Design Project #2 – Bop-It! – Team Delta

Group Members: Jason Matuszak, Zach Kubitz and Joe Tonecha

Design Overview

The purpose of this design project was to reimagine the classic “Bop It” game with our own hardware and software design. Through the use of the ATMEGA328 microcontroller and a circuit with varying inputs, an interface to play Bop It was developed. The created PCB was then placed in an enclosure for the user to easily interact with a solid and stable final product.

The design of the Bop It did not vary much throughout the design process. As our group planned out different inputs and designs for some take on the final product, we came up with some of the following ideas below:

- Use a 1x4 solid keypad as an input for users to type in a 4-digit code as a command
- Take a colored LED push button as a command input
- Implement a Piezo Vibration sensor that receives vibrations as an input for the Bop It design
- Use a simple turn potentiometer as a tactile command input
- Implement a 555-timer circuit as a means of generating different tones for commands
- Install a hex display to keep track of score
- Install an 2x16 LCD display to keep track of score and potentially shows codes for keypad input
- Use an SD card and reader connected to speaker to read verbal recordings of different inputs

While not all these ideas were realized in the final design, they offered a solid base for Team Delta to discuss viable options moving forward. For example, the LCD was used over the hex display because of thoughts that the LCD would offer more functionality and utilize less pins. Additionally, the loudspeaker was used to audibly signal commands for simplicity and efficiency. These design decisions are fleshed out more in subsequent sections.

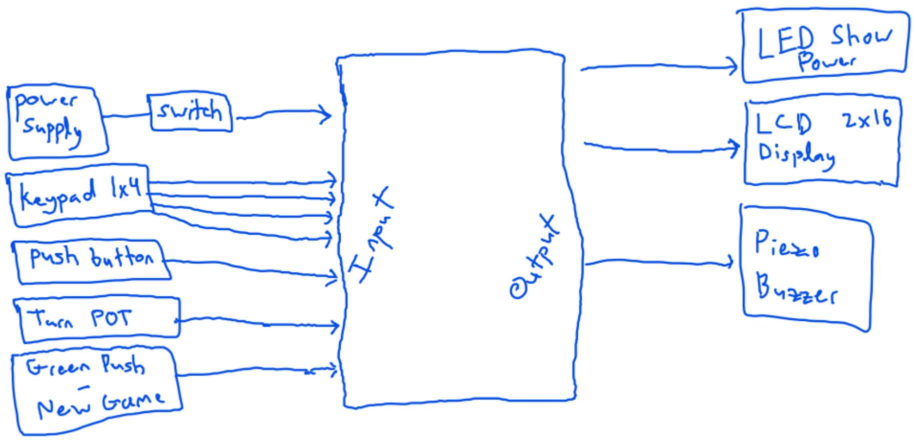


Figure 1- Final Block Diagram of Bop-It

Through a combination of independent research on online sites such as Adafruit, Sparkfun, Digikey and Amazon, as well as pulling our different ideas together, we came to the block diagram above. The final Bop It design would have three main inputs to facilitate the commands performed by the user in any typical Bop It game. The 1x4 keypad would be used in the "Type It" command. The push button would be used in the "Bop It" command. The turn potentiometer would be used in the "Twist It" command. The loudspeaker would be used to generate tones that would correspond to each of these different commands. Upon completing a command, an LCD display would be used to update a running score count. The switch that connects the Bop It power supply, 9V battery, to the microcontroller circuit was used to conserve battery when the user is no longer playing. An LED would be used to show when the power is on for the Bop It. A rough list of the parts we imagined we would need later for this circuit is in the figure below.

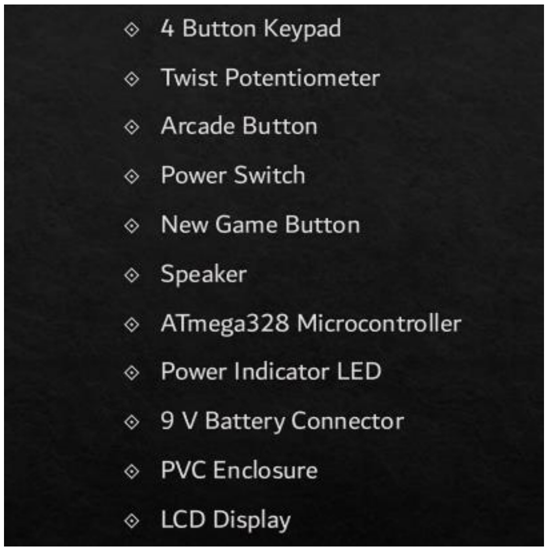

- 4 Button Keypad
- Twist Potentiometer
- Arcade Button
- Power Switch
- New Game Button
- Speaker
- ATmega328 Microcontroller
- Power Indicator LED
- 9 V Battery Connector
- PVC Enclosure
- LCD Display

Figure 2 - Components of Bop-It

Design Verification

In order to verify our design, we breadboarded our circuit once all of our parts were received. This one done slowly and in stages to verify that each component functioned properly with the ATMEGA 328 by itself before bringing everything together. This was done with the LED push buttons, the loudspeaker, the keypad, the LCD and the turn potentiometer. Additionally, throughout this process we wired everything up around the programmable circuit for the Arduino. Several times, we also programmed the ATMEGA 328 on another breadboard with the proper code before placing it back into the Bop It circuit. This process helped our group not only verify our circuit design but also refine our code as we figured out how to use functions from different libraries. For example, with the loudspeaker wired correctly to the ATMEGA 328 we were able to play with the tone() function.

Once everything functioned individually with the main chip, we started bringing things together and testing them together. For example, we checked to see that the speaker would output various tones while also checking for input from the bush button or keypad. As we verified that all these different commands worked concurrently with one another we felt more confident about our original Bop It schematic. Eventually, we breadboarded everything together. In the figure below, we have the AVR pocket programmed hooked up to the Bop It circuit so that the ATMEGA 328 could be programmed without having to continually remove it from the circuit. Over the course of several days and some debugging (expanded on further in the "Design Testing" section), we were able to verify our Bop It schematic.

Resources that aided us along the way were individual datasheets for some of our components such as the LED push buttons and 2x16 LCD. Online libraries and other people's online Arduino projects helped improve our understanding of each component and give us an idea of how our circuit should be designed. In addition, questions and answers from Piazza and fellow peers helped us gain confidence in our final design.

