Poker Chip Counting Companion

ECE 445 Final Report - Fall 2022

Project # 16

Adish Patil, David Hahn, Forrest Hare

TA: Li, Qingyu

Abstract

The Poker Chip Counting Companion is a product designed to accurately count chips and reduce the inconveniences of setting up and concluding a game of poker. With this device, playing a game of poker with your friends becomes a more fair and smoother experience. With a simple mobile application, users can easily input information (chip values and player buy-ins) and receive the correct chip distribution from the machine. At the end of the game, players can deposit their remaining chips in the machine and have their buy-out instantly presented on the app. Players no longer have to hand-count chips or calculate how much they're owed at the end of a game. Our product takes care of this and lets players enjoy the game of poker!

Contents

1. Introduction

1.1 Team Project Overview

The game of Poker is a family comparing card game, in which players wager over the best hand; a hand is an order or arrangement of cards [1]. Since the beginning of the 20th century, this game has increased in popularity, professionally and, more importantly, recreationally. A traditional game of Poker is simple to play regarding the equipment needed; all players need is a case of chips and a deck of cards. However, the setup and conclusion are the greatest barriers to playing a game. Before, during, and after every Poker game, players must hand count different colored chips that equate to different cent/dollar values. Hand counting chips when setting up a poker game is a cumbersome task that is time-consuming and can lead to mistakes. In addition, calculating the exact distribution of chips and buyouts for multiple people can result in errors such as players receiving too little or too many chips or the wrong payout.

Our project aims to solve all of the traditional problems of setting up and finishing a game of Poker. The Poker Chip Counting Companion has several features: First, it accurately dispenses poker chips based on user inputs about the game, such as the buy-in and values of each color chip (Dispensing State). The device also calculates the appropriate amount of each color chip required, which takes the guesswork out of figuring out the proper chip stacks when starting a game. At the end of a game, the device switches to an operating buy-out mode (Collection State), which correctly counts the remaining stack sizes of each player. The Power Subsystem will power the entire machine. All user interaction will occur through the phone app subsystem, connected to the device via a Bluetooth connection, where a user can input information, control the machine's state, and receive information about the game.

Ultimately, the Poker Chip Counting Companion reduces the time needed to set up and end a poker game, increasing their willingness to play. Furthermore, it correctly counts chips so the players can guarantee they receive the proper amount based on their buy-in. Finally, the phone application makes it easy for the consumer to interact with the machine.

Many aspects make our product marketable. With a construction cost under $150 and no major existing competitors, our product is suitable for any poker player in the consumer market. Unfortunately, current technology servicing this problem only exists in high-budget casinos. Nothing of the sort except standard poker chip containers exist on consumer marketplaces. Additionally, the average poker player needs help understanding the rules behind distributing

chips or buyouts; the companion can reduce that barrier to entry by doing all the grunt work and allowing players to focus on the gameplay itself.

1.2 Visual Aid

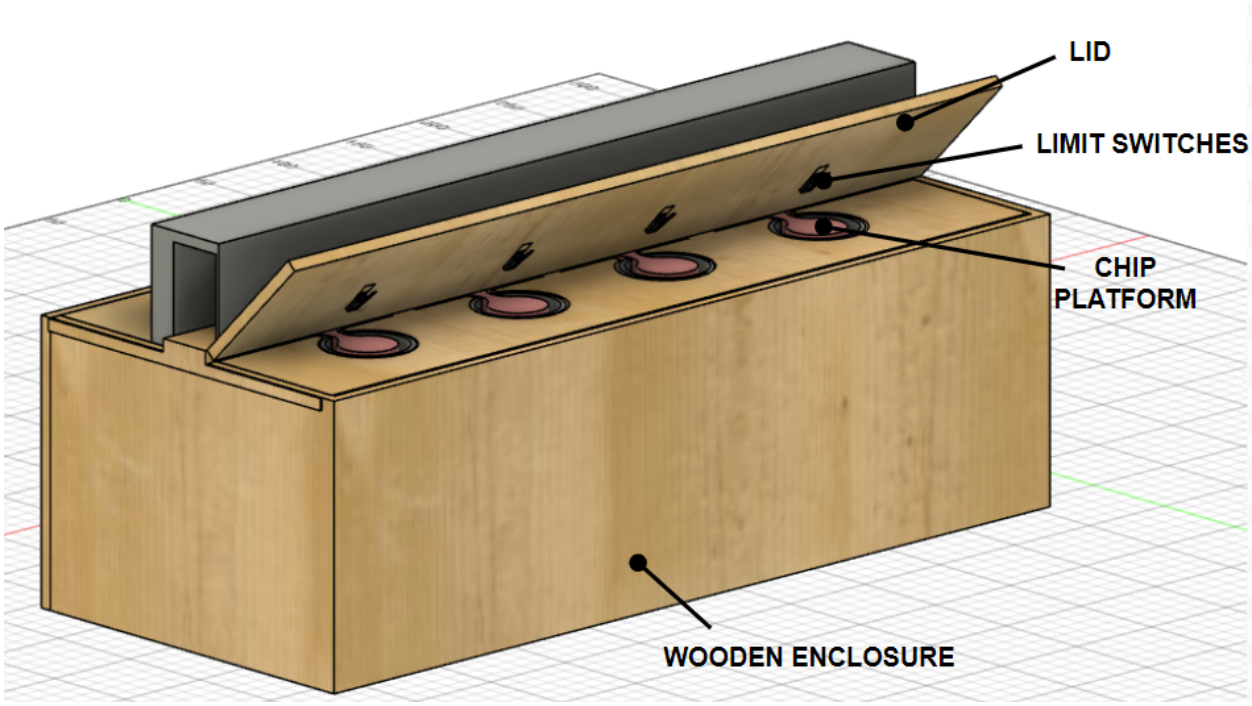


Figure 1: Poker Chip Counting Companion (North-East View)

The chip counting machine is fully enclosed within a wooden box, with a lid on top attached to a hinge. The overall dimensions of the box are 19” x 8” x 5.6”. The machine is paired to a phone application and operates based on information sent and received via Bluetooth.

