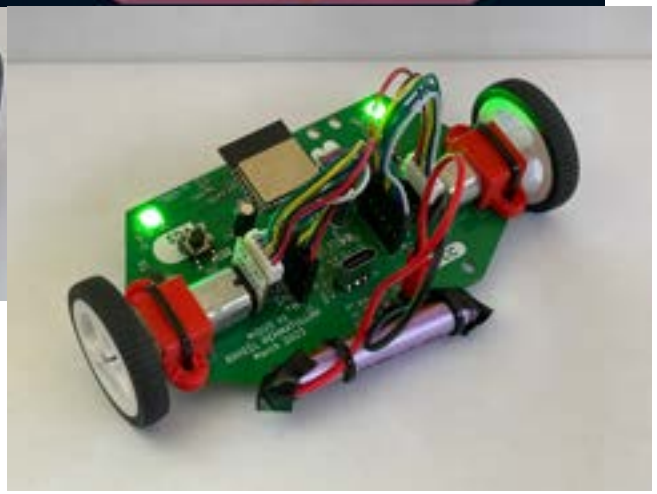
- Developed harness
- Designed, built, and assembled control panel
- Tested Data Acquisition system
- Iterated several ignition techniques
- Lots and lots of trouble shooting
- The first Bi-propellant rocket engine to be fired by a student club at Cal Poly SLO!
- Total burn time of 12 seconds to date

MIDUS (APRIL 2023) PERSONAL PROJECT

- Mini robot
- Based of ESP32-C3 with RISCV core
- Features two RGB LEDs for indicating
- Custom buck converter for one cell LiPo
- Six-Axis IMU for localicalization and control
- Support for an encoder on each wheel for finer control
- Designed in KiCad