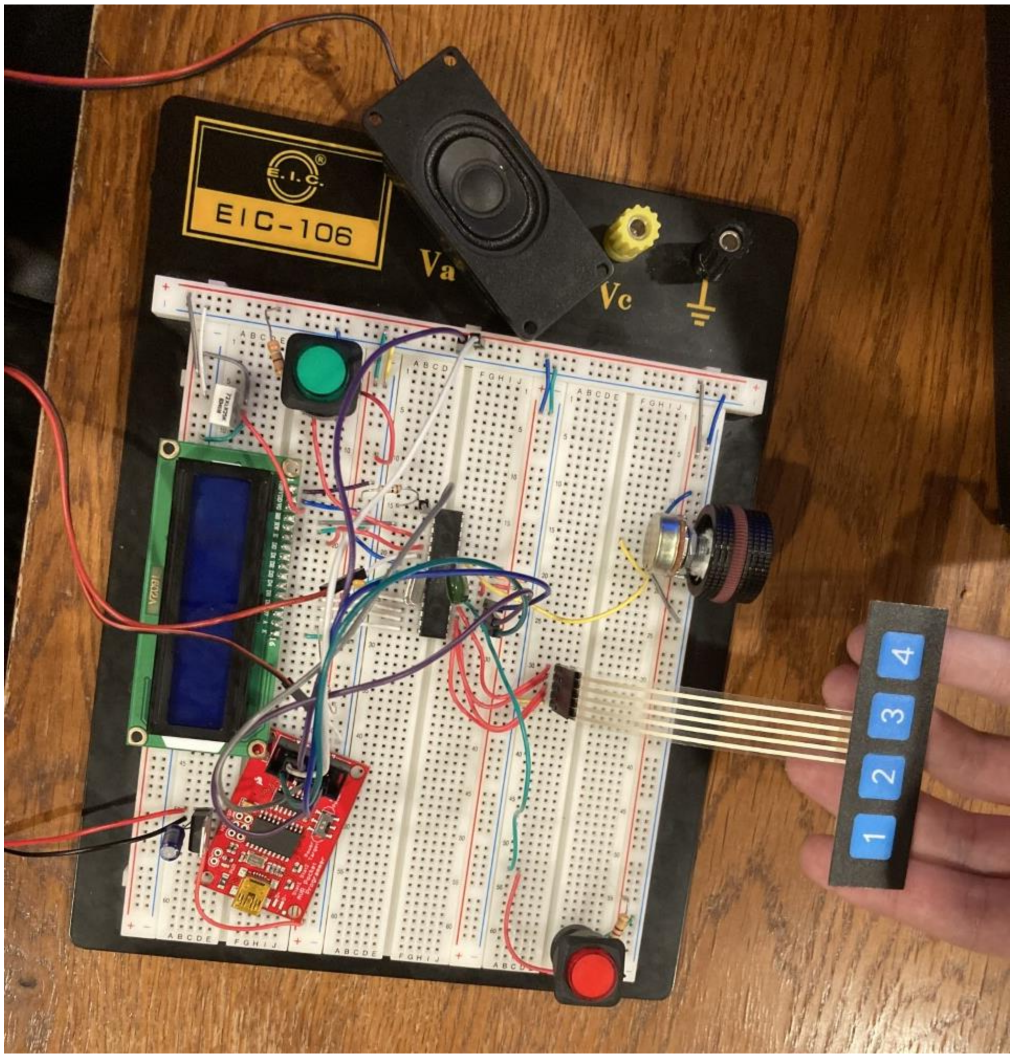


Figure 3 – Breadboard Testing of Bop-It

Electronic Design Implementation

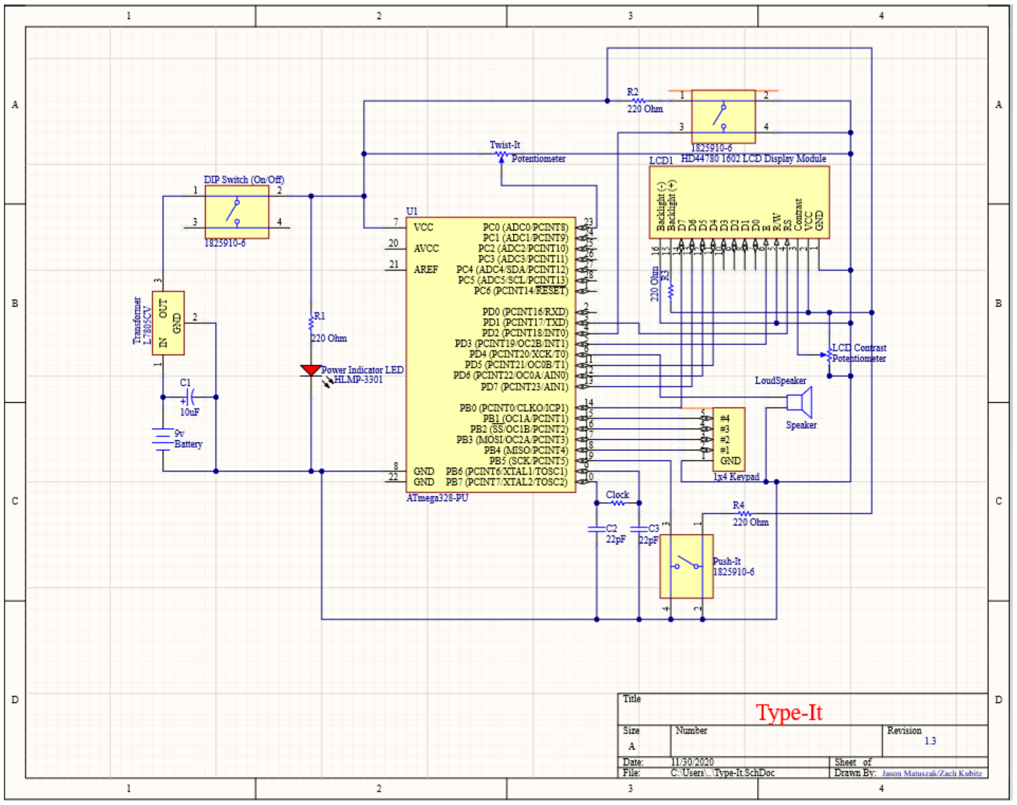


Figure 4 - Final Bop-It Altium Schematic

Within our final schematic, we have all our required game inputs such as the bop-it/push-it button, twist-it potentiometer, and the 1x4 keypad. We specifically reserved pins 15-19 for the keypad to make the schematic look neater. This also aided Zach in the software implementation part by not assigning arbitrary pins. However, the first component we had to connect to the microcontroller was the LCD. This was because our particular display required the connections to all the exact pins as shown above, and it would not function properly otherwise. From there we began to build the schematic based on the required pins for the LCD and keypad. We strategically assigned both the push-it and reset buttons to digital pins around the LCD and keypad for organizational purposes. The one component we were concerned about connecting was the loudspeaker. We were worried that the timers inside of the microcontroller would delay too much or fail due to interfacing with both the LCD and the loudspeaker simultaneously. However, this was not the case. The only affect we noticed was a slight flicker on the LCD when a sound was outputted. After that, the twist-it potentiometer was easy to decide where to place considering it was our only analog input. Next, we included the 16 Mhz clock within our design along with the 2 capacitors to function properly. In consideration of powering the circuit, we utilized a small dip switch which receives 5 volts from the regulator we employed with a 9-volt battery. In the end, we decided to scrap the power indication LED and its resistor due to both our reset and push-it buttons emitting light when the circuit is powered. We figured that would suffice for visualizing power.