1.3 High Level Requirements

To consider our project successful, our device will fulfill the following:

- The device has a capacity for at least 100 poker chips and dispenses them at a rate of 5 chips per second or faster.
- A phone application that receives and relays information to the machine with a response time within 800*ms* or faster.
- At the end of a game, when the users have deposited their remaining chips into the device, it counts their chips under 20 seconds.

2. Design

2.1 Physical Design

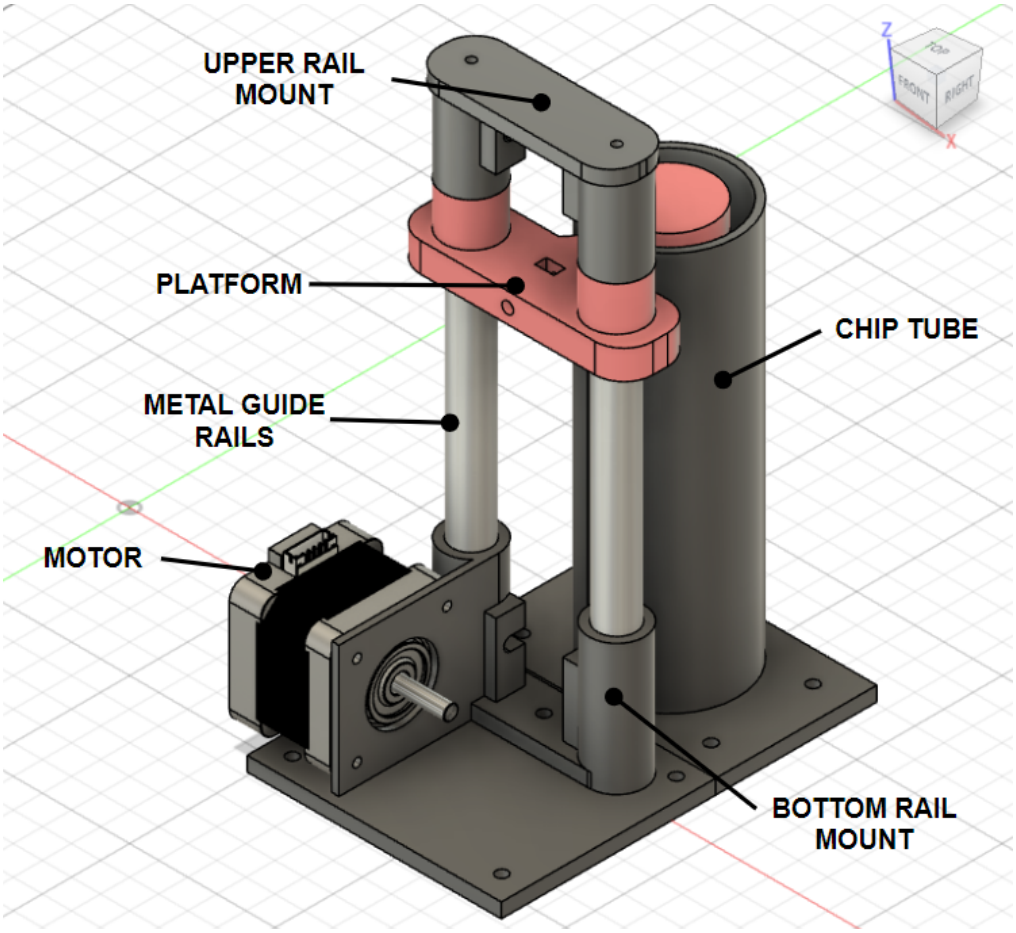


Figure 3: Elevator System (Side View)

This side view (Figure 3) shows the belt driven elevator system for raising and lowering the platform. Part of the platform and chips are enclosed within a tube so that the chips are stacked neatly.

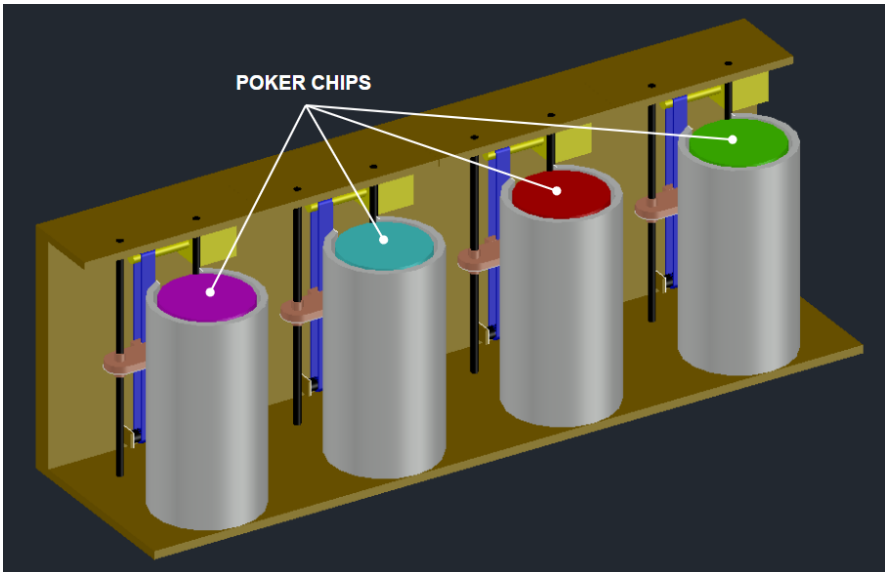


Figure 4: Open Enclosure (North-East View)

Our design consisted of 4 identical elevator systems which each correspond to a different stack of colored chips. As shown in Figure 4, there are 4 chip colors which in a game of poker correspond to different cent/dollar denominations.