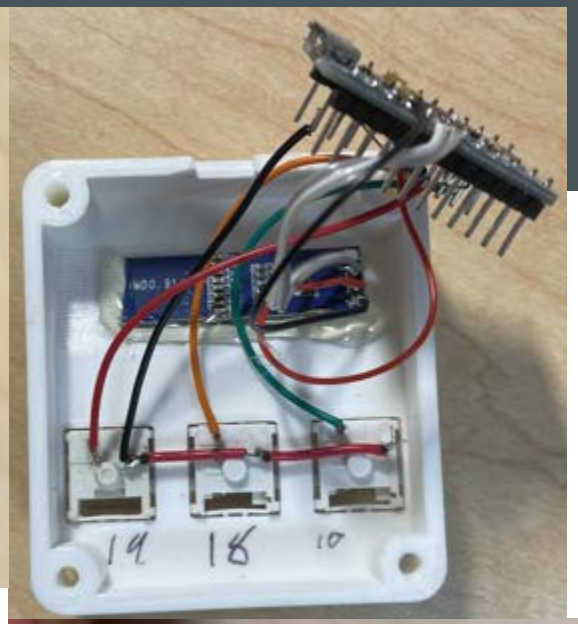


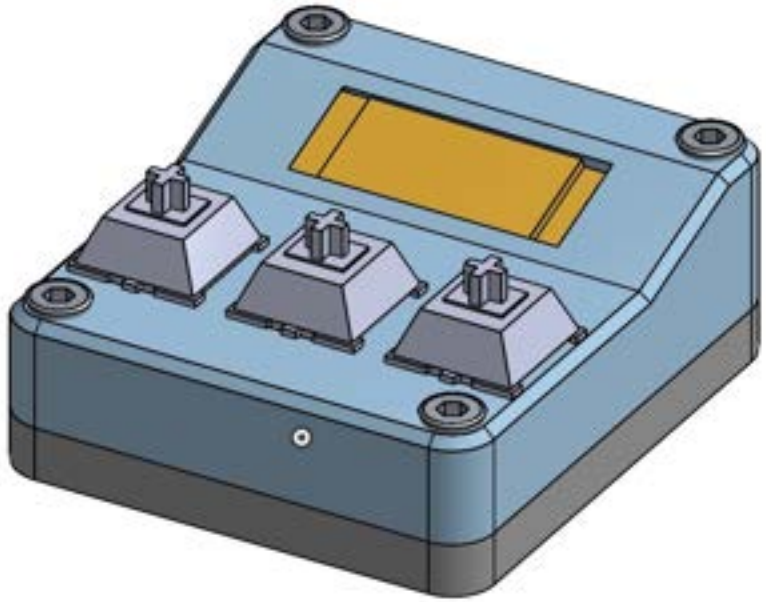




Binary Keyboard (March 2023)
PERSONAL PROJECT

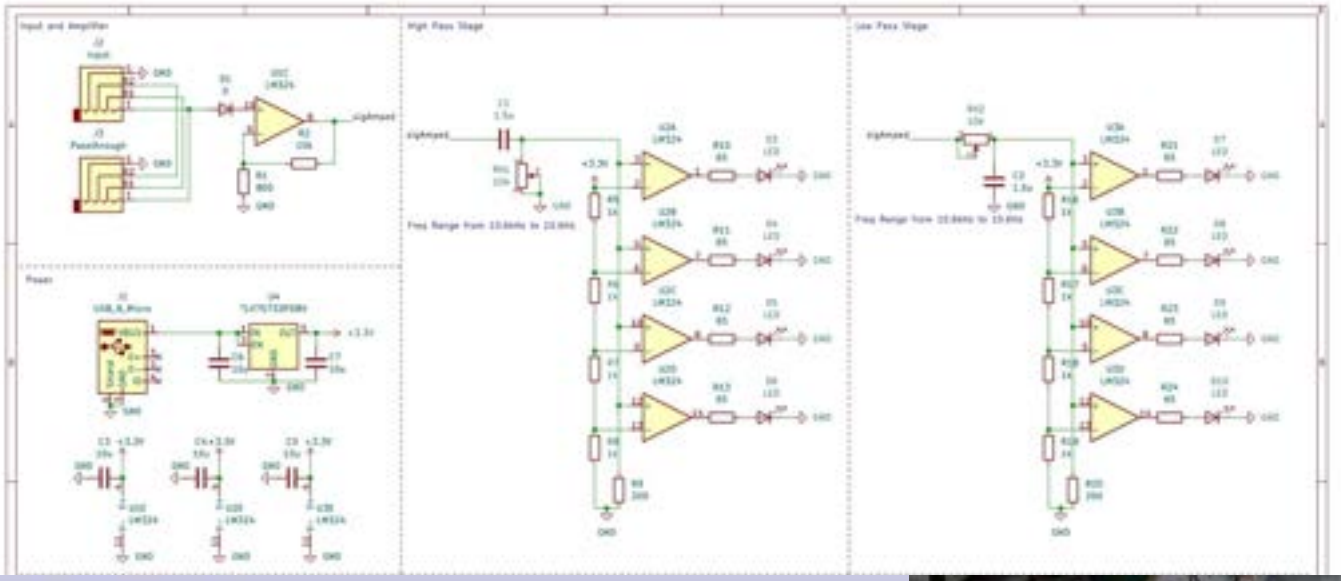
- Have you ever wished your keyboard was a little simpler
  - (and a lot less convenient?)
- Designed enclosure in Onshape
- Based on an Arduino Pro Micro
- Hand Soldered pullup resistors
- Self written custom firmware
- Scan to see in action!