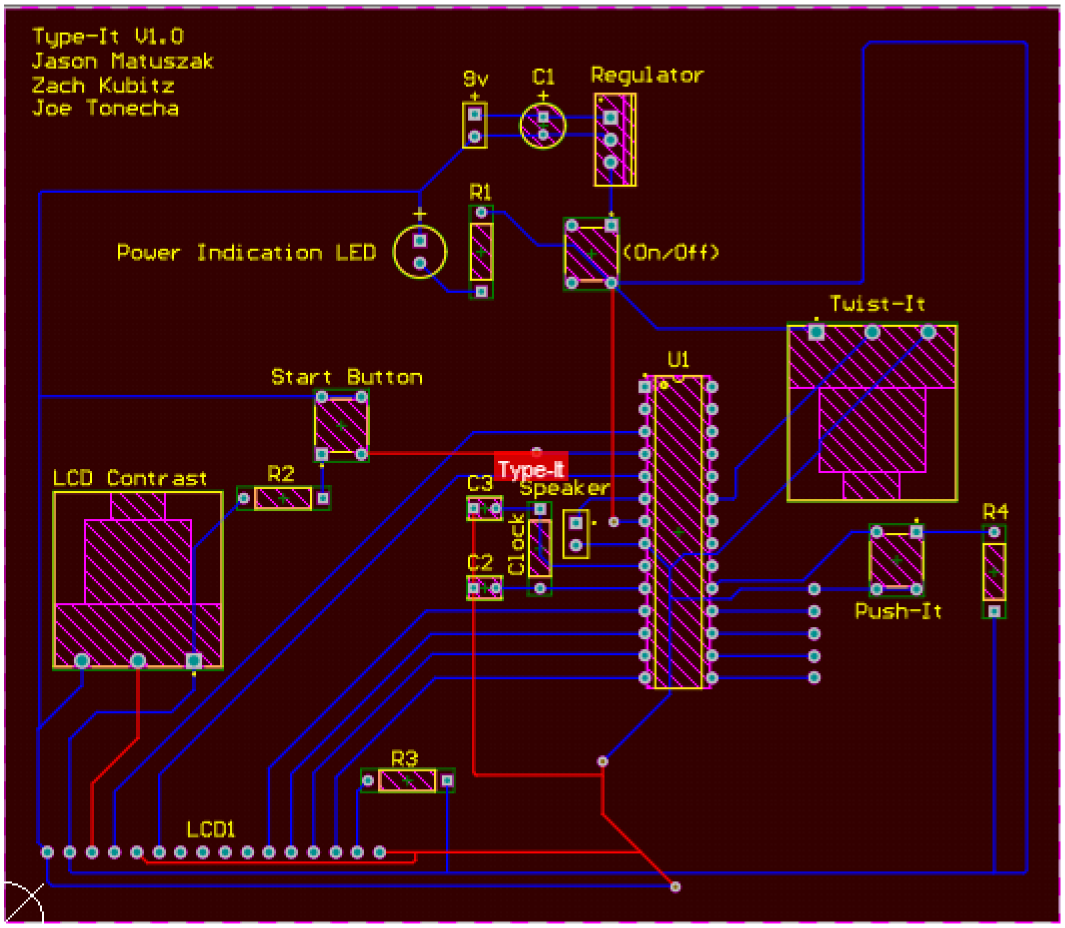
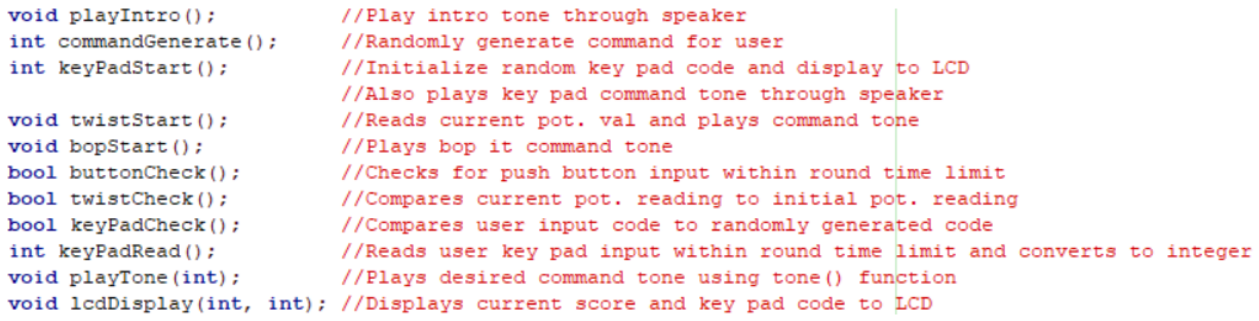


Figure 5 - Final Bop-It Altium PCB Layout

Approaching the start of the PCB layout process was difficult and stressful due to multiple factors. First, there were no Altium symbols or footprints for both the LCD and the 1x4 keypad. We tried several different modifications within the footprint wizard until we finally created footprints that would connect properly within the layout. It was hard to locate the specific data sheets for both the LCD display and 1x4 keypad to acquire the correct dimensions to implement into their corresponding symbols and footprints. Next, just like the schematic, the PCB layout had to be modeled around the LCD and the 1x4 keypad. While keeping the enclosure design in mind, we placed the LCD vias toward the bottom of the PCB so it would line up correctly with the enclosure. After that, we added in our power elements such as the battery, regulator (with capacitor) and dip switch for power. We then began to place the remaining components with respect to the locations of the power and ground routes. This required 4 vias exactly to be able to reach the twist-it potentiometer, the start/reset button, and the clock capacitors. At first, we experienced some difficulty with placing vias and having them connect, but towards the very end of the layout completion we integrated them correctly. Finally, as we began to construct our first PCB prototype, upon verifying its functionality, we decided to remove the power indication LED because our other arcade switch inputs already emitted light.

Software Implementation

The entire process of developing software was broken up into three sections: pseudocode, a testing program, and the main script. First, the pseudocode was a rough outline of the main gameplay logic along with a listing of any functions necessary to make said main program run. Most of the program is written into these functions to keep the main loop minimal and used mostly to control the state of the game. Next, as the parts for the bop it arrived, we wanted to verify that we could interface each piece with the microcontroller and receive correct inputs or outputs for all our components. To test the input pieces (twist potentiometer, bop it button, and keypad), Zach wrote software that would check each of these inputs individually and then output a random tone to the speaker. The tone generation for the speaker was done through the included tone() function already available in Arduino. All it took was an input denoting the desired frequency of the tone and another input designating the length of said tone. After he verified all the main input pieces and speaker worked as expected, Zach wrote some more test code to try and interface with the LCD display. The code for the LCD was straightforward as the display we used had an Arduino library already created for it. All it took was several premade function calls to initialize the LCD to the correct pins and a single function call to display data to the screen. After determining all our parts worked correctly and could be interfaced with the microcontroller, Zach began combining the code from the test program with the main logic of the game to create the final program.



```
void playIntro();          //Play intro tone through speaker
int commandGenerate();     //Randomly generate command for user
int keyPadStart();         //Initialize random key pad code and display to LCD
                           //Also plays key pad command tone through speaker
void twistStart();         //Reads current pot. val and plays command tone
void bopStart();           //Plays bop it command tone
bool buttonCheck();        //Checks for push button input within round time limit
bool twistCheck();         //Compares current pot. reading to initial pot. reading
bool keyPadCheck();        //Compares user input code to randomly generated code
int keyPadRead();          //Reads user key pad input within round time limit and converts to integer
void playTone(int);        //Plays desired command tone using tone() function
void lcdDisplay(int, int); //Displays current score and key pad code to LCD
```

Figure 6 - Listing of all functions used in main program

The main program relied heavily on function calls to keep the main loop compact and used primarily to control the states of the game. The playTone() function was called any time noise was to be produced from the speaker. This function consists of five custom tones Zach created by playing around with the built in tone() function and loops. This made it easy to distinguish between which input the user was expected to enter due to a different number of beeps being produced for each command. The other two custom tones were used to indicate the start of a game and when the user enters an incorrect command. Next, the commandGenerate() function randomly picks one of the commands for the user to enter, then calls the appropriate component start function to initialize that input and play the correct command tone for the user. At this point, the user has been provided which input to enter and the appropriate check input function is called from the main loop. The twistCheck() and buttonCheck() functions are very simple and work by checking for changes in voltage levels on the desired input pin of the microcontroller. A slight problem occurred when attempting to use the built in timers of the microcontroller and the analogRead() function for checking the potentiometer input after the twist command was called. While the timers were running, the analogRead() function produced constantly varying values of the potentiometer even if it was not twisted at all. This meant that the timers and analogRead() could not be used in conjunction and a hard coded delay of 2.5 seconds was used each time the twist input needed to be checked. The timers did not affect the other two check input functions and Zach was able to properly decrement the user response window for the keypad and bop it inputs, but the window for the twist input was always 2.5 seconds for the entirety of the game. Lastly, the most difficult function to develop was keyPadRead(). This subroutine collected individual inputs from the keypad until a full code was entered or the timer ran out. It then took this array of integers and converted it into a single number which was then compared to the randomly generated keypad code. When first testing this function, we were unable to get the keyPadCheck() function to return true even when the correct code was entered. This meant some things had to be debugged. Zach changed the keyPadRead() function around several times until he realized the problem. The pins from the keypad itself were not mapped in chronological order. This meant every time he pressed the key for 1, the microcontroller read the key for 2 and vice versa. This was also happening with the third and fourth buttons. After realizing the pin mappings were wrong, all it took was changing one number in the keyPadRead() function for it to work correctly. Now that all the check input functions were working, the main logic of the program could be completed. All the main loop consists of is a simple do, while loop which kept the loop running until the user entered an incorrect response, ran out of time, or reached the max score of the game. Each iteration of this loop also updated the LCD with the incremented user score and decremented the time the user has to respond to each input. All in all, it was gratifying to see the final version of the software functioning properly in our very own game.

Enclosure Design

The final enclosure design was decided on mainly based on simplicity. A simple box enclosure that could house the various components and PCB was created via an online box generator tool by “makercase”. Through providing box dimensions, choosing to have a lid, specifying material thickness close to a quarter inch, and choosing finger joints, we arrived at the box below. Finger size is automatically set and once all these options are selected the svg files were ready for laser cutting.