2.2 Block Diagram

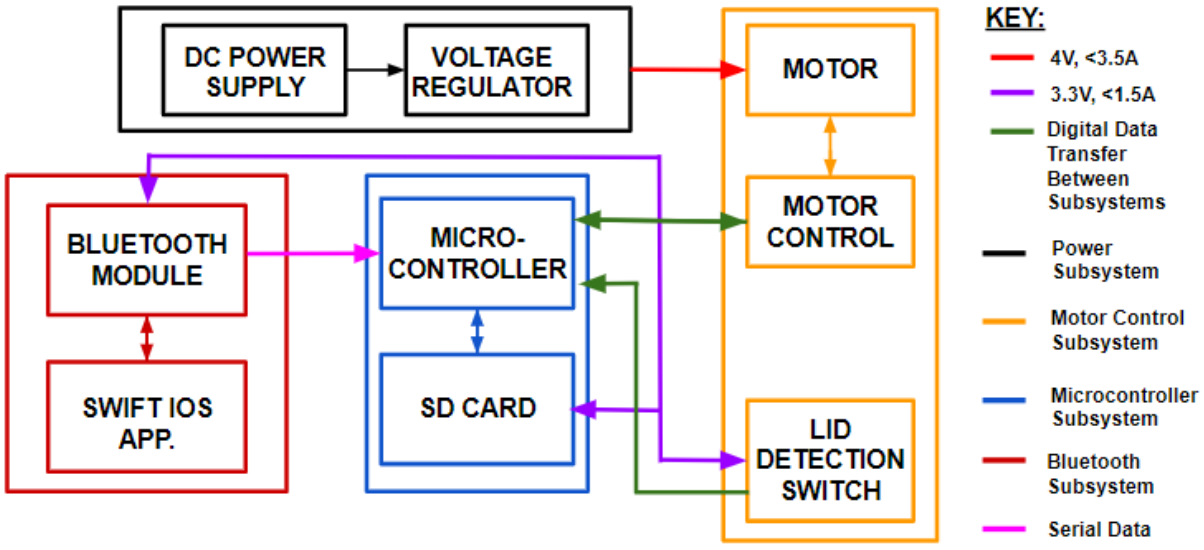


Figure 5: Block Diagram of Poker Chip Counting Companion

2.3 Subsystems

2.3.1 User Mobile Application

System Design

For the mobile application, we decided to pursue Swift IOS development because our group uses iPhones. Within Swift, we used the new SwiftUI framework that allows the creation of an easy and more user-friendly UI. Figma allows the easy transferability of high-fidelity design prototypes to code in SwiftUI.

Swift IOS development was the best implementation choice because it is simple to implement Bluetooth Low Energy (BLE) Communication with the Apple Core Bluetooth framework. In addition, the framework allows for more control over performance, such as data speed and advertising data [1].

With the Core Bluetooth framework, we planned to use the Generic ATTribute Profile (GATT) structure to communicate information between the BLE module and our application. The BLE GATT structure is set up to organize the information you want to be sent between devices in a step-down model [2]. The broadest type of structure is a profile, a "collection" or services (a subset of the profile). There are lists of characteristics within services that contain data being sent and received. That is how the structure of communication works. Based on our research, we concluded that a suitable BLE module for our project was the HiLetgo ESP-WROOM-32.

User Journey & Functionality Flow Diagram

The app design process began with the completion of a user journey flow diagram that highlighted the functionality of our desired application. It represented the decision/result tree of how the user would interact with the application from start to finish; this includes an in-depth analysis of the dispensing and collection states. Finally, the diagram also showcased the application's data flow in, within, and out.

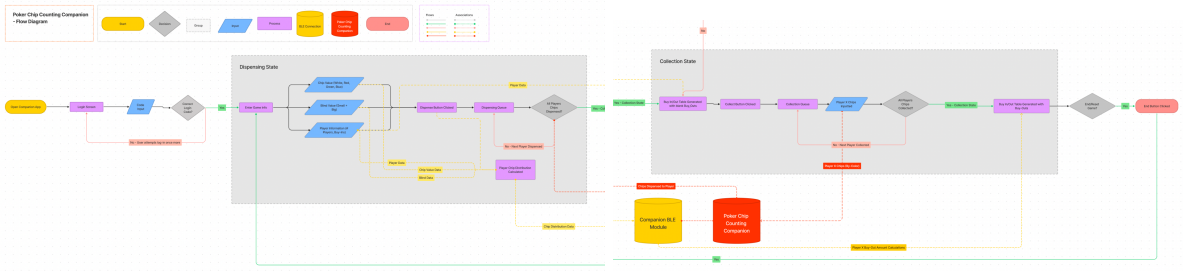


Figure 2: User Application Flow Diagram

(Press this link for the diagram)

Low & High Fidelity Designs

The app UI design process began with basic sketches. Initially, we focused on the information the user would have to input, how they would interact with the dispensing/collecting process, and how they would view their buy-outs.

