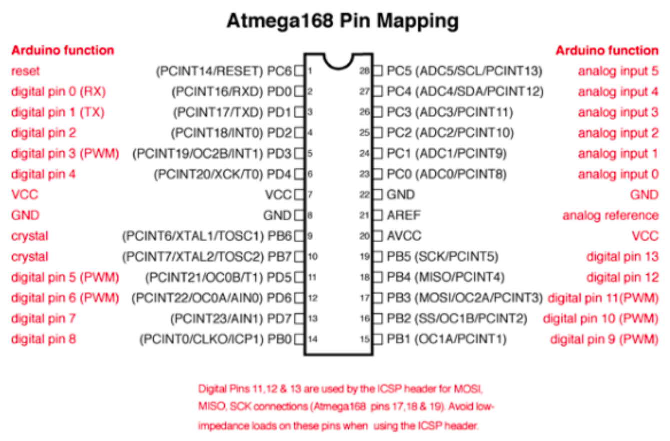
Design Overview

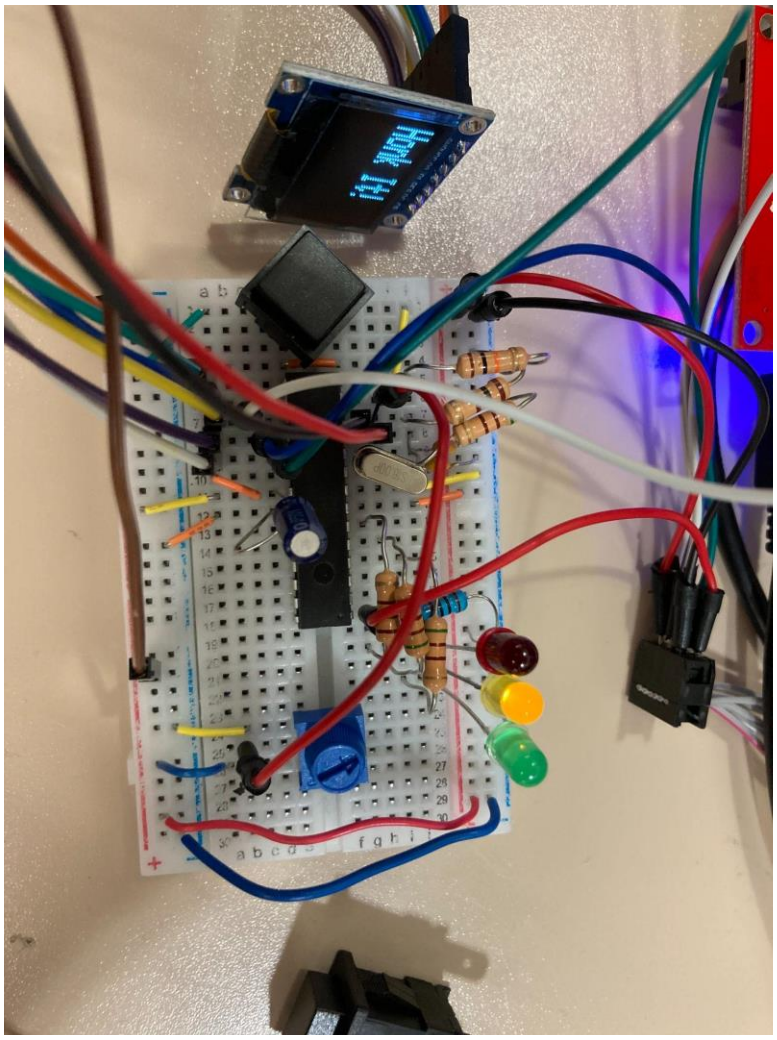
Our bop-it we created is a driving themed game called “Stop It”. We started our design phase by coelessing our individual design ideas into a single game. We took several aspects from each other's initial plans and eventually settled on three simple inputs. Given troubles previously working with PCBs, we decided to keep the inputs to the game straightforward. Likewise, given our simple commands, we decided to use the ATMega32p for our microcontroller. For our first command we have “Honk It”. “Honk it” was designed to simulate honking the horn of a car. We implemented this with a simple momentary switch button that would complete the connection from VCC to GND on one of the input pins. Initially we planned to have a speaker sound as well when the button was pressed, however this was changed later in development. Our second command we designed was “Turn It”. “Turn it” was designed to simulate turning the steering wheel of a car utilizing a potentiometer. This potentiometer would always be driven to an input pin, and when moved from high to low or vice versa, the command would be read. We also planned to 3D print a steering wheel to attach to the potentiometer. The final input we implemented is “Brake It”. “Brake It” was designed to simulate pulling a stick shift of a car to put it into brake utilising a flip/lever switch. The two states of the flip switch (up and down) would have a continuous connection to two input pins on the chip and whichever one was given power would be the input for that round of the game. In addition to these inputs, we implemented a simple power switch to turn on and off the game. For feedback to the player, we used three coloured LEDs to signify game states. A red light indicated a player failure (i.e. wrong input or a timeout). A yellow light indicated waiting for player input. Lastly, a green light indicated a correct player input. These LEDs would coincide with an LCD screen which would give player score and game commands. The same speaker for honk it would be used to give unique auditory queues as well. We used a Circuit Digest tutorial on LCD connections to arduino as a base for our display code (linked below). For our enclosure, we designed to have the PCB fit into a 3D printed case in the shape of a stop sign that consisted of a lid and a container that would screw together. We used a reference image as well to help with assigning pins in code (attached below).