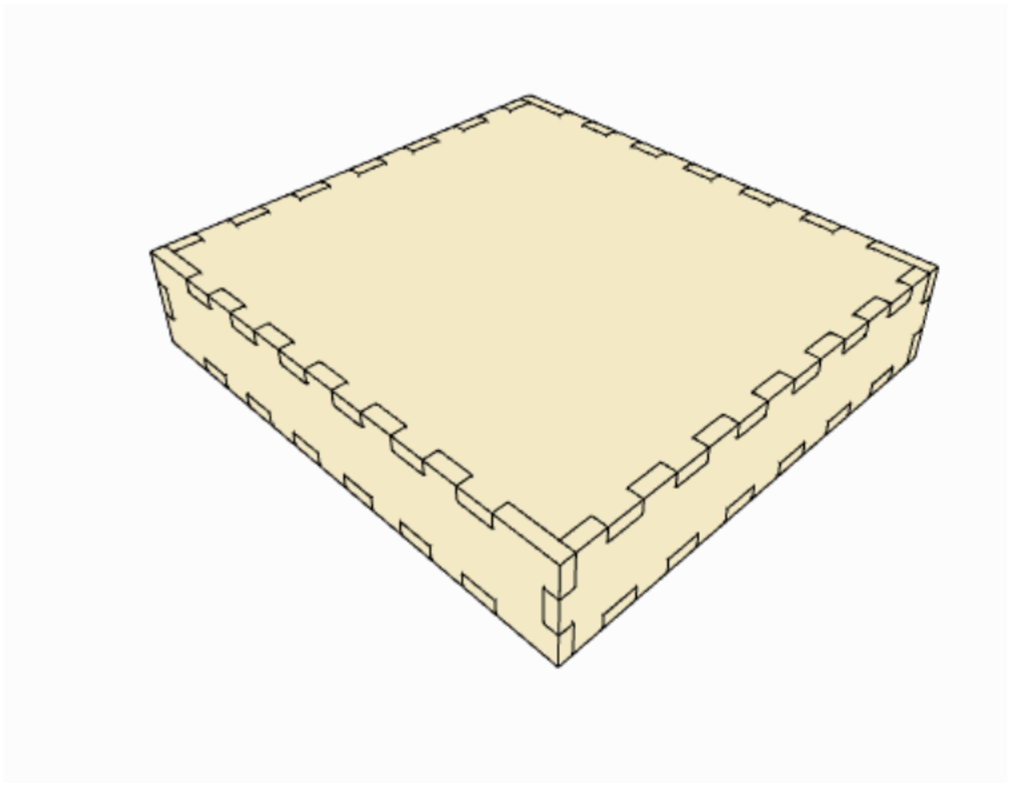


Figure 7 – Enclosure Case

Customizing the lid was the trickiest part of designing the enclosure. In order to have the varying tactile inputs be accessible from the outside, holes had to be cut in the lid. Doing this required transferring the svg files into Autodesk 360 Fusion, placing appropriately sized holes at certain places across the surface of the lid and then extruding the new surface. In order to get the right sized holes, we had to cross check dimensions of components with data sheets and also consider the placement of the PCB within the box. Holes in the lid were placed for the turn POT, two LED push buttons, a 2x16 LCD, a dip switch, an LED, the keypad and for sound from the speaker to exit the box. The svg file for the lid was separate from that of the box.

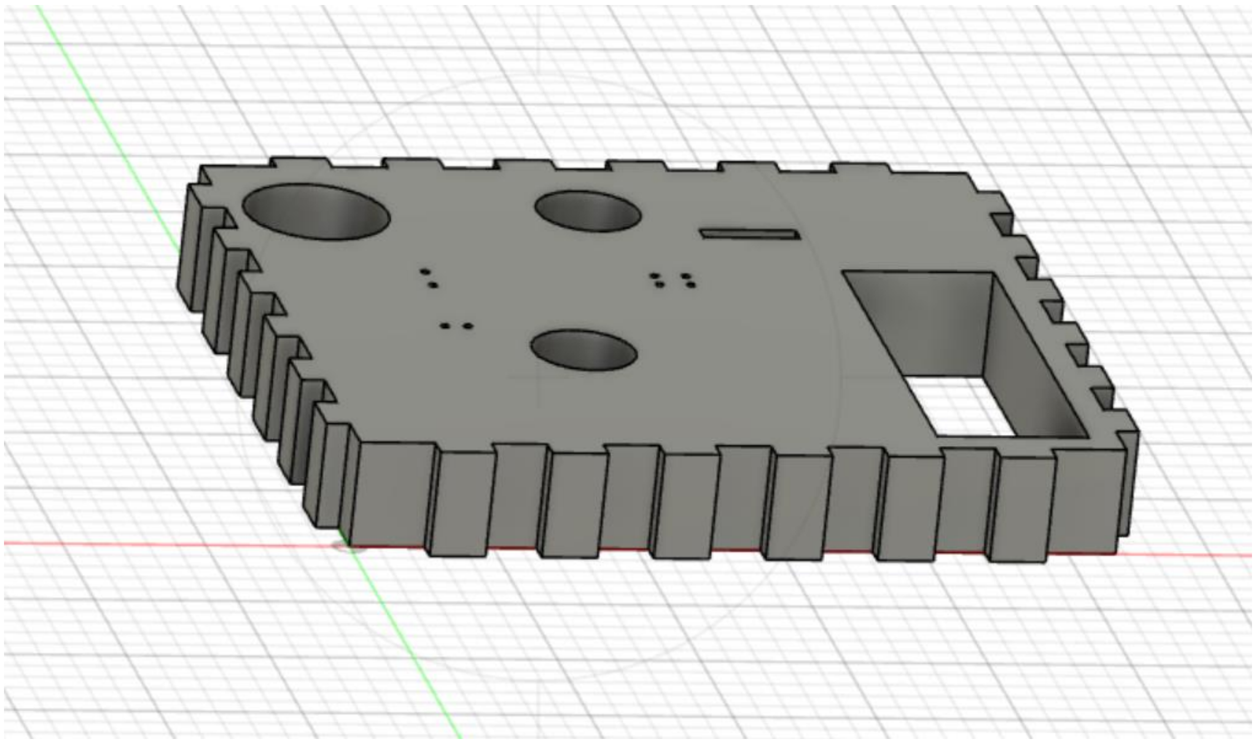


Figure 8 - Enclosure Lid

In order to manufacture the design, we set up a time to use the Glowforge laser printer with Bill. Basswood was used for the box and acrylic was used for the lid so we could see the hardware setup inside of the box. After each piece of the box was laser cut, we used super glue on the finger joints to fit the various pieces together (aside from the lid). The lid was meant to rest on the top and set neatly into the box joints. This way the lid would be removeable but also snuggly rest on the top when needed. However, different kerf values for the wood and acrylic made it difficult to fit the lid onto to the top. To fix this problem we sanded some of the box joints on the wood so the lid would then fit. Figures of the final enclosure and hardware inside of the enclosure are shown below.