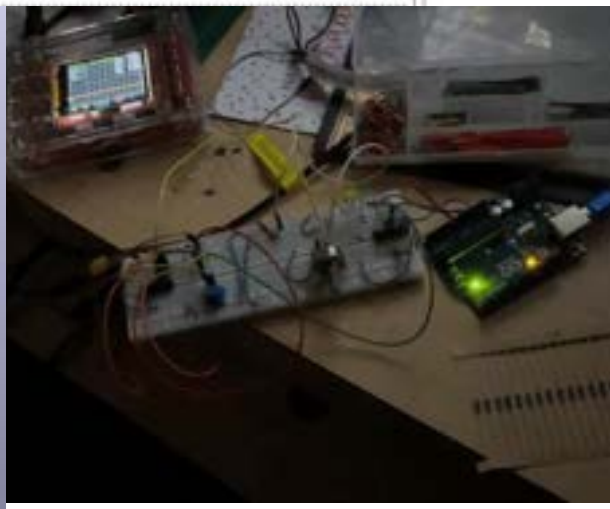




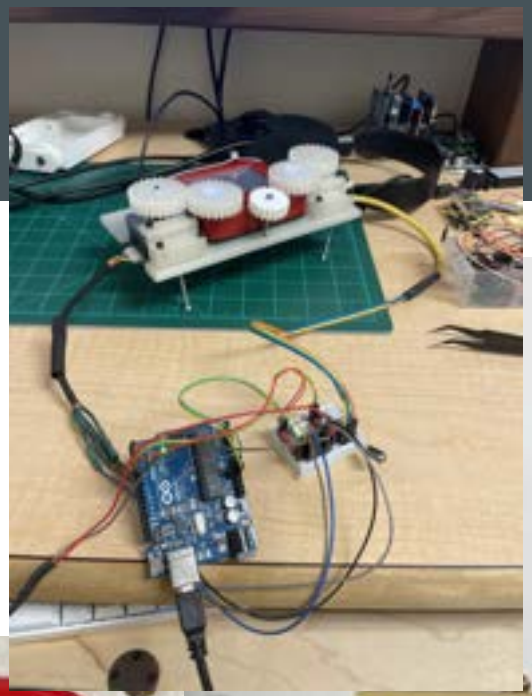
VUCARD (February 2023) PERSONAL PROJECT

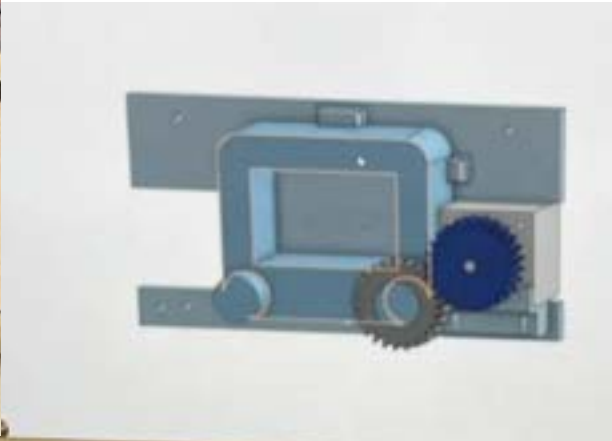
- Business cards!
- Headphone jack audio split between a low and high pass filter
- LED's light up depending on audio wave amplitude and filtering
- Simulated design using Falstad
- Designed in KiCad
- Click or scan to read about my design process