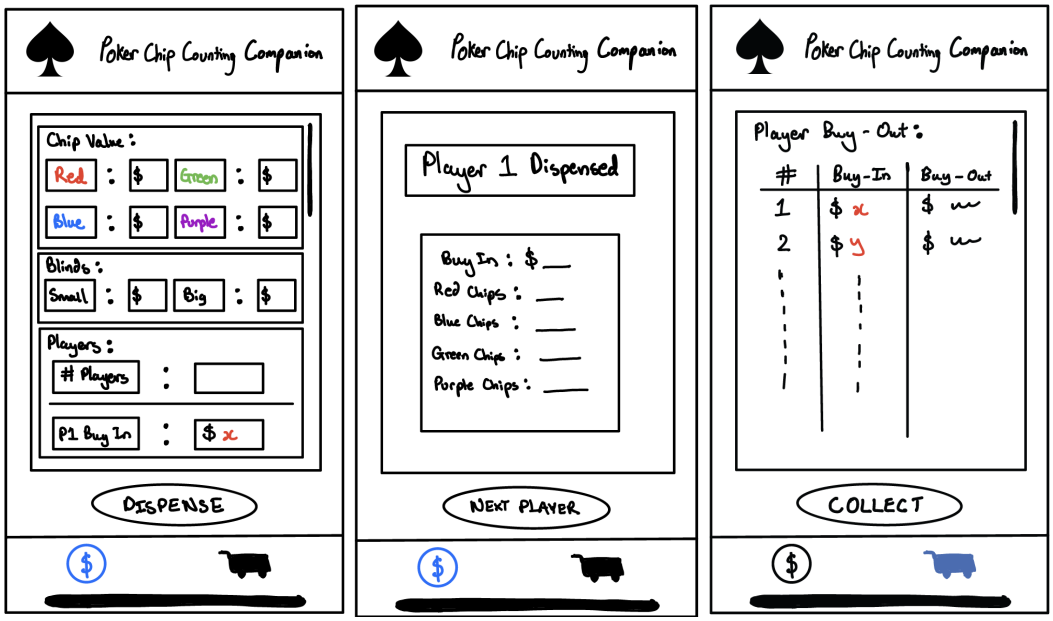


Figure 7: User Application Low-Fidelity Sketches

After completing these sketches, we began thinking more about how we wanted users to interact with the application. This meant considering questions such as: how should they enter their information? How should they tell the machine when they need their chips collected? The answers to these questions guided the high-fidelity prototypes we created on Figma. These are shown below.

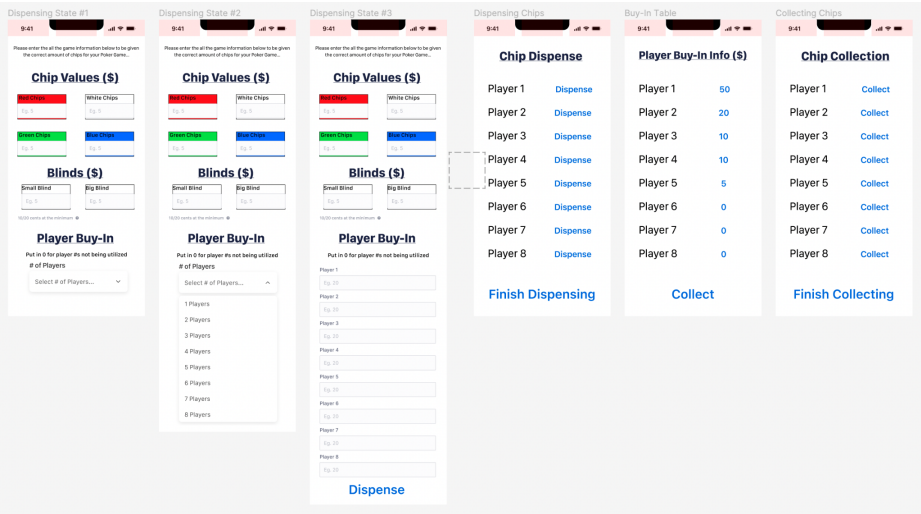


Figure 8: User Application High Fidelity Prototypes

2.3.2 Power

Power needs to supply both the motors and the digital components such as the BLE unit and inverters. It was originally thought the motors needed to be run at a higher voltage compared to the rest of the circuit, but we found that the motors run best in a low voltage and high current environment. We found the ideal operating conditions to be around 3.5V and 3.8A for the motor with the digital components requiring 5V for proper behavior. The motors can be damaged with currents exceeding 1.5A but for the motors to be damaged the machine would need to pull more than 6 A, which most power supplies cannot provide. Since it would be difficult to achieve 6A, much of the circuit protection circuitry was not needed for the functioning of the device. The microcontroller can produce the 5V needed to run the other digital components.

2.3.3 Microcontroller and Bluetooth

The microcontroller handles the communication and control signals of the circuit. We decided to purchase a combined microcontroller and Bluetooth unit in the ESP32. This unit was able to handle the BLE communications and was also able to generate the necessary motor signals. Data from the user inputs in the app are communicated through Bluetooth, this data is then processed by the microprocessor and converted into motor signals.

2.3.4 Motor Control System

Stepper motors require four signals in order to properly rotate; the first signal is a square wave at 50% duty cycle with its minimum being 0V (low), and the maximum being the operating voltage (high). The second signal is the first signal inverted, meaning when the first signal is

high the second signal is low and vice versa, the frequency of this square wave determines how quickly the motor can make steps. The third signal is the same as the first signal but delayed by 90°. The fourth signal is the inverted version of the third signal. The microcontroller is limited in the number of pins one can use. To produce all of the needed signals we would need 16 pins, to reduce this we used the microcontroller to produce the first and third signals and used an inverter to produce the second and fourth signals. The microcontroller is unable to produce enough current to run the motors so H-bridges were used to a higher-power version of the signals produced by the microcontroller and inverter chips. This allowed the microprocessor which operates under low power conditions to be able to handle the motors that operate at higher power.