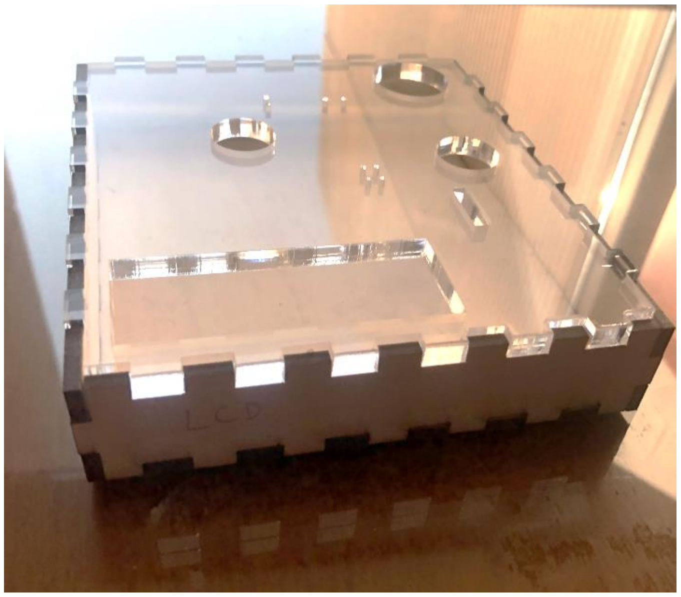


Figure 9 - Finished Enclosure

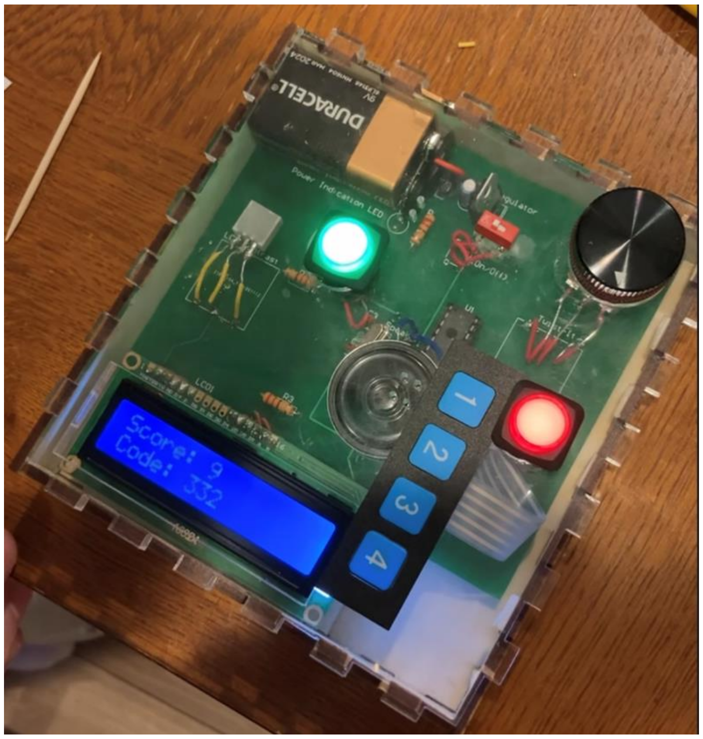


Figure 10 - Final Physical Enclosure Hardware Encapsulation

Assembly and System Integration

The start of our assembly process was rather interesting and lengthy, to say the least. We first began by verifying our Altium schematic via a breadboard after the initial breadboard verification. Having all met to construct a circuit and test components, we used a small program that Zach wrote to see if we were interfacing with all the inputs/outputs correctly. This attempt was unsuccessful at first. Trouble getting the LCD to work presented issues and stalled our design verification. We were also unaware that the clock and coupling capacitors were required in our initial Bop It schematic. However, with Joe troubleshooting the LCD via its datasheet and helpful posts online we were able to integrate the component into the design. Zach and Jason also realized the circuits need for an external clock and included that in the circuit schematic. At this point, we had fully verified that our schematic worked on a breadboard and were comfortable moving forward with the PCB design.

Afterwards, we ordered more components and Jason began creating the Altium PCB layout. Once the PCB was finalized, it was reviewed by all team members so we could discover and fix any potential errors. Luckily, there were none. Immediately after we received our PCBs, Jason began soldering a rough prototype without an enclosure by utilizing all of the components from the breadboard verification test (this was why we ordered more parts after schematic/breadboard verification). As soon as the prototype was ready, Jason gave it to Zach who ran the same input/output test program on the board, and interfacing was completely successful. Now that Zach and Jason both knew the PCB implementation was functioning properly, they communicated the information to Joe so he knew the enclosure design process could be finalized. From completing the Autodesk Fusion 360 tutorial, Joe was able to use some features of the software to make a custom enclosure for the PCB design and components.

While this took place, Zach now had a fully constructed PCB prototype to test his software. As Zach was working on the software, Joe delivered the completed enclosure to Jason who began soldering the final product immediately. However, due to the low ceiling of the enclosure and arrangement of components underneath the lid soldering proved a very difficult task for Jason. This also made it difficult for the acrylic lid to rest in the box joints. If there were time to make another iteration of the enclosure this would be heavily considered (look to "Summary, Conclusions and Future Work" section for more details). The difficulty in housing the PCB was also partly the reason we decided to remove the power indication LED from the final product. We also decided to use a smaller speaker within our final design due to it not fitting in the enclosure correctly. Thankfully, Joe had a small speaker on hand for use in the project. Our keypad also did not end up in the ideal location that we had imagined—on the side of the box. However, due to the short ribbon cable length, we had to compromise and place it on top of the enclosure next to the push-it input. However, Jason did a good job soldering the board despite these challenges.

Soon after the final hardware design was constructed, Zach finished the entire software portion and verified its stability with the rough prototype PCB. He then handed off the microchip to Jason who placed it into the final design with needle nose pliers. Upon turning the unit on, it did not function properly and cut out during the starting tones of the game. Jason began troubleshooting and debugging possible hardware issues with a multimeter. No issues were found until Zach placed the chip back into the prototype design and noticed the same error. It seemed that the chip had either been damaged or malfunctioned. Flashing code to another ATMEGA 328 solved the issues that were experienced with the other chip. Again, Jason received the microchip from Zach and placed it into the final PCB. Connected to a power supply, it immediately responded, and we were playing our Bop-it game within minutes. Now that we knew the end product was functioning properly, Jason placed it into the case of the enclosure. Again, our enclosure did not shut completely but securely housed the hardware with the use of tape.

Design Testing

Preliminary Testing:

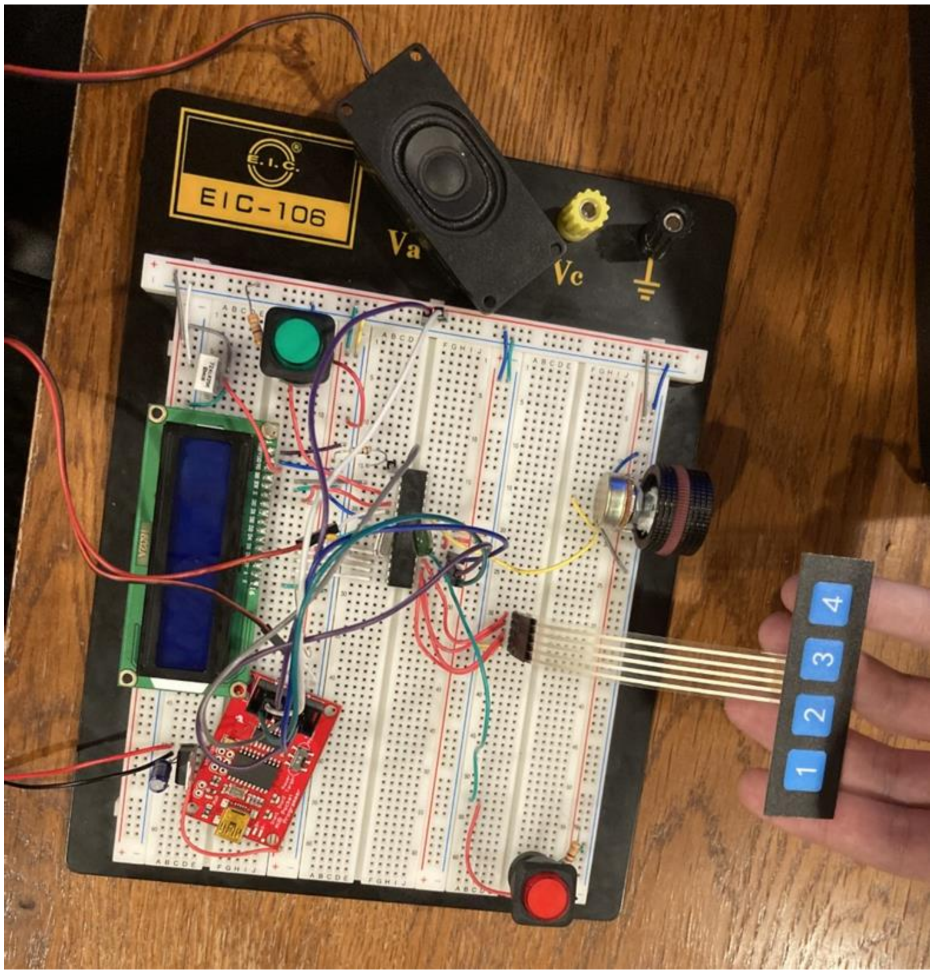


Figure 11 - Preliminary Design Test on Breadboard (Reused from Design Verification)

To begin preparation for this project, we started by verifying all our individual input/output components on a breadboard both before and after our Altium schematic was completed. Although we never created a fully functioning breadboard prototype due to the software taking a lot of time and effort, it was not necessary because we confirmed all of our ideals and intentions with this first preliminary design test. We also decided to keep all the components hooked up to the programming circuit during testing to be able to re-flash the microchip with new software a lot faster. After we were able to correctly interface with all of our components, we slowly began disassembling the programming circuit around the breadboard to see what portions of it were necessary for full functionality. We concluded that we only needed to include the clock and its 2 capacitors in our future implementations and updated our schematic accordingly. After this preliminary test was concluded, we began moving onto the next phase of the project which was laying out our PCB.