CNC ETCH-a-SKETCH (February 2023) PERSONAL PROJECT

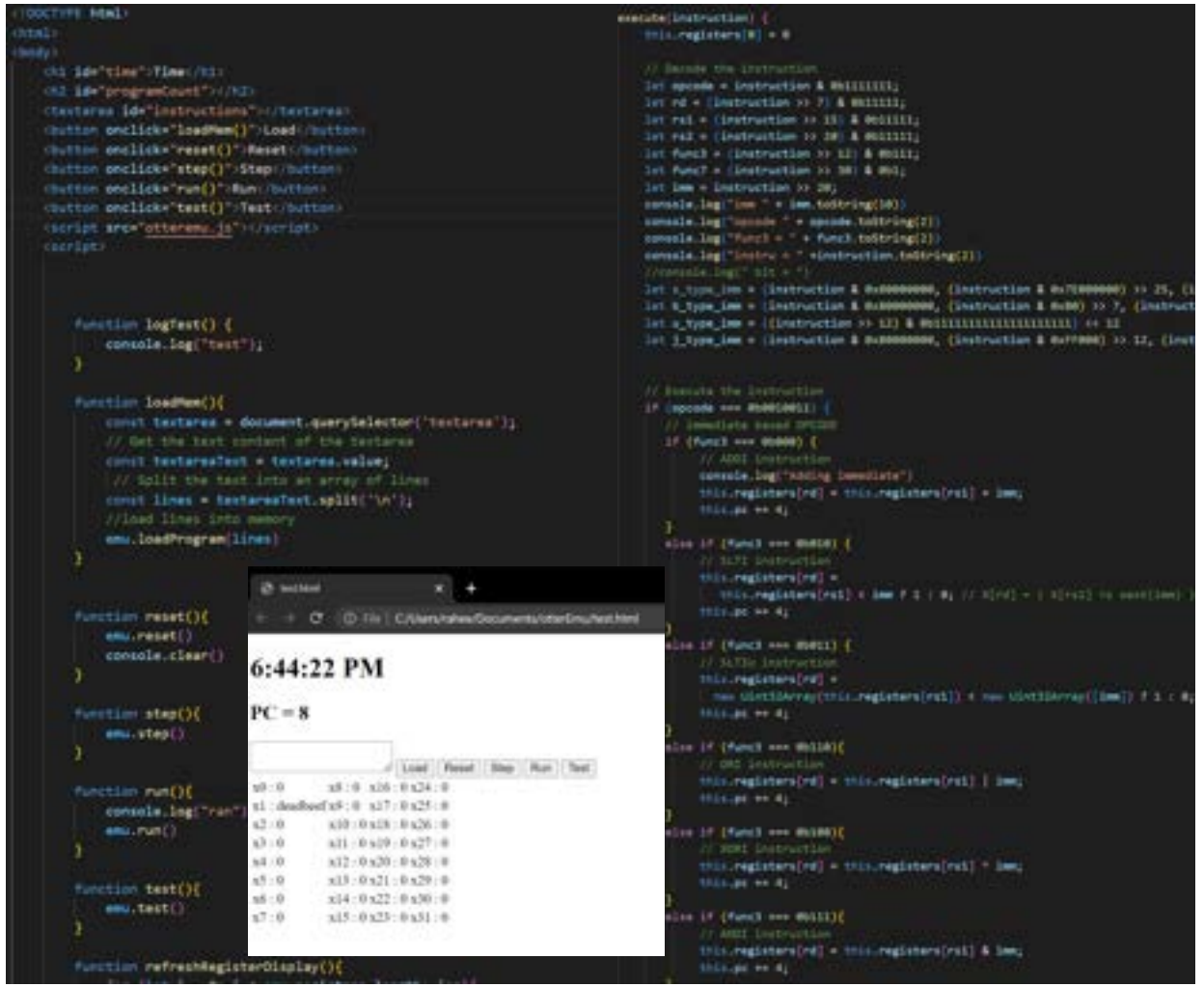




- Built in a weekend
- Designed in Onshape
- Based on GRBL and an Arduino UNO
- Custom herringbone gears
- Printed on Prusa i3 mk3
- Scan to see in action!

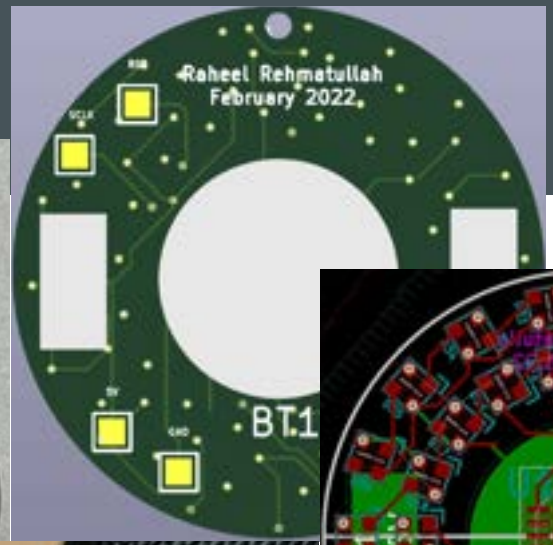
otterEMU (January 2023)
RISC-V Emulator in JS

- 32 Bit data path and addressing
- Based on the RISC-V Otter used in CPE 233 at Cal Poly SLO
- Runs natively anywhere that supports basic HTML + JS
- Winter Break project after completing CPE233 and wanting to learn more about the 32 bit RISC-V ISA



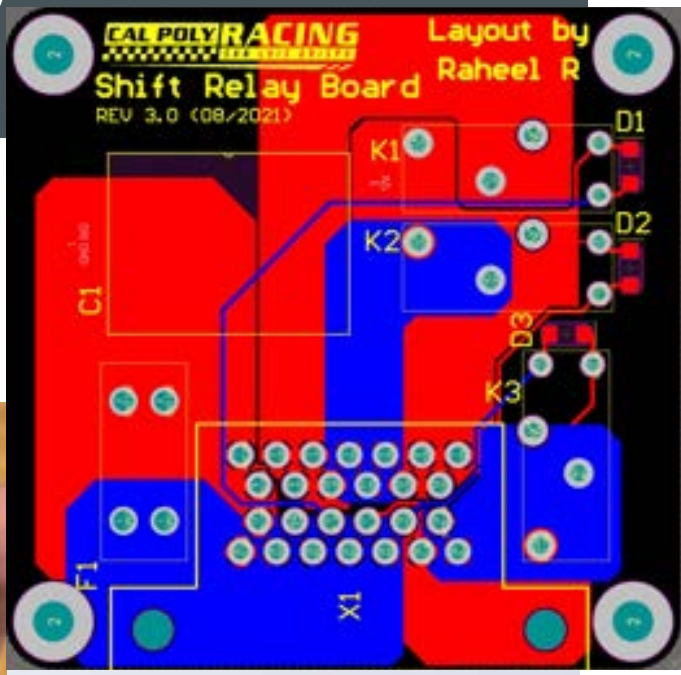
ATTINY EARRINGS V2 (March 2022) PERSONAL PROJECT