2.4 Tolerance Analysis

Communications:

Transmitting all user data to the processor via the Bluetooth module can take around 200ms. It could take an additional 200ms for the processor to handle the data and then another 400ms for the motor control to communicate with the motors, leading to a 800ms delay with an acceptable tolerance range of 20%.

Control:

The microprocessor is able to perform at 9600 operations per second. This value should not change much, but there can be some more time delay between memory and the microprocessor; this delay will be on the order of microseconds so synchronous read and write operations should be possible.

Motor:

The stepper motor is able to move 1.8° ∓ 5%, which will correspond to a fraction of the height of a poker chip. This will ensure that the motor has enough accuracy to raise the chips within a 1-chip height accuracy.

Furthermore, the motor must be able to lift and lower a stack of 50 poker chips. The weight of 50 chips is about 600 grams. According to Newton's second law, the amount of force that each stack will exhibit is 5.88 Newtons. We can then calculate the torque requirement for the motor:

$$Torque = F * r/(n)$$

$$r = Radius\ of\ drive\ pulley + Length\ of\ platform\ arm\ (m) = 4.6\text{cm}$$

$$n = Efficiency\ of\ belt\ system = \sim 90\% = 0.9$$

So, Torque required = 30 N-cm

The stepper motor can provide a maximum torque of 42 N-cm so these motors will work well for our 30 N-cm requirements.

We also had to determine the number of steps that the motor needs to make. This was found by taking the diameter of the wheel pulley and using this value to determine the change in height of the chips for each step of the motor. We know that each step of the motor results in a movement of 1.8°. The diameter of the wheel pulley was found to be 9.6 mm. We measured the average height of a poker chip and found it to be around 3.3 mm. Using these values we can determine the number of steps we need to take. $\Delta h = 9.6\pi(\frac{1.8}{360})$, $N = 3.3/\Delta h = 21.88$. This gave us a value to start off with, then we adjusted it later based on our observations to better match the experimental values.

3. Design Verification

3.1 User Mobile Application

3.1.1 Building Responsive UI

Once the high-fidelity prototypes were complete, we started creating them on Swift using their new framework, SwiftUI. SwiftUI uses declarative syntax, so you can state what your UI should do [3]. Xcode (Swift IDE) provides intuitive design tools that make building UI easy. As we code, everything we edit can be viewed entirely in sync. The code is instantly visible as a preview on the right side of the editor. As we implemented the prototypes in Swift UI, Xcode instantly recompiled our changes and inserted them into a running app version. A sample result of the built-out UI is in the image below.

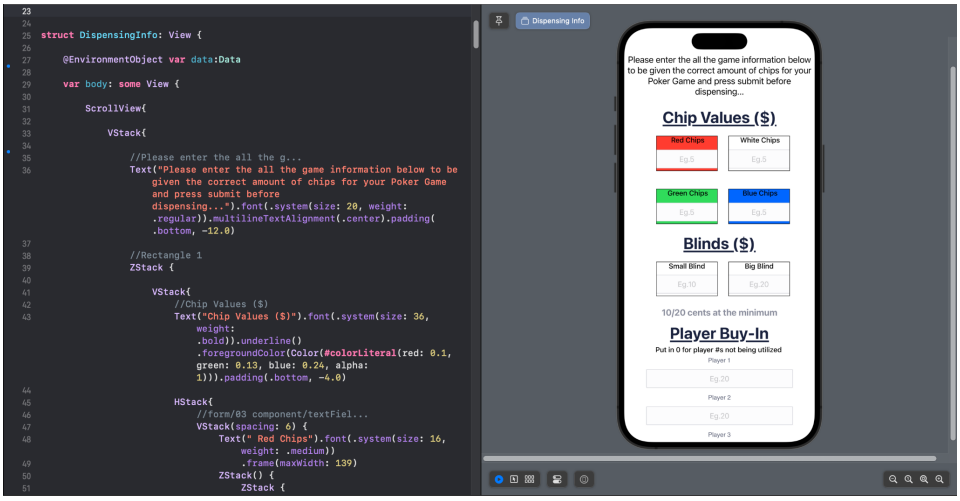
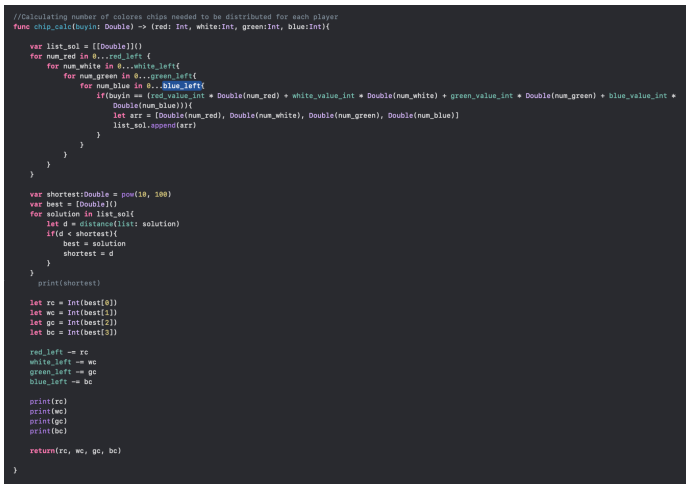


Figure 5: User Application UI Coding and Testing (Preview) Example

User Responsiveness was established in two methods: user input and buttons. Buttons either called functions or link different screens to one another by implementing the NavigationLink() function within the NavigationView Class [4]. The action within this button would display another view (a term for a screen) on the user's iPhone. The primary user inputs were in the form of numbers (Chip Values, Player Buy-Ins). The values were stored in a @Published property wrapper within a @ObservableObject. The @Published wrapper allows us to create variables that indicate the overarching data object when its variables changes [5]; in our case, it's when a user inputs a value. In addition, an ObservableObject protocol creates classes that can store data [6]. With this structure, we can now have a user input their game information and keep it for further calculations and Bluetooth communication with the machine.

