ELECTRONIC BETTING SYSTEM FOR POKER

By

Umang Chavan

Anand Giridharan

Varun Pitta

Final Report for ECE 445, Senior Design, Spring 2019

TA: Nicholas Ratajczyk

01 May 2019

Project No. 62

Abstract

This paper outlines the motivation, design and testing process that went into the creation of a fully electronic and chip-free betting system for poker. The final product is meant to have all the capabilities of a normal poker game such as raising, folding, checking, and calling, allowing for a fully functional game of poker in both in-home and professional settings. The final system consists of N-player devices and one central hub device, both of which are integral to the system to work as intended.

Contents

1. Introduction

1.1 Objective

Texas Hold'em is a variant of the card game of poker. Each player in the game is dealt two cards face down and the dealer goes through three rounds of community betting flipping over 5 cards in total. Each player then makes bets on each round based on the strength of their hand. Each player in the game starts with the same amount of money to use while betting. The money is represented using poker "chips," which are small discs that are often made of plastic or clay. Without these chips, playing poker becomes very difficult.

If a group of people wish to play poker, they would need a deck of cards and a poker chip set. The issue is that not everyone owns a poker chip set and those who do only have a limited amount of chips. This restricts the number of people that can play and how much money can be bet. Poker sets can be rather expensive and sometimes if the chips are made from authentic clay, they can chip and break easily. The problem statement that we are addressing is to see if there is a more effective way of playing poker for cheap without worrying about chips.

Our proposed solution is an electronic betting system that completely eradicates the need of physical poker chips. The idea is to have a centralized unit where all betting happens, and each player can see the community pot. Each player will also have their own device that allows them to see their own money and make poker actions such as raising the bet or calling the bet. By taking away the need to use poker chips, everyday people can play poker without having to worry about the financial restraints and the game restraints.

1.2 Background

The idea of removing accessories from games and using an electric alternative has been around for a while. Monopoly Electronic Banking Edition by Hasbro eliminated the need for paper cash that is normally used in games. Instead they created a debit card system, where each player swipes a machine to perform all transactions with the bank or with other players. The purpose of this was to remove the hassle of having so much paper laying around. All the math and counting is done by the middle unit and the player only has to know what action they want to take. The debit cards in this game use magnetic strips to take care of player identification

Our goals are similar to this, but we also wanted to add the feature where each player knows the amount of money they hold as well as the community pot. Instead of using magnetic strips, our player identification will be handled with RFID readers. The hope for the end-product is that

it will be efficient enough to eliminate poker chips, be affordable, and still provide enjoyment to the game.

1.3 High Level Requirements

- Player should be able to join the game at any time and see the amount of money other players have as well as the amount of money in the community pot.
- Players should be able to raise, call, or fold from their device, and the central hub should display the chosen action.
- All devices including the central display must be powered with a battery pack. The battery pack must last the duration of the game