Atmega168 Pin Mapping
Arduino function
Arduino function
reset (PCINT14/RESET) PC6 1 | 28 PC5 (ADC5/SCL/PCINT13) analog input 5
digital pin 0 (RX) (PCINT16/RXD) PD0 2 | 27 PC4 (ADC4/SDA/PCINT12) analog input 4
digital pin 1 (TX) (PCINT17/TXD) PD1 3 | 26 PC3 (ADC3/PCINT11) analog input 3
digital pin 2 (PCINT18/INT0) PD2 4 | 25 PC2 (ADC2/PCINT10) analog input 2
digital pin 3 (PWM) (PCINT19/OC2B/INT1) PD3 5 | 24 PC1 (ADC1/PCINT9) analog input 1
digital pin 4 (PCINT20/XCK/T0) PD4 6 | 23 PC0 (ADC0/PCINT8) analog input 0
VCC VCC 7 | 22 GND GND
GND GND 8 | 21 AREF analog reference
crystal (PCINT6/XTAL1/TOSC1) PB6 9 | 20 AVCC VCC
crystal (PCINT7/XTAL2/TOSC2) PB7 10 | 19 PB5 (SCK/PCINT5) digital pin 13
digital pin 5 (PWM) (PCINT21/OC0B/T1) PD5 11 | 18 PB4 (MISO/PCINT4) digital pin 12
digital pin 6 (PWM) (PCINT22/OC0A/AIN0) PD6 12 | 17 PB3 (MOSI/OC2A/PCINT3) digital pin 11(PWM)
digital pin 7 (PCINT23/AIN1) PD7 13 | 16 PB2 (SS/OC1B/PCINT2) digital pin 10 (PWM)
digital pin 8 (PCINT0/CLKO/ICP1) PB0 14 | 15 PB1 (OC1A/PCINT1) digital pin 9 (PWM)
Digital Pins 11,12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