3.1.2 Implementing Chip Dispensing Algorithm

Our team needed to determine how we would calculate the optimal amount of colored chips each player receives from the machine based on their buy-ins. To accomplish this, we developed an algorithm that considered the value of each colored chip (highest to lowest). Additionally, this algorithm considered the full capacity of the machine (25 chips/color), and ensured the total dispensed chip amount would not exceed 100. Additionally, these values were formatted into two-digit hex in a concatenated string to be sent to the machine for simplicity.



```
//Calculating number of colores chips needed to be distributed for each player
func chip_calc(buyin: Double) -> (red: Int, white:Int, green:Int, blue:Int){

    var list_sol = [[Double]]()
    for num_red in 0...red_left {
        for num_white in 0...white_left{
            for num_green in 0...green_left{
                for num_blue in 0...blue_left{
                    if(buyin == (red_value_int * Double(num_red) + white_value_int * Double(num_white) + green_value_int * Double(num_green) + blue_value_int *
                        Double(num_blue))){
                        let arr = [Double(num_red), Double(num_white), Double(num_green), Double(num_blue)]
                        list_sol.append(arr)
                    }
                }
            }
        }
    }

    var shortest:Double = pow(10, 100)
    var best = [Double]()
    for solution in list_sol{
        let d = distance(list: solution)
        if(d < shortest){
            best = solution
            shortest = d
        }
    }
      print(shortest)

    let rc = Int(best[0])
    let wc = Int(best[1])
    let gc = Int(best[2])
    let bc = Int(best[3])

    red_left -= rc
    white_left -= wc
    green_left -= gc
    blue_left -= bc

    print(rc)
    print(wc)
    print(gc)
    print(bc)

    return(rc, wc, gc, bc)
}
```

Figure 9: Chip Calculation Algorithm & Hex Formatting Code

3.1.3 Core Bluetooth Implementation

Using the Core Bluetooth framework provided by Apple [7], we implemented all the general classes and objects needed to create a connection and communication channel between our application and the BLE module within a BLE Manager Swift file. We made a CBCentralManager object with functions that scans for, discovers, connects to, and manages peripherals [8]. Furthermore, we created a CBPeripheral object representing remote peripheral devices that our app discovers with a central manager; we modified the scan function to connect automatically to the Poker Chip Counting Companion when found. The CBPeripheral object is used to discover, explore, and interact with the services on the BLE module. We can further interact with characteristics that provide details about a peripheral's service; this can be data from the machine.

```
struct Peripheral: Identifiable {
    let id: Int
    let name: String
    let rssi: Int
}
class BLEManager: NSObject, ObservableObject, CBCentralManagerDelegate, CBPeripheralDelegate {

    var myCentral: CBCentralManager!
    var discoveredPer: CBPeripheral?
    var transferCharacteristic: CBCharacteristic?
    var readCharacteristic: CBCharacteristic?
    @Published var isSwitchedOn = false
    @Published var peripherals = [Peripheral]()

        override init() {
            super.init()

            myCentral = CBCentralManager(delegate: self, queue: nil)
            myCentral.delegate = self
        }
    func centralManagerDidUpdateState(_ central: CBCentralManager) {
         if central.state == .poweredOn {
            isSwitchedOn = true
         }
         else {
            isSwitchedOn = false
         }
    }
    func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNumber) {
        var peripheralName: String!
//        print(peripheral)

        if let name = advertisementData[CBAdvertisementDataLocalNameKey] as? String {
            peripheralName = name
        }
        else {
            peripheralName = "Unknown"
        }

        let newPeripheral = Peripheral(id: peripherals.count, name: peripheralName, rssi: RSSI.intValue)
//        print(newPeripheral)
        if(peripheralName != "Unknown"){
            peripherals.append(newPeripheral)
        }

        if peripheralName == "Poker-Chip-Counting-Companion" {
            discoveredPer = peripheral
```

Figure 11: Bluetooth Manager Code Snippet

In addition to the Bluetooth Manager code, we created an additional helper function that wrote and read data from the peripheral device (machine). The write function sent our concatenated hex string, while the read function read in an integer.

```
/*
 *  Write some test data to peripheral
 */
func writeOutgoingValue(data: String){
          print("write")
        let valueString = (data as NSString).data(using: String.Encoding.utf8.rawValue)

        if let peripheral = self.discoveredPer {

          if let transferCharacteristic = transferCharacteristic {
            peripheral.writeValue(valueString!, for: transferCharacteristic, type: CBCharacteristicWriteType.withResponse)
            }
        }
}
```

```
/*
 *  Start reading for test data from the peripheral
 */
func readValue(){
    if let peripheral = self.discoveredPer {

      if let readCharacteristic = readCharacteristic {
        peripheral.readValue(for: readCharacteristic)
        }
      }
}
```

Figure 12: Bluetooth Write/Read Data Function Code

3.2 Power

3.2.1 Motor Power

All motors were able to run and could move all of the chips in a timely manner, and the motors were not damaged.

3.2.2 Digital Power

All of the IC chips and the microcontroller functioned properly and were not damaged.

3.3 Microcontroller and Bluetooth

3.3.1 Reading Data and Writing Data

In order to test the communication between the Bluetooth module and Bluetooth unit an instruction from the app was sent to the machine and the data was printed on a serial monitor on a laptop. The duration between the user hitting the send button and the data appearing on a serial monitor was timed with a stopwatch and recorded for five trials in the Reading table.