Prototype Testing:

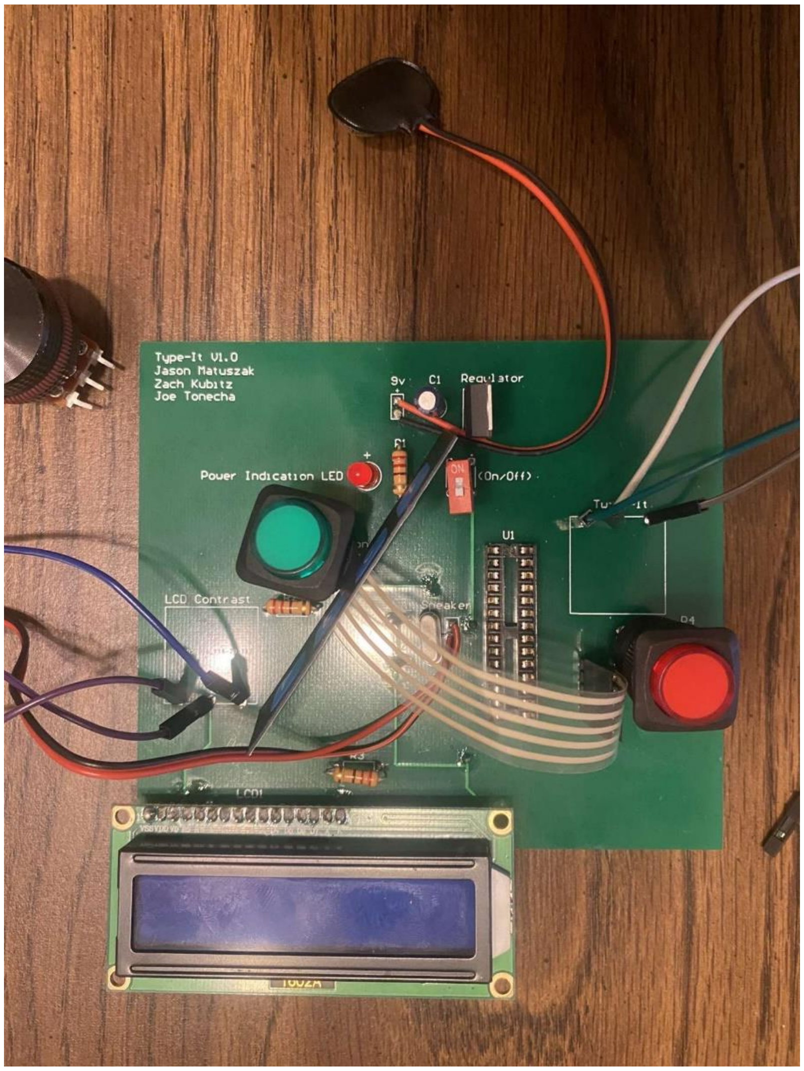


Figure 12 – Rough Prototype PCB

Once we received our laid out and received our fabricated PCB, Jason began soldering using all the components from the preliminary breadboard test, but without the enclosure. After the hardware was constructed, Jason then gave the prototype to Zach who at first ran our initial preliminary test program to confirm that our PCB was working properly. Then Zach began vigorously testing multiple new portions of actual game code as he wrote it. The prototype remained in Zach's possession for the remainder of the project since he wrote most of our game program. During the software implementation process, Joe was working on constructing the enclosure. Once it was finalized, the enclosure was transferred to Jason who then began building the final design.

Final Testing:

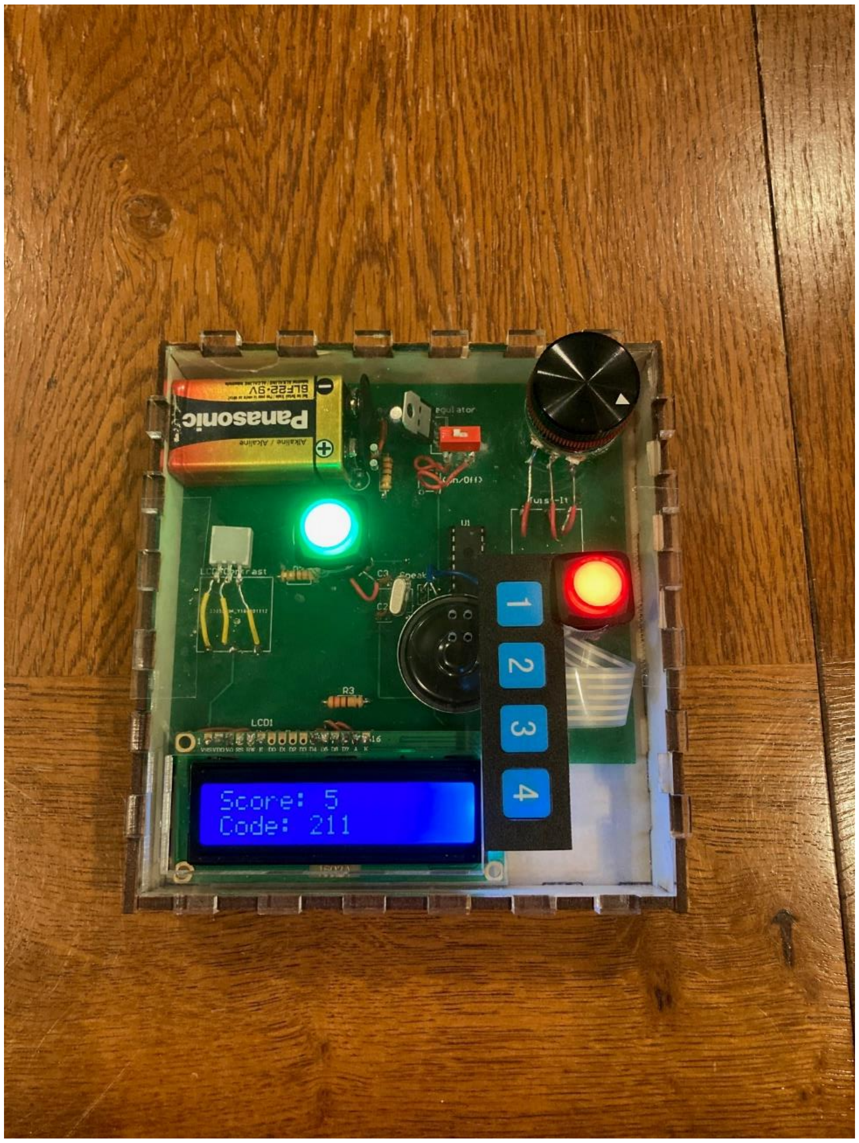
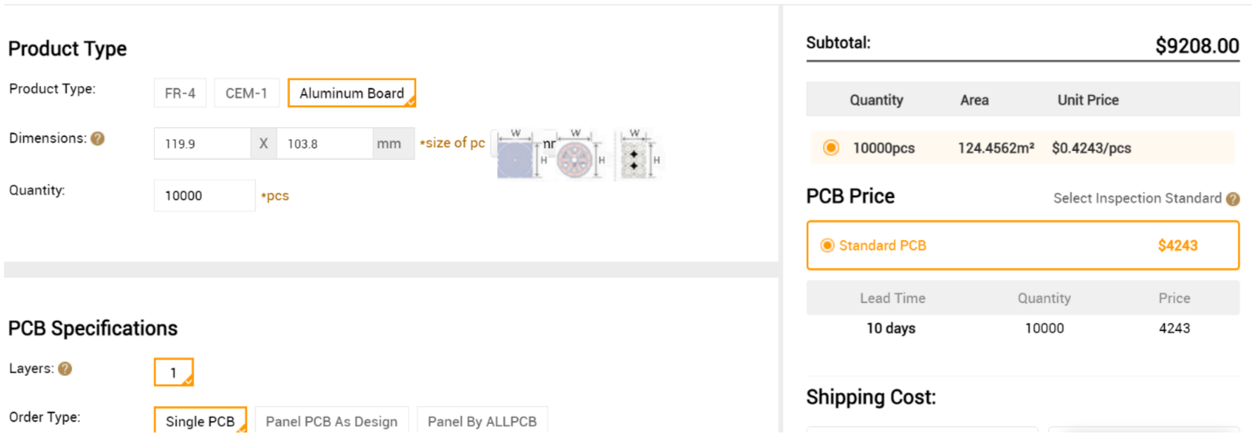