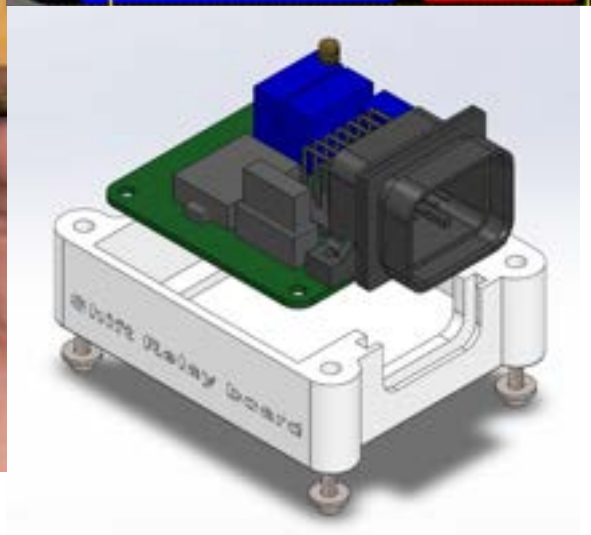
- 36 SK6805Addressable LEDs
- Programmable via ICSP
- Based on Attiny85 Microcontroller
- Hand soldered QFN Microcontroller
  - (JLC was out of stock ☹)
- Powered by CR1216 coin cell
- Designed in KiCad
- Half the size of the V1's!
- 9 times the LED!



SHIFT RELAY BOARD AND ENCLOSURE (August 2021) CAL POLY RACING

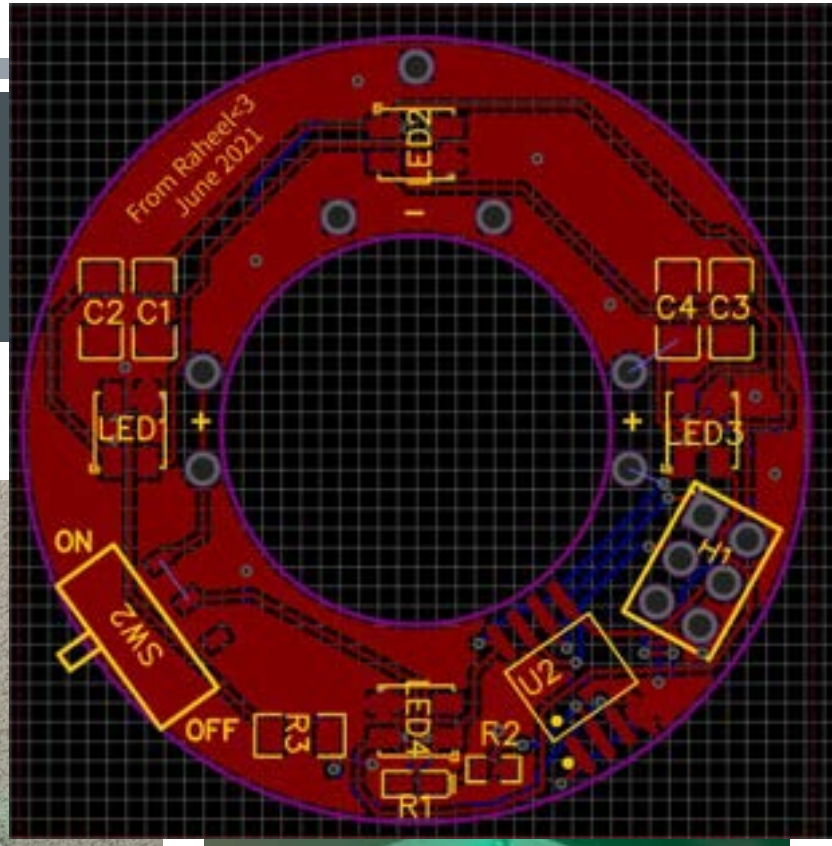
- Designed to regulate high amperage loads across an electromechanical gear shifting actuator
- Redesigned layout features a single connector and reduced weight and volume footprint
- 3d printed enclosure
- Designed using Altium and Solidworks