To test the writing speed, a predetermined value was sent to the user app and the Bluetooth characteristics were then printed once it was found. The time in between the send and receive was measured with a stopwatch and was recorded in the Writing table.

Reading

Trial	Time (ms)
1	8
2	8
3	12
4	8

Writing

Trial	Time (ms)
1	12
2	12
3	16
4	12

5	12
Average	9.6

5	16
Average	13.6

3.4 Motor Control System

3.4.1 Dispense

To determine the dispense speed of our device, the duration for the device to present 5 chips was measured. This was repeated five times in order to determine how accurately the motor can move chips.

Trial	Number of Chips Requested	Time (ms)	Number of Chips given	Chips per Second
1	5	200	5	25
2	5	200	5	25
3	5	230	5	21.73
4	5	200	5	25
5	5	230	5	21.73
			Average:	23.692

3.4.2 Collect

The machine was put into the collection state and an arbitrary number of chips were deposited to be counted. The duration between the first motor moving and the last motor stopping was the desired timeframe. All four motors were involved in this test and we concluded that our device successfully counted the buy-outs under our 20 second high level requirement.

Trial [Motor 1: Red]	Number of Chips Placed	Time (s)	Chips Counted
1	15	3.65	15
2	8	8.5	8
3	8	7.1	8
4	8	6.2	8
5	8	4.4	8

4. Costs

4.1 Labor

The average starting salary for CE/EE graduates is $80,000/year or $40/hour [2].

We estimate that each team member will dedicate 100 hours to this project.

Using these values, we can calculate an estimate for the total cost of labor.

$$100\ hours\ \times \frac{\$40}{hour} = \$4{,}000/\text{person}$$

$$\frac{\$4{,}000}{Person}\ \times\ 4\ Group\ Members\ \times\ 2.5\ Overhead\ =\ \$40{,}000\ for\ labor$$

4.2 Parts

Description	Manufacturer	Part Number	Quantity	Unit Price ($)	Cost ($)
PLA Filament	Sunlu	23260000	1	20.7	20.7
Stepper Motor	TwoTrees	17HS4401S	4	10.99	43.96
1/8 in. x 48 in. Round Rod	Everbilt	801567	2	4.38	8.76
Timing Belt Pulley	Zeelo	ZR-19090203	1	14.99	14.99
1/4 in. x 2 ft. x 4 ft. MDF	Home Depot	1508104	3	10.49	31.47
H Bridge	Bridgold	L293 L293D	4	0.81	3.24
Power Supply	Facmogu	5V 4A	1	12.99	12.99
BLE Microprocessor	HiLetgo	ESP-WROOM-32	1	10.99	10.99
				PARTS TOTAL:	147.1

Figure 12: Parts List

4.3 Grand Total

$$Total\ cost\ =\ Cost\ of\ Labor\ +\ Cost\ of\ Parts\ =\ \$40{,}000\ +\ \$139.83\ =\ \$40{,}147.1$$

5. Conclusion

5.1 Accomplishments

We were able to create a working machine and meet all of our high level requirements. On average the reading speed was measured to be 9.6ms This is 8,300% faster than our HL requirement of 800ms. Furthermore, we calculated our writing speed to be 13.6ms, which is 5,882% faster than our HL requirement of 800ms. We also measured our dispensing speed to be 23.69 Chips per second on average, this is 473% faster than our HL requirement of 5 Chips per second. We also found all of our collection times to be under 20s with the max being 8.5 seconds.

5.2 Uncertainties

There were a few uncertainties left at the end of this project. The first being the full performance of our PCB. Due to unforeseen circumstances, our PCBs were ordered late by the department. Additionally, our 4th order was mishandled and lost. Hence, we had to rely on releasing our final product using a PCB we ordered ourselves during Thanksgiving break. Because of this, we did not have ample testing time with our new PCB. Despite this, we are still confident in the entire machine's performance.

The delay of our PCB orders stagnated our project's progress. Without a proper PCB, we couldn't test all functionality. This led to the delayed order of our power supply, which was delivered after our final demo date. Hence, the uncertainty is that we don't know the performance of our machine with a proper power supply.

5.3 Ethical & Safety Considerations

5.3.1 Accidental Misuse - Child safety:

Issue - A child could accidentally place their hand inside the device and get their hand pinched by the motor and motor stage.

Code of Ethics Breach - With this issue, we are subject to breaking I.1 of the IEEE Code of Ethics that says "to hold paramount the safety, health, and welfare of the public…"[6]. We would also be subject to breaking 1.2 of the ACM Code of Ethics that instructs computing professionals to "Avoid harm" and not inflict physical or mental injury.

Solution - The device & housing was fully enclosed to avoid pinch spots so that in the event of a child being near it, the device would not harm them.

5.3.2 Harmful Implications - Gambling Addiction

Issue - The Poker Chip Counting Companion strives to make the Poker gameplay experience as seamless as possible. With this, we run the risk of creating gambling addictions for those who use the companion to a high extent.

Code of Ethics Breach - With this issue, we are subject to breaking I.1 of the IEEE Code of Ethics that says "to hold paramount the safety, health, and welfare of the public…"[8]. We would also be subject to breaking 1.2 of the ACM Code of Ethics that instructs computing professionals to "Avoid harm" [9] and not inflict physical or mental injury. Gambling addiction is a serious mental safety concern.

Solution - We put a disclaimer in the phone App that lists the National Problem Gambling Helpline Network hotline (1-800-522-4700).

Figure 14: Application Homescreen

5.4 Future Work

One improvement to be made is to run all of the motors at the same time instead of each motor being run sequentially. Another issue that we had was that the poker chips have some variation in height that lead to some errors so making more precise poker chips would lead to better results for the user. Having a custom power supply would also lead to better results.