https://circuitdigest.com/microcontroller-projects/arduino-ssd1306-oled-display

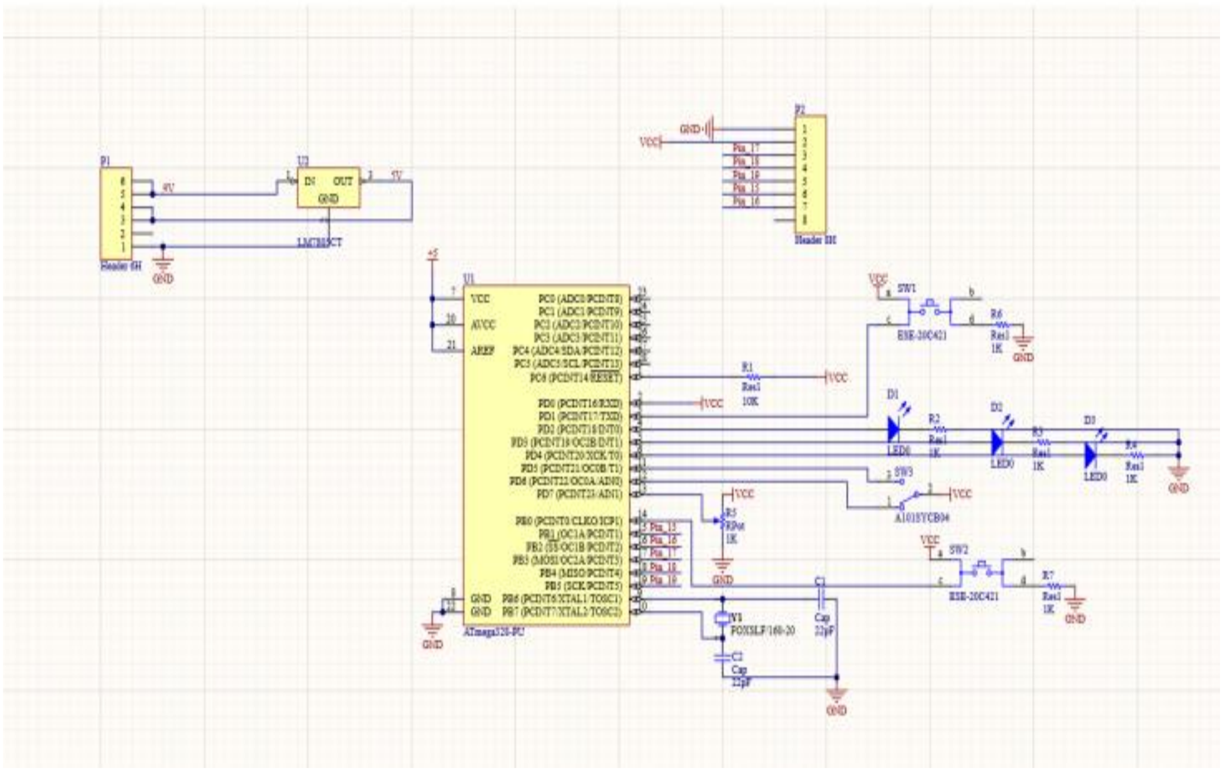
Design Verification

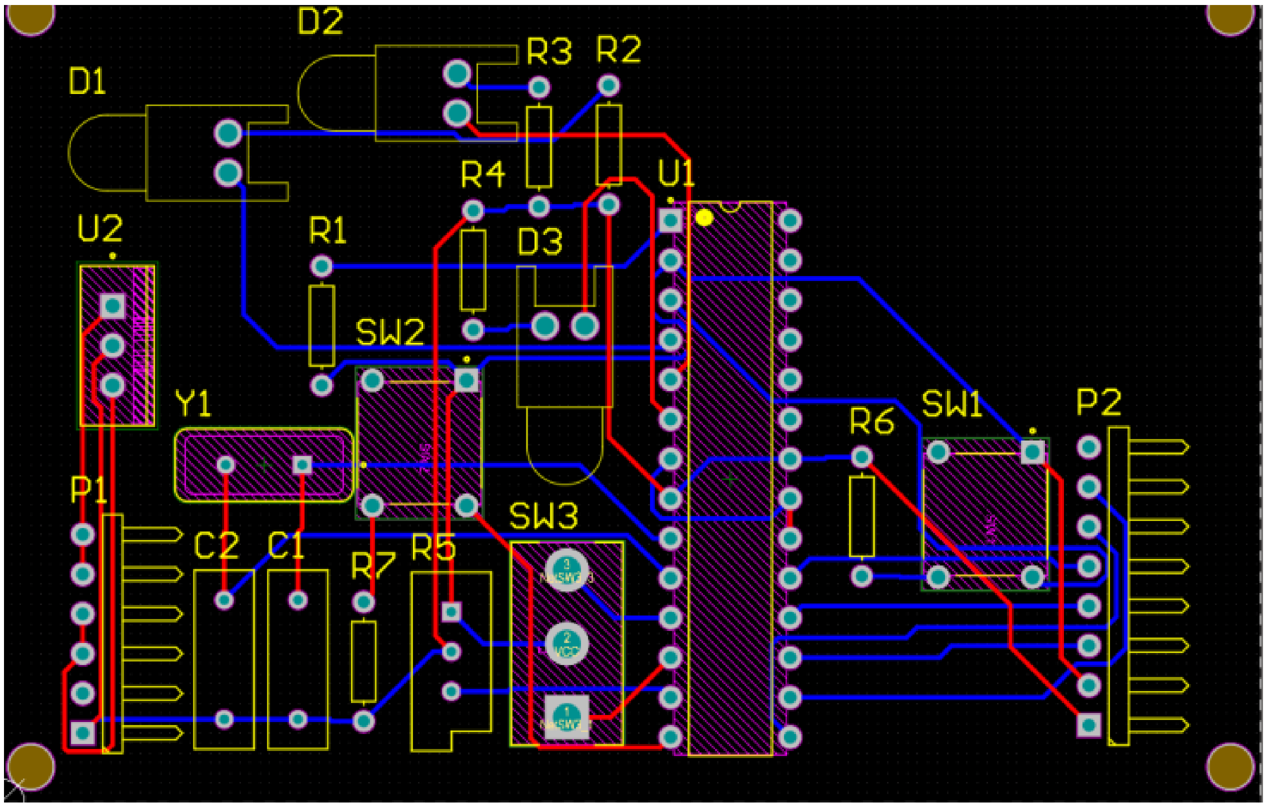
For our verification we completed a full breadboard of our design. We used the same components in the final design in order to get the closest possible simulation. This was done in tandem with the creation of the PCB. Doing this highlighted some issues with our initial design, such as suboptimal resistor placement and rethinking how we would have our on/off switch work. These were corrected when assembling our PCB, but would have been better if we had done a full verification first. These tests were done with the chip attached to a bootloader so we could test code changes on the fly as well. Attached below are pictures of the breadboard. During our final initial testing phases, we were using sound queues in tandem with the LCD screen utilizing a speaker. This speaker also made the horn noise for honk it. We had to scrap this aspect of our design as we kept blowing out speakers by accident and ultimately ran out of time to implement it properly in the PCB. Given our simple commands, we did not need to use LTSpice to simulate the components as they were simple boolean high/low inputs/outputs. Below is an image of our first working breadboard.

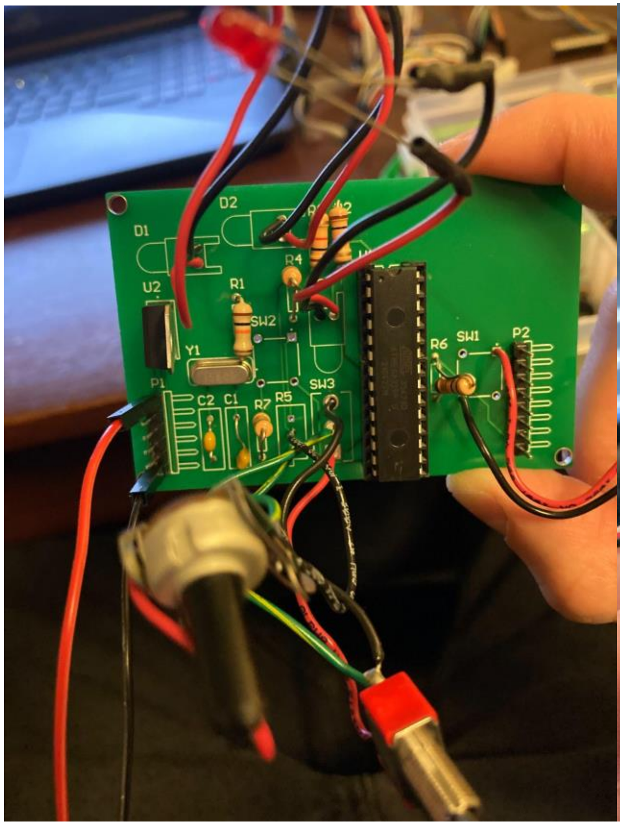


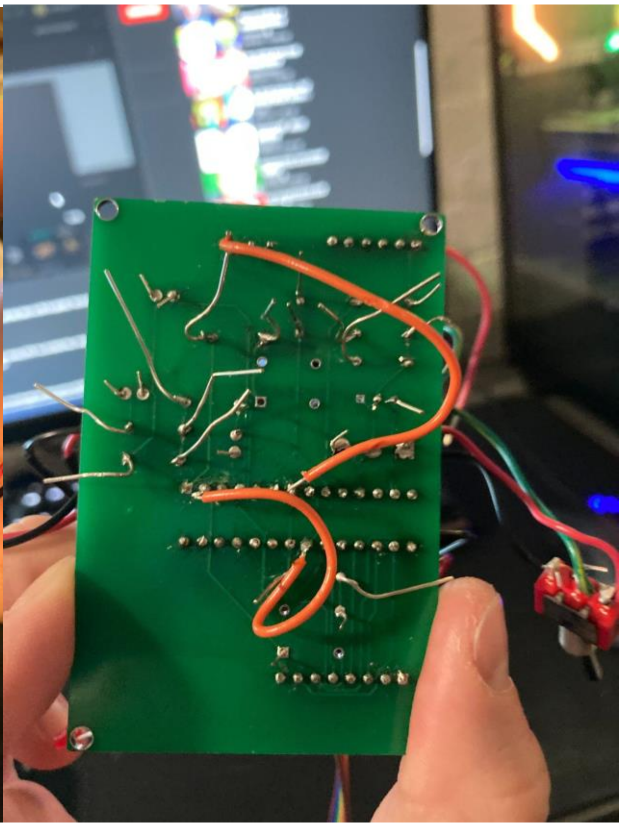
Electronic Design Implementation

Our PCB was designed to be as compact as possible while still leaving room for needed adjustments. For best practices, we used all through hole components since our verification was not complete and would facilitate changes easier. We kept the actual layout of the board as logical as possible. Input devices were put near their respective pins, as well as the LEDs and LCD pins. We left space on the board for headers in order to easily connect/disconnect the LCD screen and power pins. One issue we ran into was not actually connecting our regulated voltage pin to the chip, this was due to poor naming conventions with the nodes. This problem was remedied by hard wiring the regulated voltage pin to the voltage pin on the on/off switch. After fixing the power issue we had an issue with some of the inputs on the board due to aforementioned suboptimal resistor placements. This was easily fixed by hardwiring components and resistors using wires. Lastly, four mounting holes were included at each corner of the PCB to attach to our enclosure. Despite our challenges, the PCB ultimately was mended through some ingenuity.