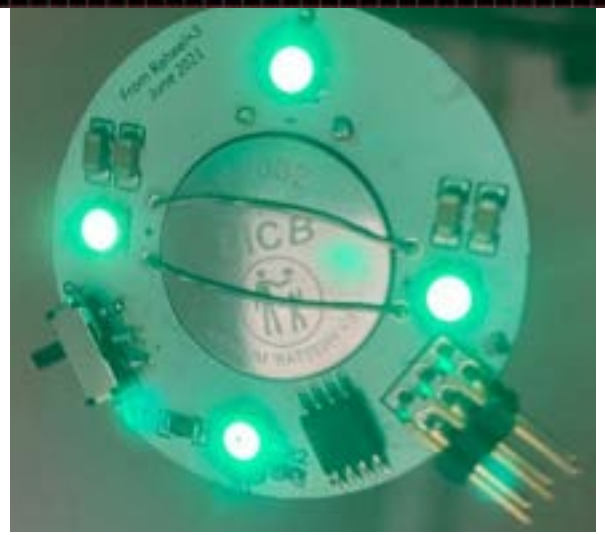




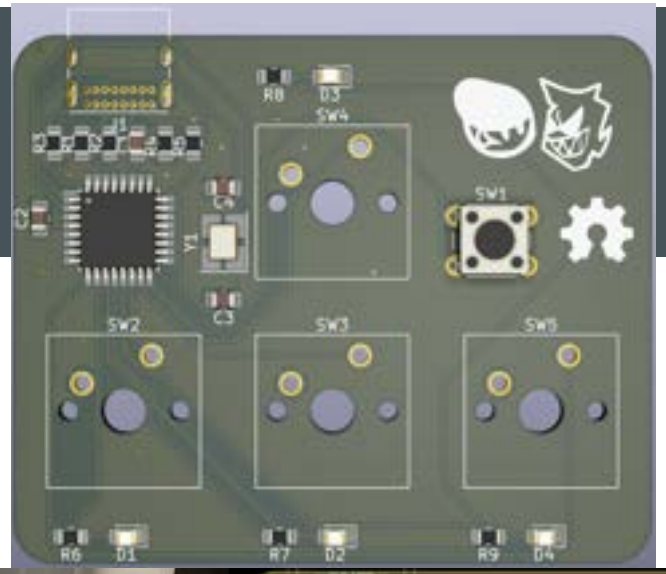
ATTINY EARRINGS (June 2021) PERSON PROJECT

- 4 WS2812B Addressable LEDs
- Programmable via ICSP
- Based on Attiny45 Microcontroller
- Custom CR2032 Battery mounting using PCB geometry
- Designed in easyEDA
- Cute earrings!