2 Design

2.1 Block Diagram

2.1.1 Central Unit Diagram

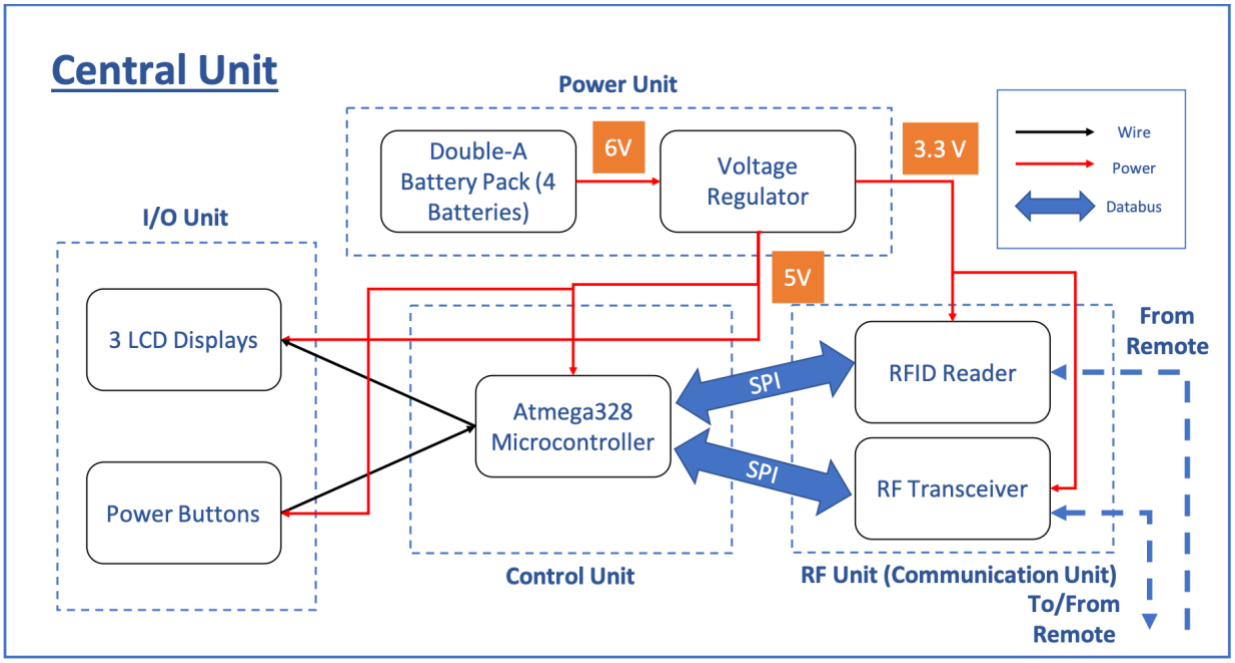

Figure 1: Block Diagram of Central Unit

2.1.2 Player Device Diagram

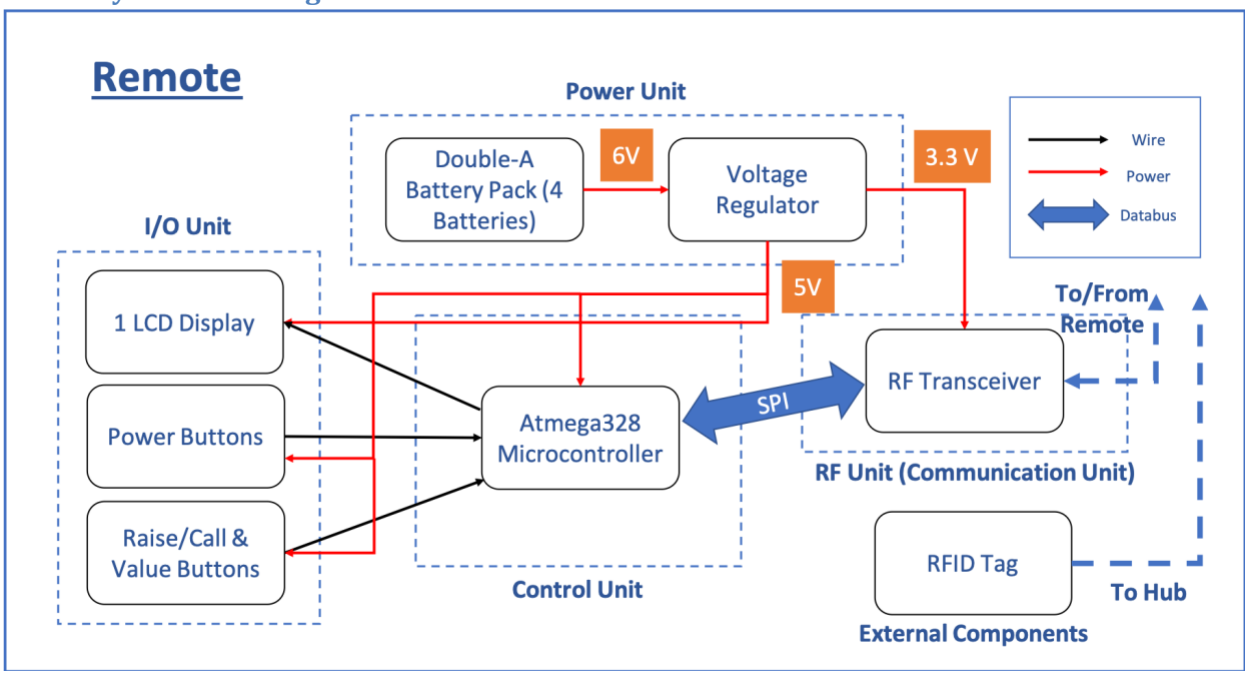

Figure 2: Block Diagram of Player Device

2.2 Physical Design

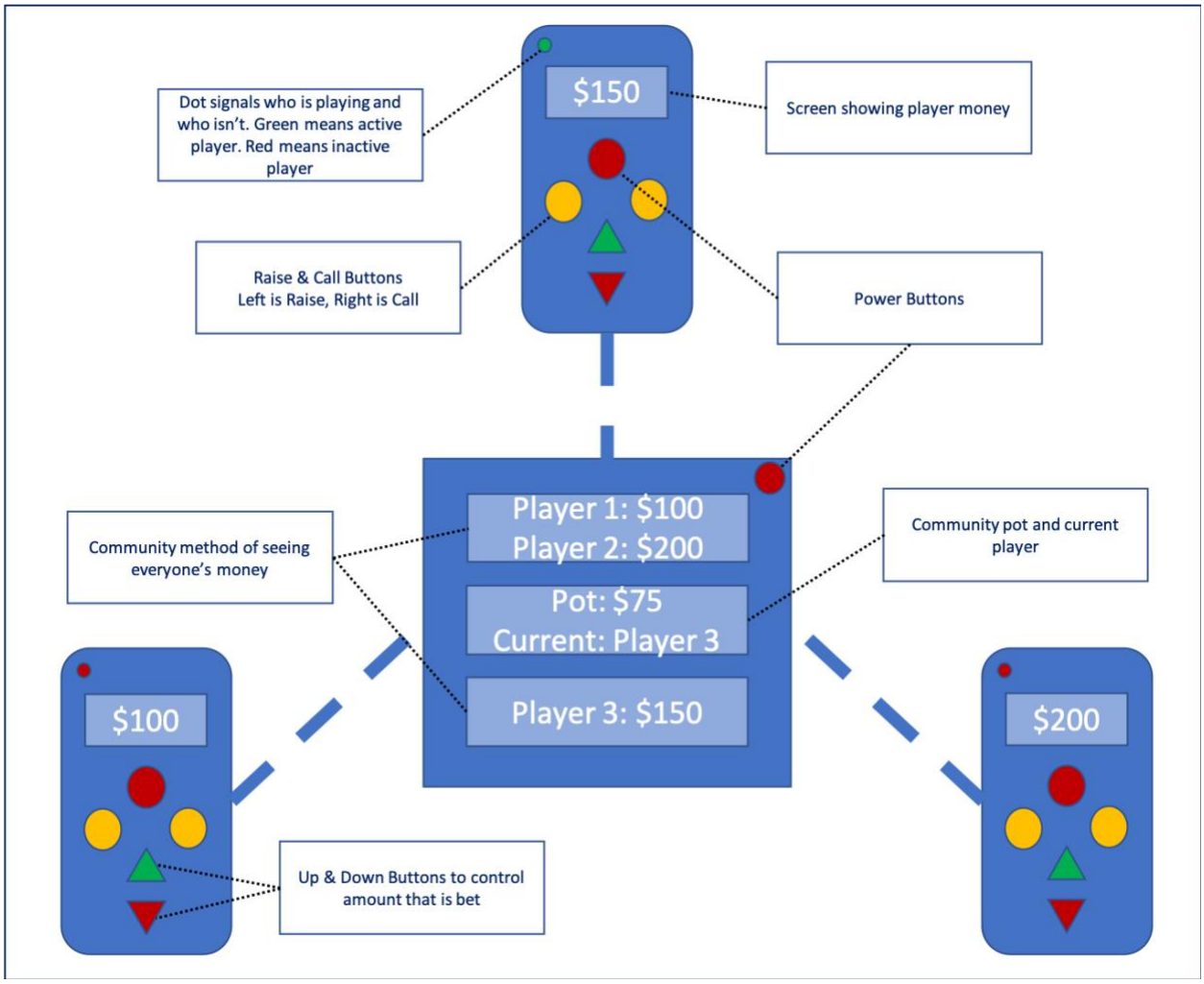

Figure 3: Physical Layout of Full System

Our physical design consists of essentially a main display device and the remote(s) for each player. On the main display we plan to display:

1) Each Players Total Money
2) Total Pot Money
3) Current Player

The main display will also contain a power button to turn the device on and off. For the remote(s) we plan to display:

1) Player's total money

We will also have various buttons and indicators such as:

1) Call push button
2) Raise push button
3) Up push button (to increase amount you'd like raise by)
4) Down push button (to decrease amount you'd like to raise by)
5) Power button

2.3 Functional Overview

2.3.1 Power Supply

We require a power supply for nearly all our units. We will be using a battery pack with four AA batteries for our power supply. The power supply unit will be the same in both the player device and the central hub. In both cases, the output of the power supply will be 6V. The reason why we went with the decision of using 4 batteries was because we needed the circuit to have a minimum amount of 1A flowing so that the transceiver and reader can use the microcontroller at the same time. We also needed to make sure the game lasted at least 2.5 hours which is the approximate duration of a poker game.

2.3.2 Voltage Regulator

We are using two voltage regulators, one for our RF transceiver and one for our microcontroller/LCDs. This is necessary to be able to step down our 6 V input (4 batteries) to 3.3 V max for our RF transceiver (STMicroelectronics LD1117AV33), and 5 V max for our microcontroller and LCDs (Sparkfun L7805). The 3.3V regulator was also necessary for the RFID reader which had the same approximate operating voltage as the transceiver. In order to keep the regulation consistent, we connected capacitors in parallel to minimize the noise in the circuit. If the voltage is not consistent, then there could be problems with the way our modules operated.