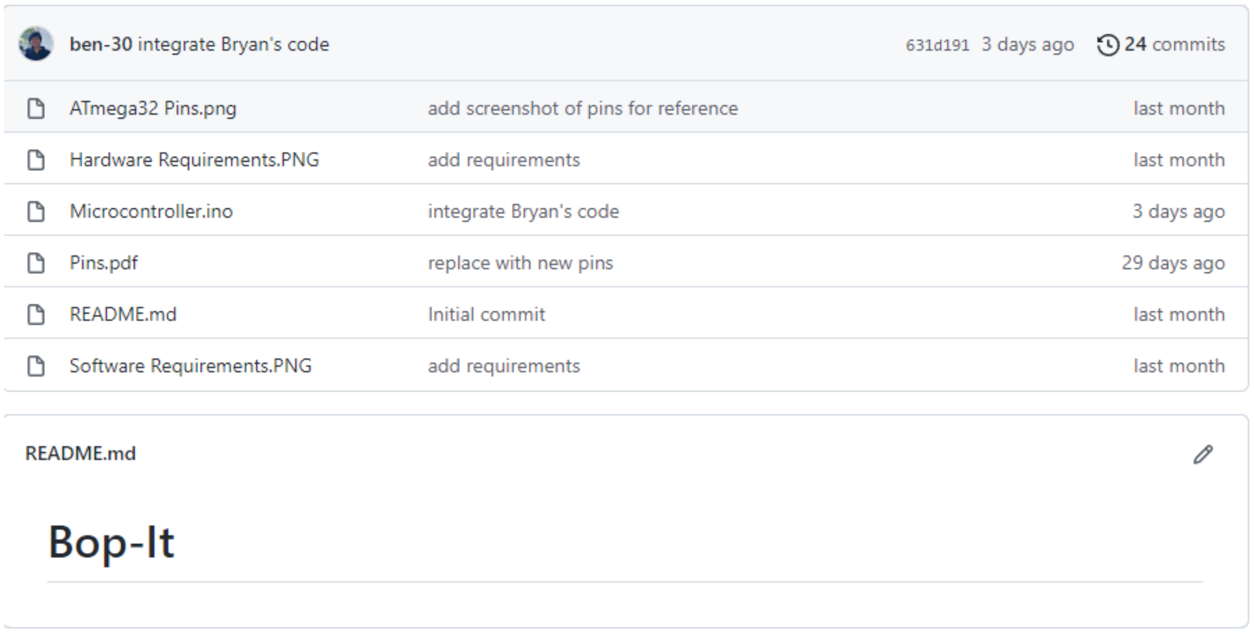

D1
D2
R3
R2
R4
U1
U2
R1
D3
SW2
Y1
SW1
P2
R6
P1
SW3
C2
C1
R7
R5





Software Design Implementation

Our software development process was the longest of the bunch as we used software to fix many of our minor hardware issues. At a high level, our code runs in an infinite loop when supplied with power that runs different states depending on user input. The main loop starts a new game at the first instance of power, and at the end of a “game over” or “game win”. Our main generates a timer and a random number between 1 and 3 which signifies one of the three commands. A while loop is then used to loop from 0 to the end of the timer with a delay at the end of each loop. Within the loop, all three of our input functions are called. All three of the functions work similarly, they all sense a change in any of the input pins and return true if that input was changed. Once one of the functions returns true, the loop breaks and a switch case is run. If only the correct input is called, then the continue game condition is met. If the wrong input was entered, or no input was entered at all, then the game over condition is met. If the score reaches 100, then the game win condition is met. The last two functions we have are to keep track of what state the potentiometer and flick switch are in. These are called at the beginning of each loop. If the potentiometer is read low, then the previous turn value is stored as 0. This is how we detect changes in the potentiometer, and similarly, the flick switch. If the player enters the correct input, then the timer decrements and the score is raised by 1. If there is a gamer over, the score is displayed to the user, then the timer and score are reset and the game restarts. If the score reaches 100, then the game ends with a “you win” screen displaying the score, then restarts the game again.