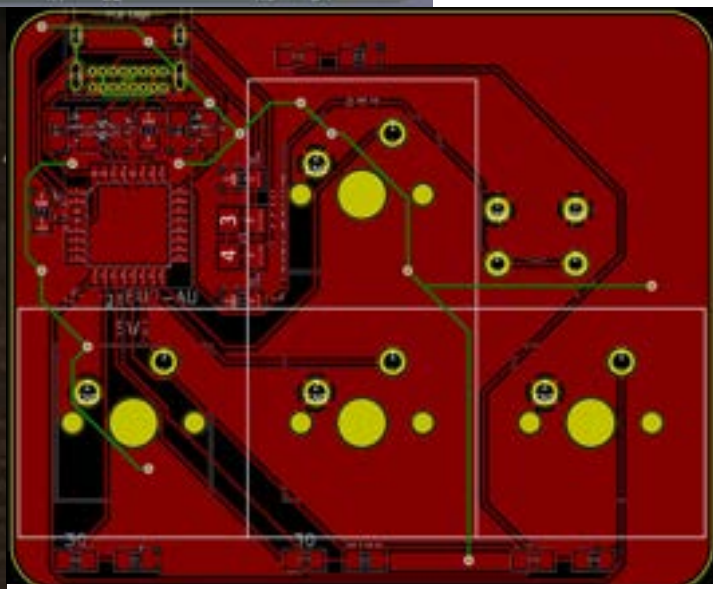




ATMEGA MACROPAD (March 2021) PERSONAL PROJECT



- Designed for use as a simple 4 button game pad
- Made to fit within $15-$20 budget at small scale production
- Based on Atmega8u2 microcontroller
- Designed using KiCad with board manufactured via JLCPCB



SPLITKEEB (April 2021) PERSONAL PROJECT

- Designed as an attempt at a DIY opensource split ~60% keyboard
- 3d printed case designed in OnShape
- PCB designed in KiCad
- Arduino Pro Micro/Atmega32u4 based microcontrollers
- Both sides communicate via I²C bus across a single TRRS cable
- Software based on QMK firmware

INVERSE AND FORWARD KINEMATICS (April 2021) WHITE PAPER

- Written as an exploration of Robotic Kinematics
- Explores inverse and forward kinematics
- Demos robot that features both kinematic models
- Scan to read!

$$\vec{C} = \vec{A} + \vec{B}$$

$$\vec{A} = \sqrt{(2\cos(\theta_0))^2 + (2\sin(\theta_0))^2}$$

$$\vec{B} = \sqrt{(2\cos(\theta_0 + \theta_1))^2 + (2\sin(\theta_0 + \theta_1))^2}$$

$$\vec{C} = \sqrt{(2\cos(\theta_0))^2 + (2\sin(\theta_0))^2} + \sqrt{(2\cos(\theta_0 + \theta_1))^2 + (2\sin(\theta_0 + \theta_1))^2}$$

Using this information, the individual X and Y components of the total vector $\vec{C}$ can be calculated using the cosine and sine functions as follows:

$$C_x = (2\cos(\theta_0)) + (2\cos(\theta_0 + \theta_1))$$

$$C_y = (2\sin(\theta_0)) + (2\sin(\theta_0 + \theta_1))$$

Translating this into Arduino programmable code gives us the following:

Where Θ1display is θ_0 and Θ2display is θ_1. (There is also the discrepancy in sin and cos being swapped functions, this is due to how the Arduino C libraries handle trigonometric functions differently, and the constants of 71/4068 convert the integer degree counts into Arduino readable radians.)

And with that, we can display the exact X and Y coordinate while being able to control both joints! Forward kinematics achieved!

4

In the case of inverse kinematics, we can take the trigonometric ideas that we used in the case of forward kinematics and inverse them. That means that in the case of the example robot, the input is going to be an x and y coordinate, and the robot will need to use inverse kinematics in order to calculate the angles to assign to the joints. A very useful but sometimes (personally) forgotten rule of trigonometry is the law of cosines, or in the case of the robot arm:

$$green^2 = purple^2 + cyan^2 - 2purple * cyan * \cos\theta_1$$

This can also be rearranged to get the angle θ_0 as follows:

$$cyan^2 = purple^2 + green^2 - 2purple * green * \cos\theta_0$$

Both of these equations can be solved for their respective angles, giving us new equations:

$$\theta_1 = \cos^{-1}\frac{purple^2 + cyan^2 - green^2}{2purple * cyan}$$

$$\theta_0 = \cos^{-1}\frac{purple^2 + green^2 - cyan^2}{2purple * green}$$

In the case of the robot, the length of purple and cyan are actually constants with a length of 2 inches, and the green side can be found by finding the magnitude of both the x and y components of the arms position.

$$green = \sqrt{x^2 + y^2}$$

By substituting all variables with their real values, we get the final equations:

$$\theta_1 = \cos^{-1}\left(\frac{-x^2 + y^2}{8}\right)$$

$$\theta_0 = \cos^{-1}\left(\frac{x^2 + y^2}{4\sqrt{x^2 + y^2}}\right)$$

This math translates into code as shown

```
theta1 = acos((sq(x)+sq(y))/(4*sqrt(sq(x)+sq(y)))) *  4068.0 / 71.0 ;
theta2 = acos((-sq(x)+sq(y))/8) *  4068.0 / 71.0;
```