2.3.3 Microcontroller

We are going to use an ATmega328p microcontroller in order to manage storing data and data transfer, as well has handle signals from various control buttons (raise, call, power buttons). The microcontroller will be programmed via Arduino which uses UART and will read/write signals from the RF module via SPI. This is arguably the most important component of the design because all the microcontroller is responsible for communication, I/O, memory, and game state. Without a properly programmed and functional microcontroller, no parts of the design would work. It is important to note that each player device and the central hub all use their own microcontroller. In total, there are 4 microcontrollers used in the whole apparatus and each one uses its own memory. Instead of soldering the microcontroller to the PCB, we soldered a microcontroller socket to the board. This made it easier to program the ATmega328p because we wouldn't need to solder and unsolder it, all we needed to do was pop it out and plug it into the Arduino for new code pushes.

2.3.4 Power Button

Even though we call this a power button, it's really a switch that keeps the device either on or off. All the player devices and the central hub have these switches. The switches connect the power supply with the voltage regulators and begin the process of sending power to the rest of the circuit. This was the only button/switch that didn't have any use case in the software.

2.3.5 Call Button

This button will only be on the player device, and it will be used to send a signal to the central hub display via the RF module. The central hub display will then automatically update the call amount to the community pot. What's important to note for this button and the remaining buttons is that the buttons are only active when it is your turn in the game state. The player can tell if it is their turn by checking to see if the green LED on their player device is lit up. If it is, then their button inputs are taken into account. Otherwise, no amount of button presses will register anything. The call button will match what the highest bid in the current round is. It will subtract it from the player's total pot and start the process of sending the message to the central hub. Ideally, we would want this button and all the buttons to be on top of the encasing of the entire player device. However, since we did not get to the casing of the device, we soldered the buttons directly to the PCB, but in a fashion where it resembles the look of a controller.

2.3.6 Raise Button

This button will only be on the player device, and it will be used to send a signal to the central hub display via the RF module. The central hub display will then automatically update the call amount to the community pot. Very similar functionality to the call button, only difference is that instead of just subtracting the amount of the highest bid, this button is responsible for setting the highest bid. The game logic has it so that the player cannot raise below a certain "min bid" value, which is in line with the actual rules of poker

2.3.7 Up and Down Button

These buttons will only be on the player devices, and they will be used to control how much the player wants to increase or decrease the current bet. The up and down functions themselves increase and decrease the bid amount in increments of 0.25. The up button will not work if the bid amount the player chooses is greater than their available pool. The down button also ensures that the bid cannot go below 0.00.

2.3.8 Fold Button

This button will only be on the player device, and it will be used to send a signal to the central hub display via the RF module. The player remote will be disabled for the rest of the round, where round is when N-1 players have folded. This button is also extremely important in determining who wins the pot. When N-1 players have folded, there is only one player

remaining and they receive the winnings at the end of the round. One the winnings have been dispersed, then the next round begins. In the case that there is a standoff at the end of the round between two players, the winning player would have to wait until the losing player presses the fold button in order to get the winnings.

2.3.9 Central Hub Display

The central display device will contain three LCD displays, with two of them containing the current players, their current chip amounts. The third one will contain the community pot, the current player and the last move. The data being displayed on the LCD screens will be coming from the microcontroller storage. Each LCD display can use the same bus and pins for the incoming data. The enable pin of each LCD is the differentiating factor in determining what messages are being shown on the different screens. Initially, we were concerned that we weren't going to have enough pins on the microcontroller to connect all the LCD's. So, we implemented decoder and NAND logic to determine which screen is going to display certain information. After testing and verifying the logic components and ensuring that they worked, we soon realized that since the enable pin was the only differing pin across all the LCD's, we only needed 3 different pins on the microcontroller. Because of this, we scrapped the idea of having the logic circuits. Another design decision that we made was to not use a Serial operation in the code because these operations would interfere with the data transfer to the LCD's and would print random characters.

2.3.10 Player Device Display

The player device will have one LCD screen so that the user can see how much money he/she has left and how much money he/she wants to raise for the current turn. The data being displayed on the LCD screen will be coming from the microcontroller storage. The LCD displays on the player devices were easier to use because there was only one pin to worry about on the microcontroller for the enable.

2.3.11 RFID Tag

Our initial thoughts were to attach the RFID tag to the player device. However, since there was no encasing, we decided to give each player an RFID tag. The tag will be used to allow a player to successfully join the game and begin the mode of communication between the central hub and the player device. To join the game the player must scan his or her RFID to the RFID reader on the central hub. After the central hub gets all 3 tags saved in memory, it begins transmitting the ID to the players in the order that they tapped into the system. The tags are important for the communication between the player devices and the central hub because the ID's are used in the messages between both parts of the system.

2.3.12 RF Transceiver

The RF Transceiver is necessary for communication between the Player Device and Central Hub, as we must pass data to be displayed almost every I/O action. The model we will be using is the NRF24L01. This has a transmit power of 12mA and operates at a 1.9 - 3.6 V range. The frequency bandwidth in which this transceiver operates is 2.4 GHz. The amount of baud (how many times the signal changes per second) ranges from 250 Kbps to 2 Mbps. We chose this transceiver because it has all the qualities that we need in a communication module. The maximum distance of communication is 100 meters which is far more than what we need for a game of poker. The transceiver also allows communication with up to 6 other transceivers, or in our case, one central hub and 3 player devices. In addition to all of this, these transceivers were one of the cheapest RF options in the market. Finally, there is a really easy to use Arduino library that lets us incorporate the transceiver in code. This library is very abstracted with its hardware implementation and this is important because it led us to some critical design decisions for the RFID reader (see section 2.3.13). The transceiver module itself utilized the SPI bus of the microcontroller which eliminated another 3 pins on the microcontroller. Since this component is one of the most important, we figured this should have priority for real estate on the microcontroller.

2.3.13 RFID Reader

The RFID Reader serves the purpose of reading the individual player's RFID tags and allows them to enter the game. This will only be on the Central Hub and will only be used once by each player then the software will keep the rotation/turns of each player going. The RFID reader, like the transceiver, also connects to the SPI bus of the microcontroller. This led to one of our issues when we tested the reader and the transceiver together. Since both were using the same bus, when one was active, the other module wouldn't work and vice versa. Since we need both to work simultaneously, we had to come up with some changes in our design. So, in our software, we used manual enables and disables to either turn on or off the reader when we needed it. Another thing we needed was a large enough current so that both devices could work together. When we were testing the devices, we used the power coming out of the Arduino board. But when hooked up to a stronger current source, we noticed that both the reader and transceiver worked together. This ultimately led us to change the way our game state logic worked (see section 2.4).