ben-30 integrate Bryan's code 631d191 3 days ago 24 commits

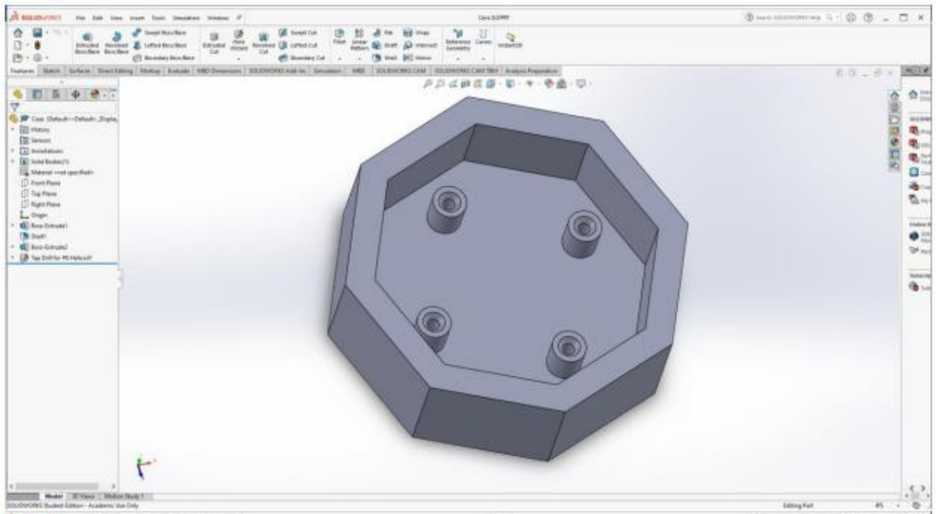
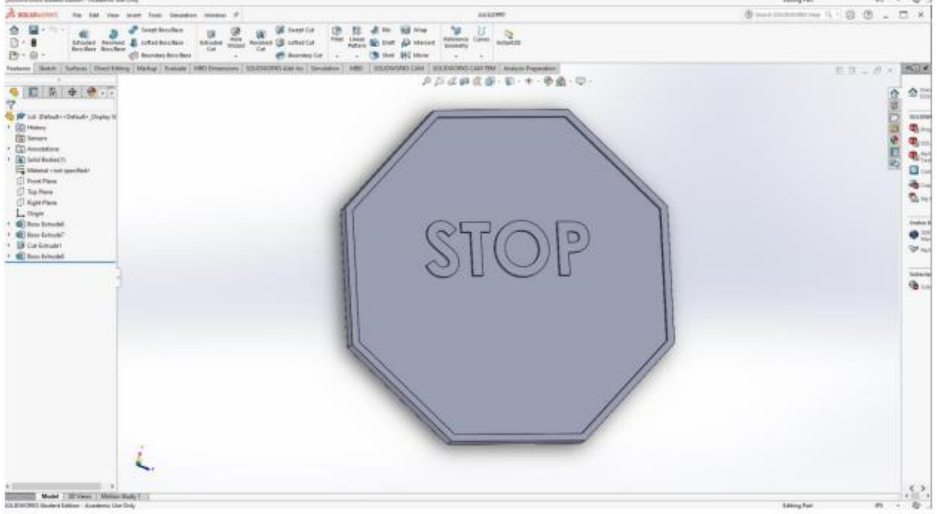
ATmega32 Pins.png	add screenshot of pins for reference	last month
Hardware Requirements.PNG	add requirements	last month
Microcontroller.ino	integrate Bryan's code	3 days ago
Pins.pdf	replace with new pins	29 days ago
README.md	Initial commit	last month
Software Requirements.PNG	add requirements	last month