Another improvement that could be implemented in the future is the ability to buy-in during the middle of a game (re-buy), instead of only at the start. With re-buys, new players could enter the game while it is in play.

5.5 Additional Deliverables

User Mobile Application Github - https://github.com/adishsp7/Poker-Chip-Counting-Companion

References

[1] “Core Bluetooth Framework.” *Apple Developer Documentation*, Apple, https://developer.apple.com/documentation/corebluetooth.

[2] Teel, John. “How to Develop a Mobile App That Communicates with Your Product Using Bluetooth.” *PREDICTABLE DESIGNS*, Predictable Designs LLC., 12 Jan. 2022, https://predictabledesigns.com/how-to-develop-a-mobile-app-that-communicates-with-your-product-using-bluetooth/.

[3] Inc., Apple. “SwiftUI Overview - Xcode - Apple Developer.” *Overview - Xcode - Apple Developer*, Apple, https://developer.apple.com/xcode/swiftui/.

[4] “Migrating to new navigation types.” *Apple Developer Documentation*, Apple, https://developer.apple.com/documentation/swiftui/migrating-to-new-navigation-types.

[5] Hudson, Paul. “What Is the @Published Property Wrapper?” *Hacking with Swift*, Hacking with Swift, 9 Feb. 2021, https://www.hackingwithswift.com/quick-start/swiftui/what-is-the-published-property-wrapper.

[6] Hudson, Paul. “How to Use @Observedobject to Manage State from External Objects.” *Hacking with Swift*, Hacking with Swift, 3 Sept. 2021, https://www.hackingwithswift.com/quick-start/swiftui/how-to-use-observedobject-to-manage-state-from-external-objects.

[7] “Transferring Data Between Bluetooth Low Energy Devices.” *Apple Developer Documentation*, Apple, https://developer.apple.com/documentation/corebluetooth/transferring_data_between_bluetooth_low_energy_devices.

[8] Beaton, Trevor. “Build a Bluetooth App Using Swift 5.” *Adafruit Learning System*, https://learn.adafruit.com/build-a-bluetooth-app-using-swift-5?view=all.

[8] ACM Code 2018 Task Force. (2018). *ACM Code of Ethics and Professional Conduct*. Code of Ethics. https://www.acm.org/code-of-ethics#:~:text=1.3%20Be%20honest%20and%20trustworthy,problems%20to%20the%20appropriate%20parties.

[9] IEEE Code of Ethics, IEEE, 2020. https://www.ieee.org/about/corporate/governance/p7-8.html

[10] *(PDF) Stepper Motor - Researchgate*. https://www.researchgate.net/publication/332835189_Stepper_Motor.

[11] -Website Author Syed Zain Nasir syedzainnasir I am Syed Zain Nasir. “ESP32 Ble (Bluetooth Low Energy).” *The Engineering Projects*, 22 Feb. 2022, https://www.theengineeringprojects.com/2021/11/esp32-ble-bluetooth-low-energy.html.

[12] AdminESC. “Ble in ESP32: Bluetooth Low Energy Connection.” *ElectroSoftCloud*, 16 Apr. 2021, https://www.electrosoftcloud.com/en/ble-in-esp32-bluetooth-low-energy-connection/.

[13] Alldatasheet.com. “ESP32 PDF, esp32 Description, esp32 Datasheet, esp32 View ::: Alldatasheet :::” *ALLDATASHEET*, https://pdf1.alldatasheet.com/datasheet-pdf/view/1148023/ESPRESSIF/ESP32.html.

[14] Bruni, Carlos, et al. “ESP32 Ble Server and Client (Bluetooth Low Energy).” *Random Nerd Tutorials*, 11 Nov. 2021, https://randomnerdtutorials.com/esp32-ble-server-client/.

[15] “The Code Affirms an Obligation of Computing Professionals to Use Their Skills for the Benefit of Society.” *Code of Ethics*, https://www.acm.org/code-of-ethics#:~:text=1.3%20Be%20honest%20and%20trustworthy,problems%20to%20the%20appropriate%20parties.

[16] *Hex Inverters Datasheet (Rev. C) - Texas Instruments*. https://www.ti.com/lit/ds/symlink/sn7404.pdf..

[17] Novichkov, Artem. “Bluetooth and SWIFTUI. Developing App for RGB Stripe Control.” *Teletype*, 1 July 2021, https://blog.artemnovichkov.com/bluetooth-and-swiftui.

[18] *SLRS008D –September 1986–Revised January 2016 L293X Quadruple Half-H ...* https://www.ti.com/lit/ds/symlink/l293.pdf.

[19] Wisintainer, Miguel, et al. “Esp32 Bluetooth Low Energy (BLE) on Arduino Ide.” *Random Nerd Tutorials*, 4 June 2019, https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/.

Appendix A Requirement and Verification Table

System Requirements and Verifications

Subsystem	Requirement	Verification	Verification status (Y or N)
User Mobile Application	1. BLE Module Setup 2. UI Designs 3. XCode and SwiftUI for IOS development	1. Application can be hosted and tested in IOS device (iPhone) 2. Ensuring BLE connection with Core Bluetooth for the collection and transmission of data between app and machine	**Y**
Power	1. Proper Power is given to each component	1. All parts Functional 2. Measured with Multimeter	**Y**
Microcontroller & Bluetooth	1. Able to Communicate accurately via BlueTooth 2. Able to Respond within a 800ms	1. Communication Test 2. Timing of communication	**Y**
Motor Control System	1. Able to move the Chips in small steps 2. Respond to microcontroller signals Quickly 3. Able to detect chips during the collection phase	1. Measure the amount the chips have been moved. 2. Time the motor response time 3. Test if the chips are able to be detected	**Y**