2.4 Software – Setup and Game State Logic

While designing our game system, we knew that there were several steps that had to be taken in the software in order for the game to work. In order to solve the issue of the transceiver and reader not working concurrently, we changed the game setup so that they are being used sequentially. So in the game state, we first wait till all the players are tagged in, then we turn off the reader and let the transceiver take over for the remainder of the game. The flow diagram in

Figure 4 shows the game setup code. After the setup is done, then the game starts, and the rounds start beginning. It's important to note that there is no way of pathing an entire game of poker because each round is radically different than the next. But the overall logic of going to the next player, reading their input and reflecting the changes in the central hub is consistent. Figure 5 goes into greater detail about how the game state is maintained and the steps that go into that.

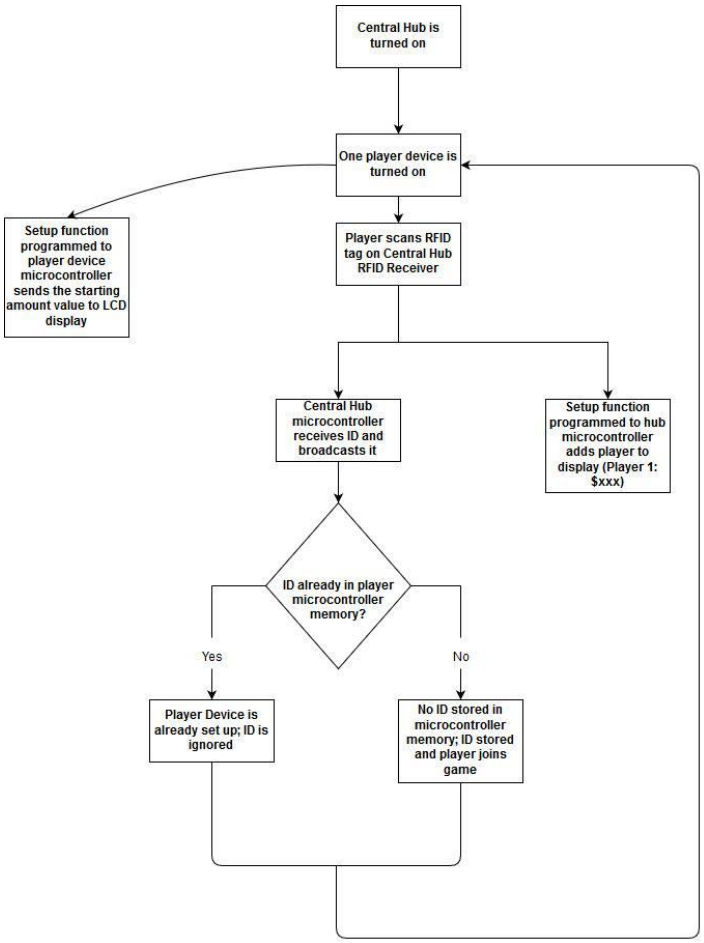

Figure 4: Game Setup Flow Diagram

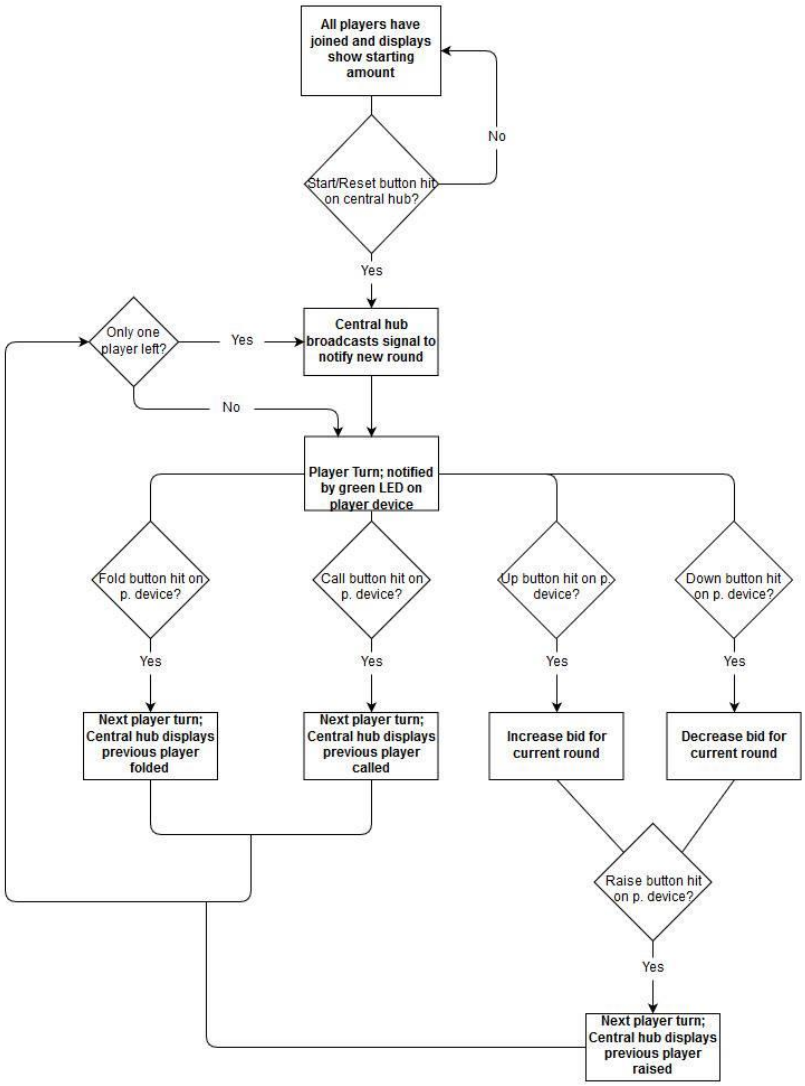

Figure 5: Game Logic Flow Diagram

2.5 Memory

The ATmega328p comes with 32k bytes Flash memory (0.5k is used for the bootloader, 2k bytes of SRAM, and 1k byte of EEPROM [8]. Flash and EEPROM are non-volatile, therefore they persist even if power is suddenly turned off [8]. In our game, some player could accidentally turn off their device, or the device could run out of battery. In this case, we still want to store all the relevant information on the microcontroller, so that we can access it after the power is on again.

On the central hub, the important values to keep track off are everyone's money and their bids, the amount in the community pot, whose turn is it, and who has folded. The good thing about the data we are storing is that we can generally assume that the casual player does not play in a game with over $1000, so our data does not consume a lot of space. The three players will have

a max of 3 digits for both their current money and bid, which is a total of 18 bits needed for those values among 3 players. We will also have three bits for the community pot, safely assuming that players will not exceed the amount. All players will be identified by a number, so keeping track of whose turn it is will only be one digit. Our prototype design will be for three players, so there can be a max two people that fold (according to our game logic) and therefore two bits for that as well. This brings us to a total of 18+3+1+2 = 24 bits that need to be stored. Even if we store each number as a character, we have 192 bytes minimum that need to be stored in the EEPROM out of 1k bytes. Additionally, the microcontroller will need to store the 96 bits of data for the RFID tag, and with three players, this is 288 bits of data, which is 36 bytes of data. In total, the microcontroller needs to store about 228 bytes of data to persist when the power turns off. This leaves a lot of room for expansions and even more data storage.