Figure 13 - Finalized Bop-It Design

WATCH *NEW* FINAL DEMONSTRATION VIDEO HERE: https://youtu.be/VfUyPxHhu6M

The final microchip was also given to Jason who placed it into the end product (besides our first chip being broken we encountered no other problems, as described in "Assembly and System Integration" section). Upon placing the chip with the final software version into the encapsulation, our Bop-It worked almost flawlessly with only a few minor bugs. As discussed in the software implementation section, the requirement of decreasing the time the user has to respond each round works for the bop it and type it commands, but a hard coded timer of 2.5 seconds had to be used for the twist it command due to the interference from the built in timers while using the analogRead() function. Also, when the user is given the key pad prompt and types in fewer than 3 keys, if the first key entered was the correct key the game automatically fills in the remainder of the code with the correct values giving the user a correct answer when they should have failed. If the user enters no code or the first value is wrong, the player will fail as expected. Other than these minor flaws we are happy with the functionality of our bop it.

Budget and Cost Analysis



Product Type

Product Type:	FR-4 / CEM-1 / [✓ Aluminum Board]
Dimensions:	119.9 X 103.8 mm *size of pc
Quantity:	10000 *pcs

PCB Specifications

Layers:	1
Order Type:	[✓ Single PCB] / Panel PCB As Design / Panel By ALLPCB

Subtotal: $9208.00

Quantity	Area	Unit Price
10000pcs	124.4562m²	$0.4243/pcs

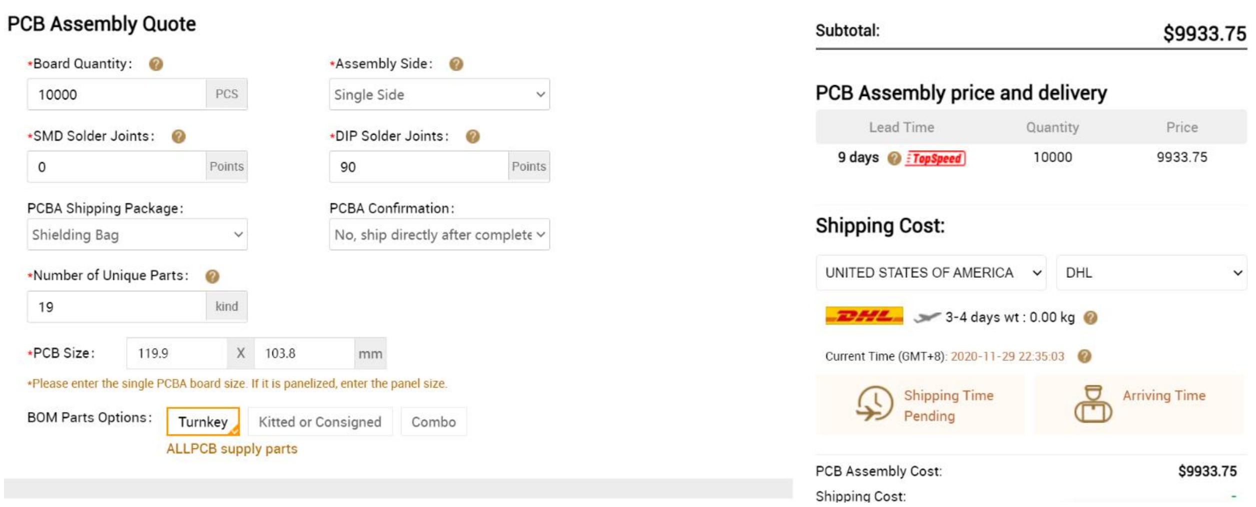
PCB Price Select Inspection Standard

Standard PCB $4243

Lead Time	Quantity	Price
10 days	10000	4243

Shipping Cost:

Figure 14 - PCB Manufacturing Quote



PCB Assembly Quote

*Board Quantity:	10000 PCS
*Assembly Side:	Single Side
*SMD Solder Joints:	0 Points
*DIP Solder Joints:	90 Points
PCBA Shipping Package:	Shielding Bag
PCBA Confirmation:	No, ship directly after complete
*Number of Unique Parts:	19 kind
*PCB Size:	119.9 X 103.8 mm
BOM Parts Options:	[✓ Turnkey] / Kitted or Consigned / Combo

*Please enter the single PCBA board size. If it is panelized, enter the panel size.

ALLPCB supply parts

Subtotal: $9933.75

PCB Assembly price and delivery

Lead Time	Quantity	Price
9 days TopSpeed	10000	9933.75

Shipping Cost:

UNITED STATES OF AMERICA | DHL

3-4 days wt : 0.00 kg

Current Time (GMT+8): 2020-11-29 22:35:03

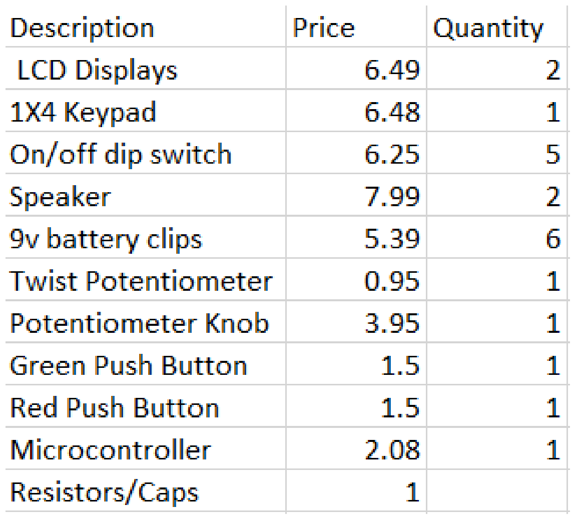
Shipping Time Pending | Arriving Time

PCB Assembly Cost: $9933.75

Shipping Cost: -

Figure 15 - PCB Assembly Quote

From the figures above, the PCB manufacturing quote from ALLPCB totals to $9,208 (shipping included) and the PCB assembly quote from ALLPCB totals to $9,993.75 (shipping included). As shown below, the cost of parts for one complete Bop It game comes out to $25.83 making the combined total for 10,000 units $258,300.



Description	Price	Quantity
LCD Displays	6.49	2
1X4 Keypad	6.48	1
On/off dip switch	6.25	5
Speaker	7.99	2
9v battery clips	5.39	6
Twist Potentiometer	0.95	1
Potentiometer Knob	3.95	1
Green Push Button	1.5	1
Red Push Button	1.5	1
Microcontroller	2.08	1
Resistors/Caps	1	

Figure 16: Cost of components

Extruded Acrylic was used for the lid design. This type of acrylic is best for laser cutting. A sheet with dimensions of ¼’’x4’’x4’’ is generally around $2 and this roughly fits our PCB dimensions of 4.65’’x4.05’’. ¼’’ basswood is ideal for laser cutting the rest of the enclosure. A minimum of 32 in^2 is needed for the other parts. 6’’x6’’ basswood runs roughly around $1.80. Therefore, for the total enclosure construction, and ignoring cost of using the Glowforge laser cutter, the price runs at $3.80. For a 10,000 volume order the total would run close to $38,000.