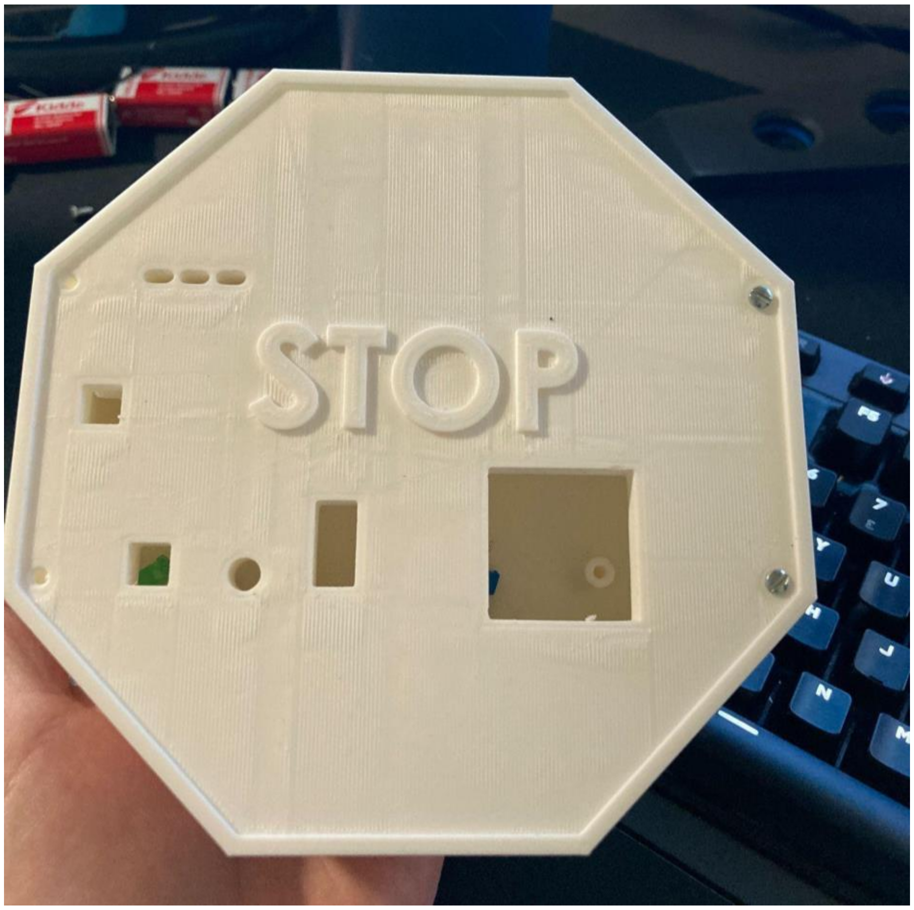
README.md

Bop-It

Enclosure Design

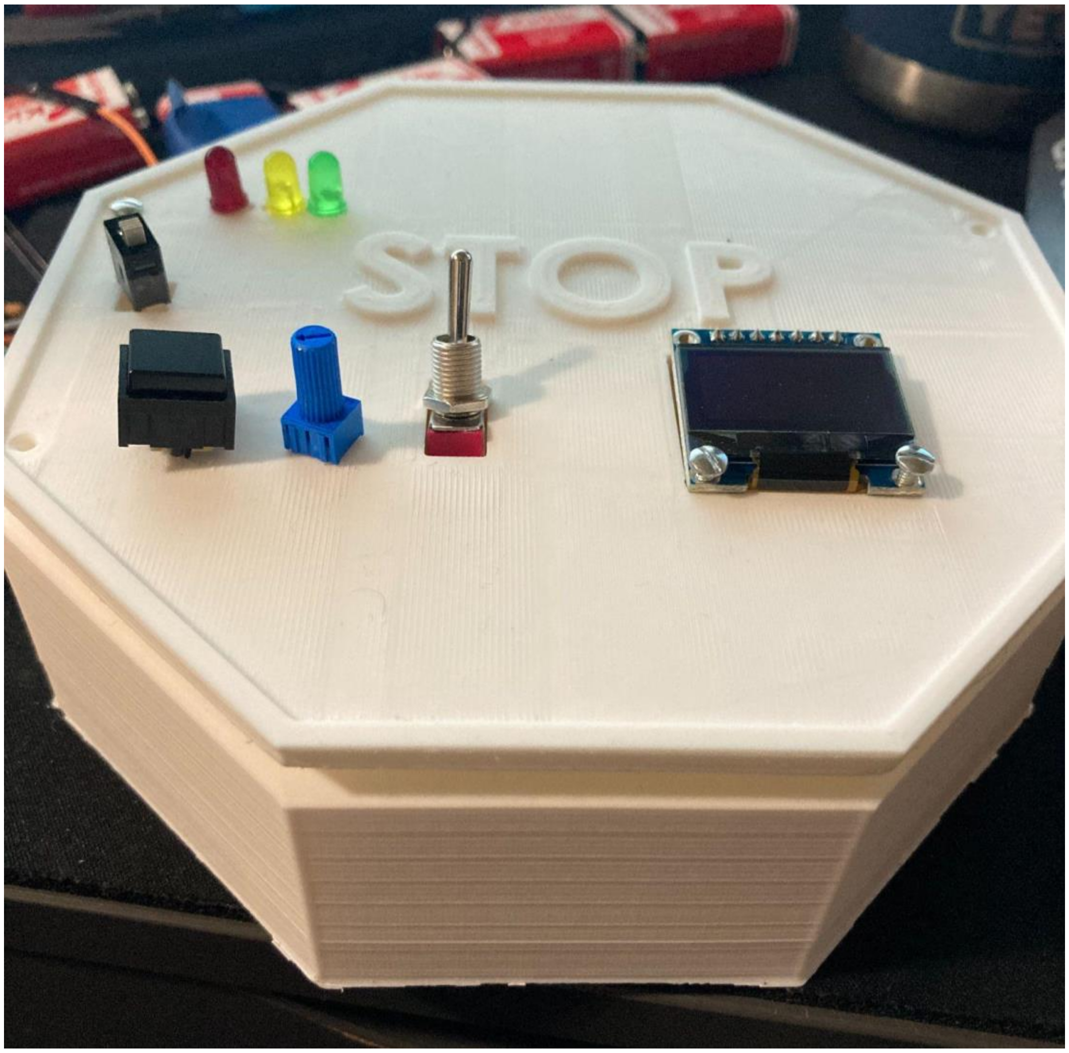
The enclosure was the first portion to be completed. The enclosure is a hexagon in the likeness of a stop sign. The 3D printed enclosure is divided into two pieces, a lid and a case. The two fit together and are screwed shut. Inside the enclosure are four standoffs used to attach the board to the enclosure to avoid sliding around. These standoffs also leave room for a 9-Volt battery to power the board. The biggest issues we met were with the physical assembly and printing a steering wheel. Initially, a steering wheel was supposed to be printed and attached to the potentiometer, however the wheel would not be stable while printing and ultimately had to be scrapped. This is why the potentiometer sticks out of the enclosure opposed to underneath the lid. The other issue was the lip used to align the lid up the case was too large and couldn't fit the lid flush to the case. Similarly, some of the exact components were changed after the print, so some of the component holes are a little small/large.