For each player device, the value that need to be stored is like what the central hub has to store, but it only needs to keep track of its individual money count, bid, whether it's the current player's turn and the RFID tag associated to the device. Therefore, there is significantly less than 228 bytes of data storage in the EEPROM needed to persist, which allows for more expansion for other data to be stored (personal names, etc.)

3. Design Verification

Many of our individual components that were utilized in our PCBs, both on the player device and on the central hub, were tested and verified according to the specifications given in the data sheets. More information about tests and their results can be found in Appendix A. In addition to single component tests, we also did various modular and integration tests with our subsystems, which we will go into deeper in the following section.

3.1 Mechanical Design

We envisioned our PCB to serve as a controller for the player device and we designed it as such. The central hub PCB was designed in a way to enable the most space for the external LCD displays that we required. We began verifications of the PCB connections one-by-one as we began integrating individual components. First, we soldered our battery pack and took a voltage and current reading by the switch on our PCB to ensure at least 6V (+/- 0.5V). Next, we placed in the voltage regulators and tested the input and output voltages for both. Although we had done individual tests of both voltage regulators separately via a breadboard, we wanted to ensure that there was no errant connection in our PCB that was accidentally sending the wrong voltage to another component. Once those tests were done, we placed in the microcontroller, LCD display(s) and the transceiver and ran a simple integration test in order to confirm proper connections. The test we ran was a “Hello World” test, in which we treated one device as a transceiver and another device as a receiver. We sent over the message and displayed the message on the LCD and verified proper connections once we saw the “Hello World” message displayed on the LCD with no errant or extra characters.

3.2 Software Modules

Our software modules required the most verifications due to the large amount of edge cases that we needed to handle. We ran tests individually for our setup logic, game state and transition logic and memory management.

3.2.1 Setup Logic

Within the setup phase of the game, the primary thing we had to ensure was proper communication between the central hub and the individual player devices. The setup flow is discussed in the software flow diagrams which are in the design section. Our main goal was to ensure that each player device received a unique id from the central hub, so that is how we set up our tests. First, we ran a few simple communication tests which dealt with increasing the number of players in the game. We started out with only one player, and then ran two additional tests (one with two players and the last one with all three players). Once we established that communication was working with the central hub and the player devices, we did our second set of tests. We sent three messages with the three ids from the central hub and

tested if the player devices each received one unique id. This condition was satisfied and completed the verification of our setup logic.

3.2.2 Game State and Transition Logic

All requirements of the Game State and Transition Logic were met and verified. The primary goals of this phase were to ensure that the communication at any given time was only between one transmitter and receiver, the player devices that were out of turn were rendered inactive, the game was able to transition from one player to the next (excluding any players that had folded), and the correct values would be updated and displayed on the central hub display as well as the individual player device. We handled edge cases such as disallowing the player to bet under the minimum bid (unless their total pot was under the minimum bid), to ensure that the game would follow poker rules and no incorrect values would be sent. Additionally, we incorporated an LED on each player device that would light up when it was that player's turn. For all other players, the LED would not be lit and their buttons would not change any of their local values, which met our requirements of only one active player at a time and player devices out-of-turn rendered inactive. When the up or down buttons were pressed on the player device of the active player, we would update the display to reflect the action pressed (increasing bid when up arrow is pressed, decreasing bid when down arrow is pressed). We handled the remaining updates right after a raise, call or fold button was pressed, after which we updated the display values locally and then sent the information over to the central hub for it to display the updated values. We ensured that the updated were handled first, before moving on to the next player or the next round. This met our requirement of displaying the correct values during the game.

3.2.3 Memory Management

The last component of our Software subsystem was Memory Management. We were able to meet most of the requirements we set for ourselves in this regard. The types of variables and values that needed to be stored both for the central hub and the player are described in further detail in the memory section. The main purpose of having memory management throughout the game was to ensure that the game could progress even if a player device or the central hub was discharged or accidentally turned off. We tested this functionality by turning off a device, or a combination of devices, during various points in our game state. For the most part, when we turned the devices back on, the game state progressed as expected. We ran into a few issues if a device was turned off during the setup phase, or during the update display phase. While we were able to address some of these issues, we were unable to fix all of them in the time that we had.

4. Costs

Since we are expecting to finish the entire project during the duration of the semester, we don't need to account for partial work calculations. We estimate our development costs to be $30/hour, 15 hours/week for 3 people. Therefore, our total development costs will be:

4.1 Parts

Part	Cost (Individual)	Cost (Project)
LCD Display (HD44780)	$2.71 (16x2) $5.61 (20x4)	$19.16
Microcontroller (ATMega328P)	$4.30	$17.20
RF Transceiver (NRF24L01)	$1.75	$7.00
RFID Reader (RC522)	$12.99	$12.99
Voltage Regulator	$0.95 (L7805) $0.54 (LD1117AV33)	$5.96
Logic Gates (SN74HC08AN)	$0.44	$1.76
Decoder (CD4555BEE4)	$0.46	$1.84
Resistors/Capacitors Pack	$15.90	$15.90
Battery Pack (4xAA)	$1.95	$7.80
Push Buttons	$0.25	$4.00
Potentiometer	$0.95	$3.80
16 MHz Crystal	$0.95	$3.80
Total		$101.21

Table 1: Cost per module

We are planning on building only one central hub unit and three player devices, therefore our total development costs including work hours is $21,701.21.

5. Conclusion

5.1 Accomplishments

We successfully were able to make a fully communicative electronic betting system that incorporated all the rules of poker (specifically Texas Hold'em) into the game logic, smoothly able to progress between players. We were able to display the correct information for each specific LCD that we outlined in our design and properly reflect all changes in our displays as they occurred. We were able to successfully save game state to memory if the devices were turned off and ensure the information in memory was unique for each device. We were able to confirm that the devices would last the duration of at least 2.5 hours, which is what an average game lasts in an at-home setting [11]. Overall, we can confidently say a user of this device would be able to fully play a game of poker without issue.

5.2 Uncertainties

While many of our functional requirements were met and we were able to play a round of poker smoothly, we did run into a few errors and uncertainties that we were not able to address in the time that we had. Potential solutions and alternatives will be discussed in the Future Works section.

One of the biggest uncertainties that we had was proper communication. The transceiver module we used in this project is very sensitive; even the slightest disturbance would render any message being sent as un-receivable. This made debugging this issue fairly hard, since the transmitter believed it was correctly sending the message, but the receiver had no message to parse. Although the communication worked properly about 85% of the time, whenever a message was dropped, our game would halt in the middle of the game state. In this scenario, we would have to start the game completely from scratch.