All in all, the combined total of manufacturing and assembling the PCBs, the cost of parts, and the cost of enclosure fabrication brings us to a grand total of about $315,441.75. This price is much more than we expected but factoring in bulk orders in the future our final cost could decrease significantly.

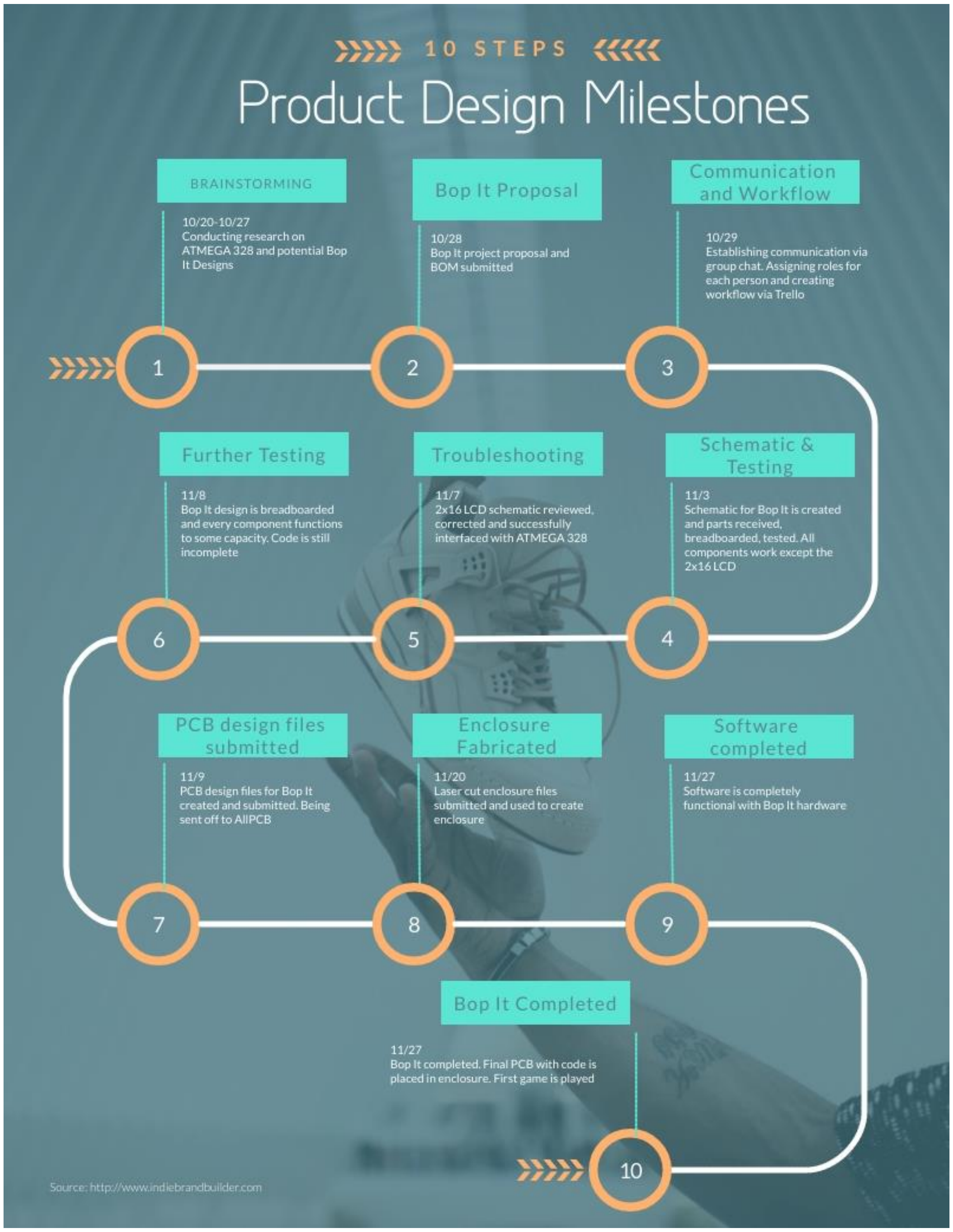
Team

Our team was comprised of three ECE students. Jason Matuszak is a senior COE student with interests in robotics and machine learning. Zach Kubitz is a junior COE with interests in robotics and coding. Joe Tonecha is a junior EE student with interests in power electronics and machine coding. Jason had a role in schematic creation, PCB overview, testing and hardware implementation. Zach had a role in software implementation, PCB overview, testing and hardware implementation. Joe had a role in enclosure design, retrieving parts, PCB overview and testing.

The team decided to communicate primarily through a group chat. This allowed us to readily receive communication from each other at any point of the day. There were times where Trello was used to update each other on work progress. However, the number of people in the group as well as living near each other allowed for a combination of texting and in-person meetings to set us up for efficient work progress. There were several times where we all scheduled to meet in-person in order to breadboard designs, test components and develop a functional PCB design. There were also times where group member Joe would travel to Benedum, pick up parts and deliver them to the group. Other times group members Zach and

Jason met to collaborate on coding, solder components, and testing of the product. Any in-person meetings conducted were done at one of our group members off campus apartments. Also, Github proved most useful throughout developing our PCB design files as we shared schematic and PCB documents.

Timeline


10 STEPS
Product Design Milestones
1
2
3
4
5
6
7
8
9
10
Source: http://www.indiebrandbuilder.com

BRAINSTORMING
10/20-10/27
Conducting research on ATMEGA 328 and potential Bop It Designs

Bop It Proposal
10/28
Bop It project proposal and BOM submitted

Communication and Workflow
10/29
Establishing communication via group chat. Assigning roles for each person and creating workflow via Trello

Schematic & Testing
11/3
Schematic for Bop It is created and parts received, breadboarded, tested. All components work except the 2x16 LCD

Troubleshooting
11/7
2x16 LCD schematic reviewed, corrected and successfully interfaced with ATMEGA 328

Further Testing
11/8
Bop It design is breadboarded and every component functions to some capacity. Code is still incomplete

PCB design files submitted
11/9
PCB design files for Bop It created and submitted. Being sent off to AllPCB

Enclosure Fabricated
11/20
Laser cut enclosure files submitted and used to create enclosure

Software completed
11/27
Software is completely functional with Bop It hardware

Bop It Completed
11/27
Bop It completed. Final PCB with code is placed in enclosure. First game is played

Figure 17 - Timeline of project milestones

Summary, Conclusions and Future Work

Our work on the Bop It led to a greater understanding of different software tools and packages such as Trello, GitHub and Arduino. The completion of this project enhanced our ability to effectively work as a group as well as the proper steps it takes to have a design be realized. If Team Delta were to do another iteration of Bop It, we would change several things about our design. First, instead of just playing distinct beeps to indicate which command the player must perform, we would add hardware that allowed for the commands to be words played through the speaker. Not only would this make the game easier, it would also make it resemble the original Bop It game more closely. Also, we would add to the software to include different game modes that the user can play. For example, we could add modes with varying difficulties for players who wanted more of a challenge. Lastly, we would redesign the enclosure. Its dimensions were suitable for the length and width of everything it was housing. However, the height made fitting certain components a little snug. Also, holes in the lid would be adjusted for various components. For example, the holes for the LED buttons were slightly small. Additionally, we would create a lid that could connect to a hinge rather than having one to be fitted into the top finger joints. Given that the lid is acrylic, and the box is basswood, different kerf values make fitting the top joints overly difficult. If we were mass producing this Bop It, rather than worrying about sanding/fitting box joints, a hinged lid would work much more seamlessly. It would also make soldering easier as we would not have to solder components already partially under a lid.