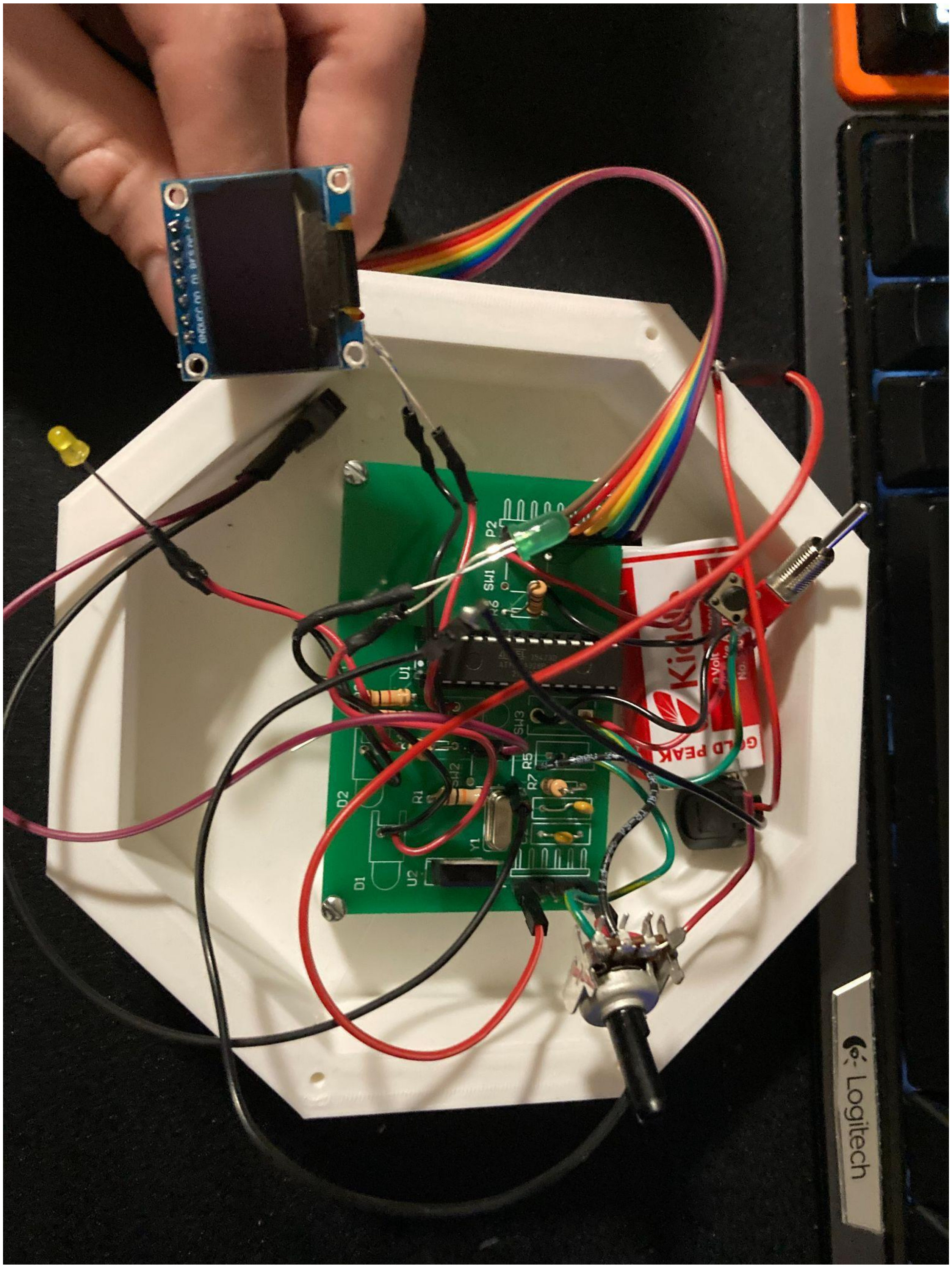



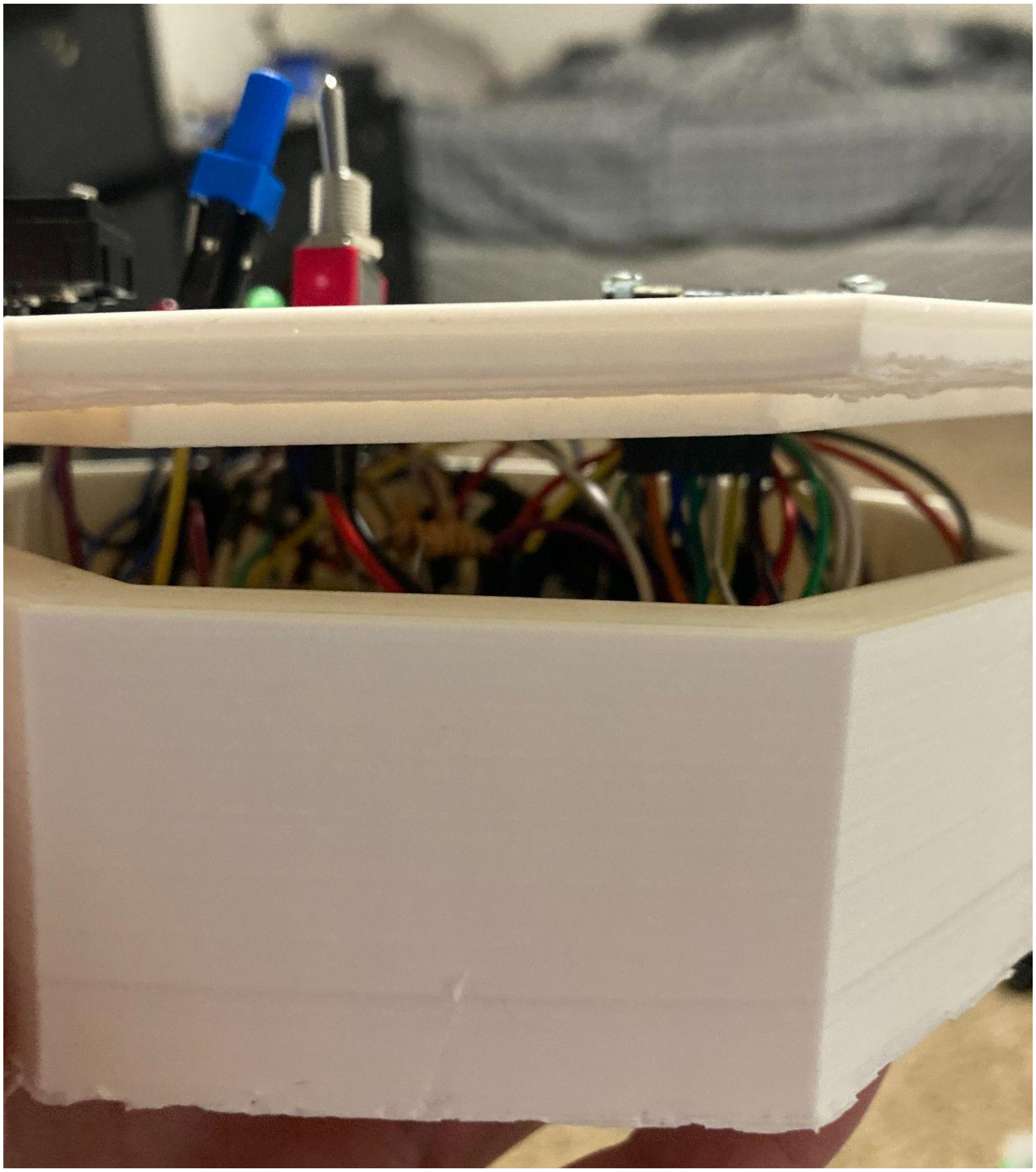

STOP

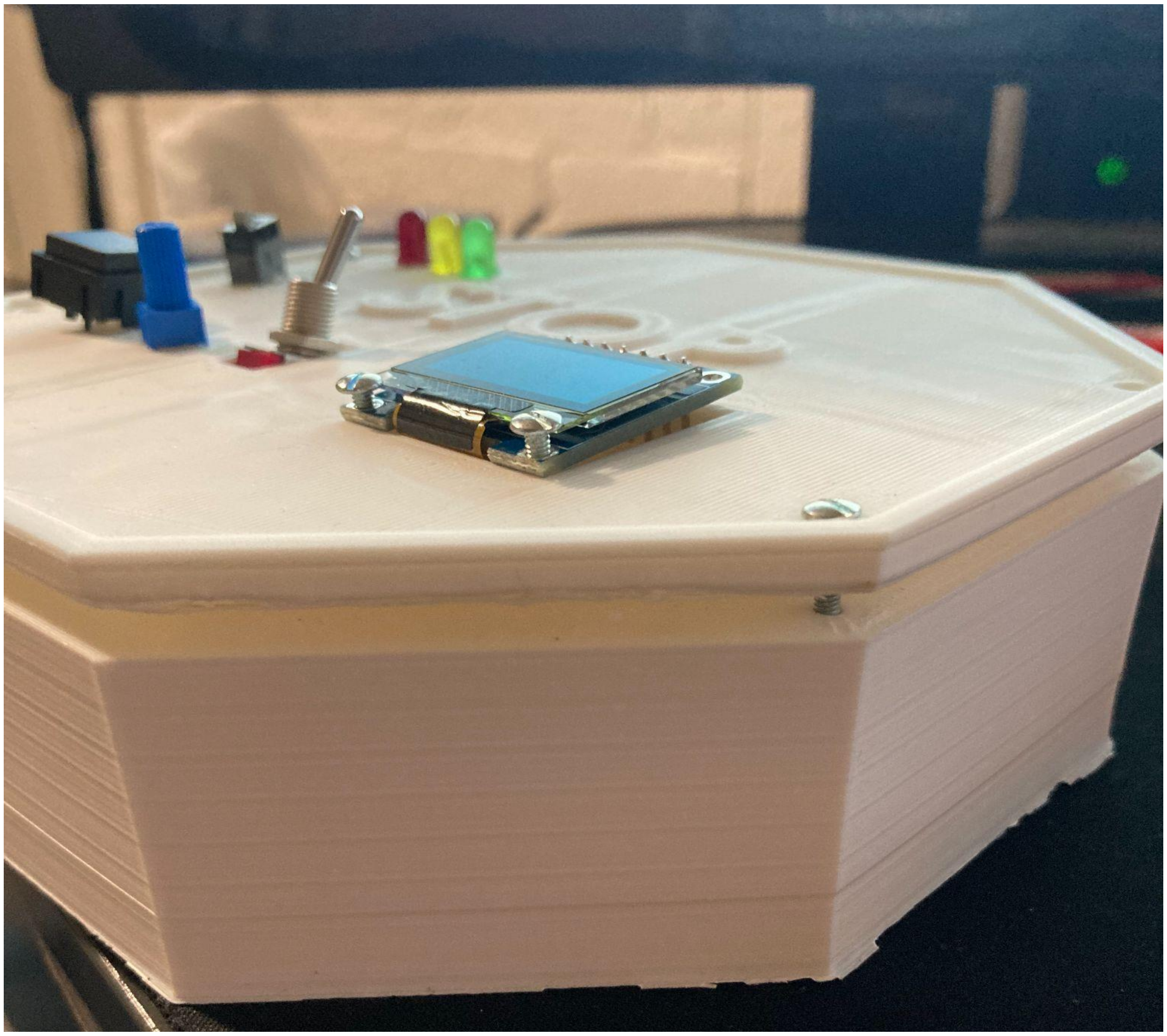
Assembly and System Integration

The assembly process was done in stages. The first stage was getting a working breadboard version of the final code. The second was getting that to work on the PCB fully. The final stage was getting it to work on the PCB in the enclosure. The first stage was completed via multiple iterations of code. The second stage required us to hardwire some parts to others. The final stage was where we ran into an oversight. One major issue with our final assembly was that the pcb was getting squashed by the lid. Due to us moving components around and hardwiring others, the components on our board stuck out more than we initially intended. We could have pushed the lid as close to closed as possible, however we wanted to avoid damaging the board, this is why in our photos, the lid is raised slightly. Also note, the "under the hood" photo of our PCB has slightly different components (i.e. the push button is different); this is because we had to desolder and resolder the parts back on in order to fit it though the lid and some newer parts worked better than the ones we originally attached.




P2
SW1
R6
U1
SW3
R5
SW2
R1
R7
D2
Y1
U2
D1
Kidde
GOLD PEAK
Logitech





Design Testing

For whole system testing we wanted to test every condition possible. These conditions are: max score reached, time out end, and wrong input end. For max score reached, we simply played the game until the max score was reached. For testing purposes, we set the max score to 10 in order to save time. We encountered an error our first few tests and the game would reset without displaying the LCD win screen. We remedied this by fixing how our timer decremented so it would end at exactly 100 turns. The time out end was tested simply by not inputting any commands. This worked perfectly as intended. Likewise, the timer decremented each turn as inputs on the 10th turn would time out faster than inputs on the first. The wrong input condition was tested similarly to the time out end. This also worked perfectly fine.

Budget and Cost Analysis

Part	**Price**
PCB Manufacturing	≈$2,472
PCB Assembly	≈$5,369
Components	≈$196,600
Enclosure	≈$460,200
Total:	≈$664,641