Another uncertainty we ran into was the characters being displayed on the LCD. This was a very minute uncertainty, as we figured out that the garbled display was due to the accidental contact of wires on our PCB. Once we separated the wires more, we were able to avoid this issue. It remains as an uncertainty because without encasing, a player can accidentally push wires together again.

Lastly, an uncertainty that stems off of the previous issue is error handling. Say wires are pushed together and something was wrong in transmission, or a button was accidentally pressed, our system has no way of going back a round to fix the issue. In near-perfect situations, we are able to play the game as expected, but if any of the previous two issues occur, then we have no way of correcting it.

5.3 Ethical considerations

With any electrically heavy product, safety concerns are quite apparent. In our product specifically, the biggest safety concerns we have are a potential power overload and potential moisture/water short circuits.

Since we have multiple AA batteries (up to 4) powering our devices, it is possible that we could have voltage overload, which will eventually lead to a power overload, causing a potential explosion. We are attempting to regulate the voltage outputted by these batteries using both using a UBEC [1], which is a universal Battery Elimination Circuit, and the resistive capabilities available in the PCBs.

We are also concerned with the potential for moisture/water to get within the devices, whether intentionally or unintentionally. If water gets into the device(s) it will inevitably lead to a short-circuit as the components will be damaged. Since the product is ideally to be used indoors, or in an area where water cannot get into the device, a normal casing should be enough.

The goal of our product is to bring a fun, and fair system to the common or even professional poker player, by eliminating chips from the game and allowing for a fair system of money representation. By doing so we eliminate potential cheating from the game which satisfies IEEE Code of Ethics, #2: “to avoid real or perceived conflicts of interest whenever possible, and to disclose them to affected parties when they do exist” [2]. Thus, this product serves to not only allow for more players to participate at once, but also allows for a cheat-free system by getting rid of the need for physical chips.

While our product is meant to be harmless, as poker is simply a card game, it may aggravate certain illnesses such as a gambling addiction. This is in violation of IEEE Code of Ethics, #9: “to avoid injuring others, their property, reputation, or employment by false or malicious action” [2]. With the potential ease of this product, there is a chance of further aggravation to a person’s already harmful addiction. We do not have a means to solve such a health issue - we are assuming that players are playing responsibly and are careful with their money.

5.4 Future work

5.4.1 Physical Design

The physical design needs vast improvement to allow for this product to be mass-produced. The first and foremost thing that needs to be addressed is the lack of casing. Without casing, the device and all its electrical components are exposed. Firstly, this is a very unappealing look for a finished product, but more importantly this can cause functional issues as well. Without casing there is a strong chance of wire positions being manipulated by the surface that the device is placed on, and in turn causing them to touch each other since they are in such close proximity. This leads to disruptions in functionality and display as two different signals touching

have the potential of being mixed into one random one. Casing also serves as a way to ensure that the components are safely away from potential hazards such as moisture as this could cause short circuits.

5.4.2 Communication Method

For this project the communication was handled through the use of RF. While RF was a good choice for communication for the most part, there were times where the transmission was dropped due to interference and other extremities. A different method of communication (such as Bluetooth or Wi-Fi) and/or a different frequency band (such as 5 GHz) might have allowed for a stronger connection between the devices and thus allows for the ability to complete a game without the worry of a potential transmission signal drop. Communication is a huge foundational piece of this project and getting the proper one is essential to its future success.

5.4.3 Error Handling

Functionally, all of the buttons were able to do exactly what they were expected to do on press, but there were times where an accidental tap of a button occurred (pressing Call instead of Raise etc.). This was mainly due to either human error, or due to the fragility of the device (due to lack of casing). We lacked a means to correct these errors through either software or an addition of another button. This needs to be addressed when putting a product like this into production, as errors such as an unintentional button press are quite likely to happen.

References

[1] N/A, N/A. “What Are ESC, UBEC and BEC.” Oscar Liang, 26 Nov. 2016, oscarliang.com/what-is-esc-uber-bec-quadcopter/.

[2] Publications, IEEE. “IEEE Code of Ethics.” IEEE - Advancing Technology for Humanity, www.ieee.org/about/corporate/governance/p7-8.html.

[3] ATMega, ATMega. "ATMega628." Sparkfun. N.p., 2016. Web. 21 Feb. 2019.

[4] Igor, Ardunio. "Arduino's ATMega328 Power Consumption." Gadget Makers' Blog. N.p., 14 Dec. 2013. Web. 22 Feb. 2019.

[5] Julius, Bob. "Powerdis." Low Pass Filters. N.p., 2015. Web. 22 Feb. 2019.
N/A, N/A. "What Are ESC, UBEC and BEC." Oscar Liang. N.p., 26 Nov. 2016. Web.

[6] Publications, IEEE. "IEEE Code of Ethics." IEEE - Advancing Technology for Humanity. N.p., n.d. Web.

[7] QU1500_US_UL1, QU1500_US_UL1. "QU1500_US_UL1." D2ei442zrkqy2u.cloudfront. N.p., 2015. Web. 21 Feb. 2019.

[8] Martino, Gianluca. “Memory.” Arduino - Introduction, 2012, www.arduino.cc/en/tutorial/memory. Web. 4 March. 2019.

[9] Dejan, Dejan. “Arduino Wireless Communication - NRF24L01 Tutorial.” HowToMechatronics, 8 Apr. 2019, howtomechatronics.com/tutorials/arduino/arduino-wireless-communication-nrf24l01-tutorial/.

[10] Rouse, Margaret. “What Is EEPROM (Electrically Erasable Programmable Read-Only Memory)? - Definition from WhatIs.com.” WhatIs.com, 2016, whatis.techtarget.com/definition/EEPROM-electrically-erasable-programmable-read-only-memory.

[11] NA, NA. “How To Play Texas Hold'em Poker.” *PokerNews*, 2007, www.pokernews.com/poker-rules/texas-holdem.htm.

Appendix A Requirement and Verification Table

Component	Requirement	Verification	Verification status (Y or N)
Power Supply	1. Batteries power all our LCDs and Microcontrollers 2. Batteries should last for ~2.5 hours	1. Test this by using a Multimeter (Voltmeter specifically) to see how much voltage is outputted from 4 number of batteries and see if they reach the minimum needed to power all the components (need ~3.3V and ~5V to power all parts). 2. Test this by keeping the player devices and central hub device on for at least 2.5 hours	Y
Voltage Regulator	1. Must ensure that the step down from 6 V (+/- 0.3V) to 3.3 V (+/- 0.3V) is successful through the use of a voltage regulator 2. Must ensure that the step down from 6 V (+/- 0.2V) to 5 V (+/- 0.2V) is successful through the use of a voltage regulator	1. Send a 6 V input voltage through the voltage regulator and measure the output to see if it is 3.3 V with a +2%/-2% margin of error as specified in data sheet, using a Voltmeter 2. Send a 6 V input voltage through the voltage regulator and measure the output to see if it is 5 V with +4%/-4% margin of error as specified in	Y

		the data sheet, using a Voltmeter	
Microcontroller	1. Should be able to store data 1024 bytes of data (see Memory Section for more details) 2. Should process user signals and update game states within a reasonable amount of time, ideally under two seconds	1. Send simple data and see if it is able to reproduce it when it called 2. Press a button on the player device/central hub and see if it produces the desired action (e.g. FOLD, CALL, RAISE)	Y
Power Button	1. Switch successfully turns on or off the console	1. Test the switch for a set number of trials (30) and considered success if all trials work	Y
Call Button	1. When CALL button is pressed a HIGH signal must be processed by the microcontroller, else it will be 0 2. The microcontroller on player device should receive a signal from the CALL button > 95% of the time and store it to be sent to the central hub by the RF transceiver	1. Using a Multimeter we will check if the current is >0 Amps when the button is pressed and 0 if it is not pressed 2. Press CALL button 100 times and see if a signal is received at least 96 times	Y
Raise Button	1. When RAISE button is pressed a HIGH signal must be processed by the microcontroller, else it will be 0	1. Using a Multimeter we will check if the current is >0 Amps when the button is pressed and 0 if it is not pressed	Y

	2. The microcontroller on player device should receive a signal from the RAISE button > 95% of the time and store it to be sent to the central hub by the RF transceiver	2. Press RAISE button 100 times and see if a signal is received at least 96 times	
Up Button	1. When UP button is pressed a HIGH signal must be processed by the microcontroller, else it will be 0 2. The microcontroller on player device should receive a signal from the UP button > 95% of the time and store it to be sent to the central hub by the RF transceiver	1. Using a Multimeter we will check if the current is >0 Amps when the button is pressed and 0 if it is not pressed 2. Press UP button 100 times and see if a signal is received at least 96 times	Y
Down Button	1. When DOWN button is pressed a HIGH signal must be processed by the microcontroller, else it will be 0 2. The microcontroller on player device should receive a signal from the DOWN button > 95% of the time and store it to be sent to the central hub by the RF transceiver	1. Using a Multimeter we will check if the current is >0 Amps when the button is pressed and 0 if it is not pressed 2. Press DOWN button 100 times and see if a signal is received at least 96 times	Y

Fold Button	1. When FOLD button is pressed a HIGH signal must be processed by the microcontroller, else it will be 0 2. The microcontroller on player device should receive a signal from the FOLD button > 95% of the time and store it to be sent to the central hub by the RF transceiver	1. Using a Multimeter we will check if the current is >0 Amps when the button is pressed and 0 if it is not pressed 2. Press FOLD button 100 times and see if a signal is received at least 96 times	Y
Central Hub Display	1. Should be able to display characters with enough brightness, so user does not strain to read the displayed information; must be visible to read ~5 ft radius	1. Try various potentiometer settings until reaching a brightness that is visible to read from ~5 ft	Y
Player Device Display	1. Should be able to display characters with enough brightness, so user does not strain to read the displayed information; must be visible to read ~2 ft radius	1. Try various potentiometer settings until reaching a brightness that is visible to read from ~2 ft	Y
RFID Tag	1. The tag should be properly attached to each player device and be accessible and read by the RFID reader at least 95% of the time.	1. Test the RFID tag and reader by attaching it to an LED and having it light up if the RFID is successfully read; test this with all the RFIDs we will have and	Y

		ensure that at least 95 times of 100, for each id, is successfully read	
RF Transceiver	1. The transfer of data from the Player Device transceiver and Central Hub transceiver (and vice versa), should consistently be correct and fast 2. The transfer of data should occur properly within a radius of at least 5 feet	1. Send simple data between the Central Hub and Player Device, both hard-coded and dynamically, to be displayed on the LCD display. Do this 100 times to ensure the communication channels work to our specified speed (~250 Kbps) and with almost 100% accuracy 2. Go to various points around the device (<= 5 feet) and see if the transfer of data is still successful between player device and central hub	Y
RFID Reader	1. Ensure the Player Devices can be read into the game through the use of the RFID reader 2. Ensure the RFID tag is read on touch (less than 1 inch distance) 3. Ensure there is at least 99% accuracy of the tag being read	1. Test each individual RFID tag on the reader and ensure the reader is able to identify each separately and open a method of communication between the Player Device and Central Hub. If successful, all RFID tags will be saved on the Central Hub's microcontroller 2. Move the RFID tag closer and further	Y

		from the RFID reader and see at what distance the reader recognizes the RFID tag 3. Tap the RFID tag on the reader 100 times and look for 99+ plus successes	