This estimate comes to about $67 per unit, however this is only a prototype and many cost cutting steps could be taken to reduce price in the design. In addition, bulk prices for the enclosure and parts would be significantly lower, however we were limited in our capacity to get quotes that high so we just multiplied the value of the highest bulk orders we could find.

Team

Team Roles:
Ben Nguyen: Microchip code designer and unit tester
Bryan Hess: Enclosure designer and system integrator tester
Aderotimi Adetunji: PCB/circuit designer and unit tester

We split our work up into logical chunks so that every member had something to work on. We planned a meeting via phone group chat and met in person at the lab as well as in class. We initially used the Kanban board, however it became easier to just give updates via the group chat like a sudo scrum meeting.

Timeline

Oct 19 - Final team idea conceived
Oct 22 - Enclosure 90% completed
Oct 28 - Parts selected and each input tested for viability
Oct 30 - PCB designed
Nov 4 - Unit testing of code 50% complete
Nov 9 - Extra parts ordered
Nov 13 - Unit testing of code 75% complete
Nov 20 - Enclosure finished and PCB 90% soldered
Nov 24 - Code fully working on breadboard
Nov 29 - Final integration 90% complete and full system testing
Nov 30 - Final iteration working
Dec 3 - We pray for a good grade

Summary, Conclusions, and Future Work

In short, this project taught us a lot about product design and prototyping. Each of us had varying degrees of experience in this field due to co-ops. The final Stop It works however there is much room for improvement. First and foremost, we would have more heavily tested the breadboard before submitting our PCB. Many of the later on issues came from overhauls in our design. Similarly, we would have designed the enclosure directly with the PCB in order to have perfect fitting cut outs. We would have also designed the enclosure to have more vertical space to allow for components to fit better. Similarly, we would have held some more meetings in person to avoid miscommunications that led to some inherent misunderstandings about the device between team members. Given more time, we would have implemented the sound queues and a start game button as well. For future group assignments we will better manage our meetings and set aside more in person times to work on each aspect together to form a more cohesive unit. All in all, the Bop It project was a lot of fun and gave a lot of insight into how prototyping works for products.