Table 2: Requirements & Verifications

Appendix B Schematics

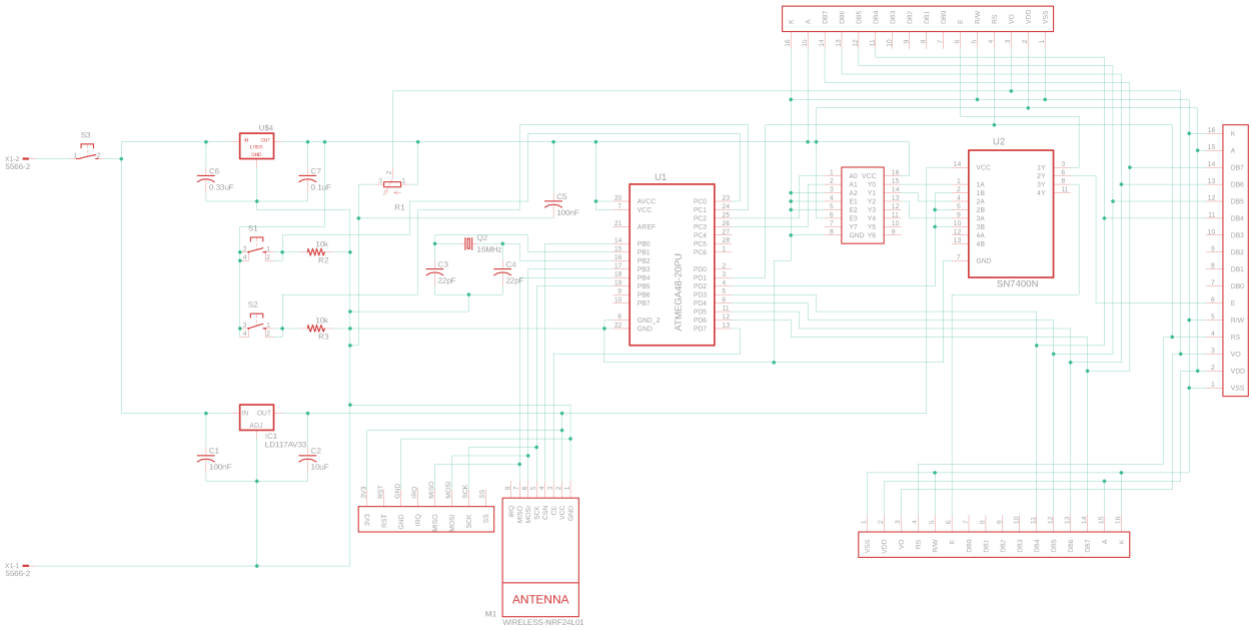

Figure 6: Central Hub Schematic

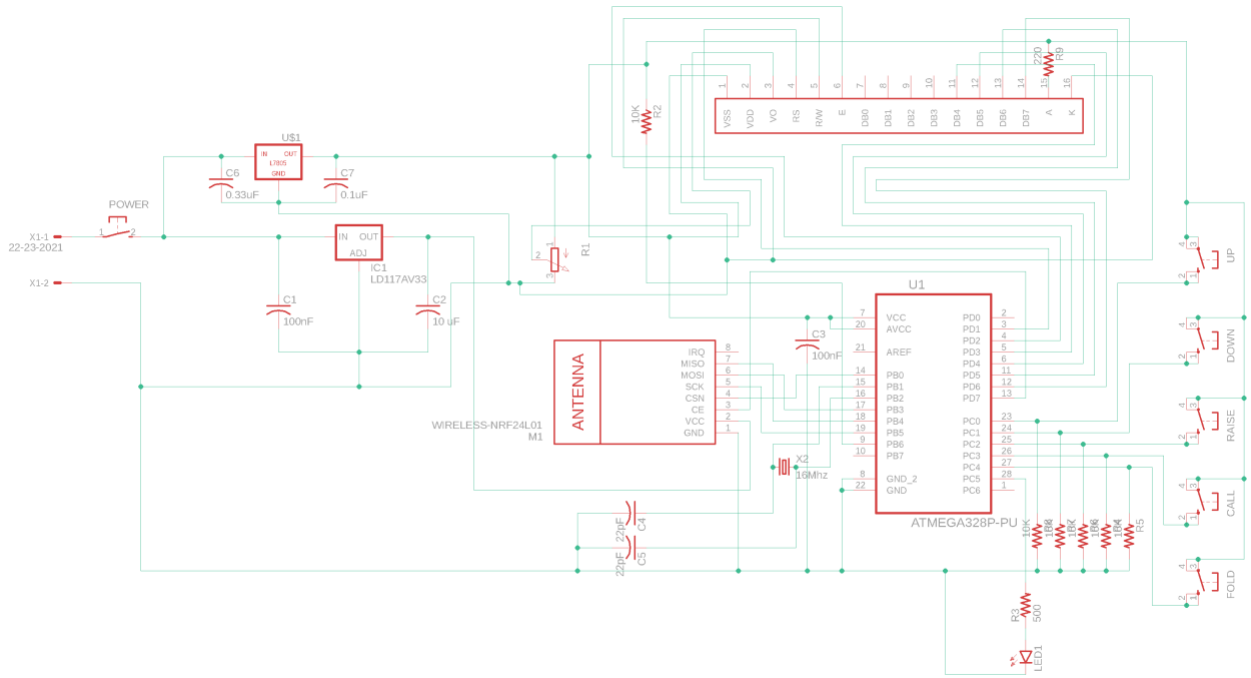

Figure 7: Player Device Schematic

Appendix C Real-World Images

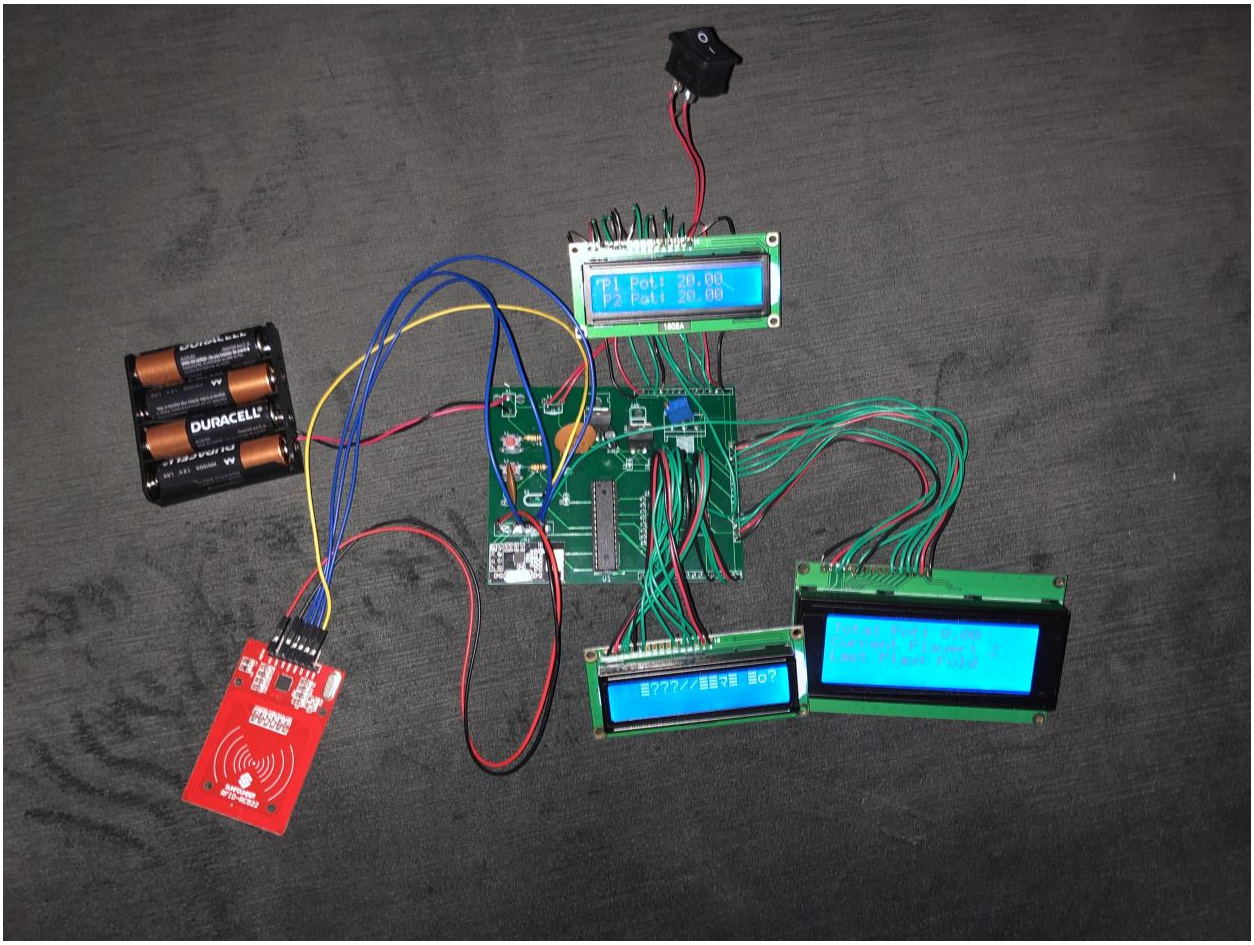

Figure 8: Central Hub

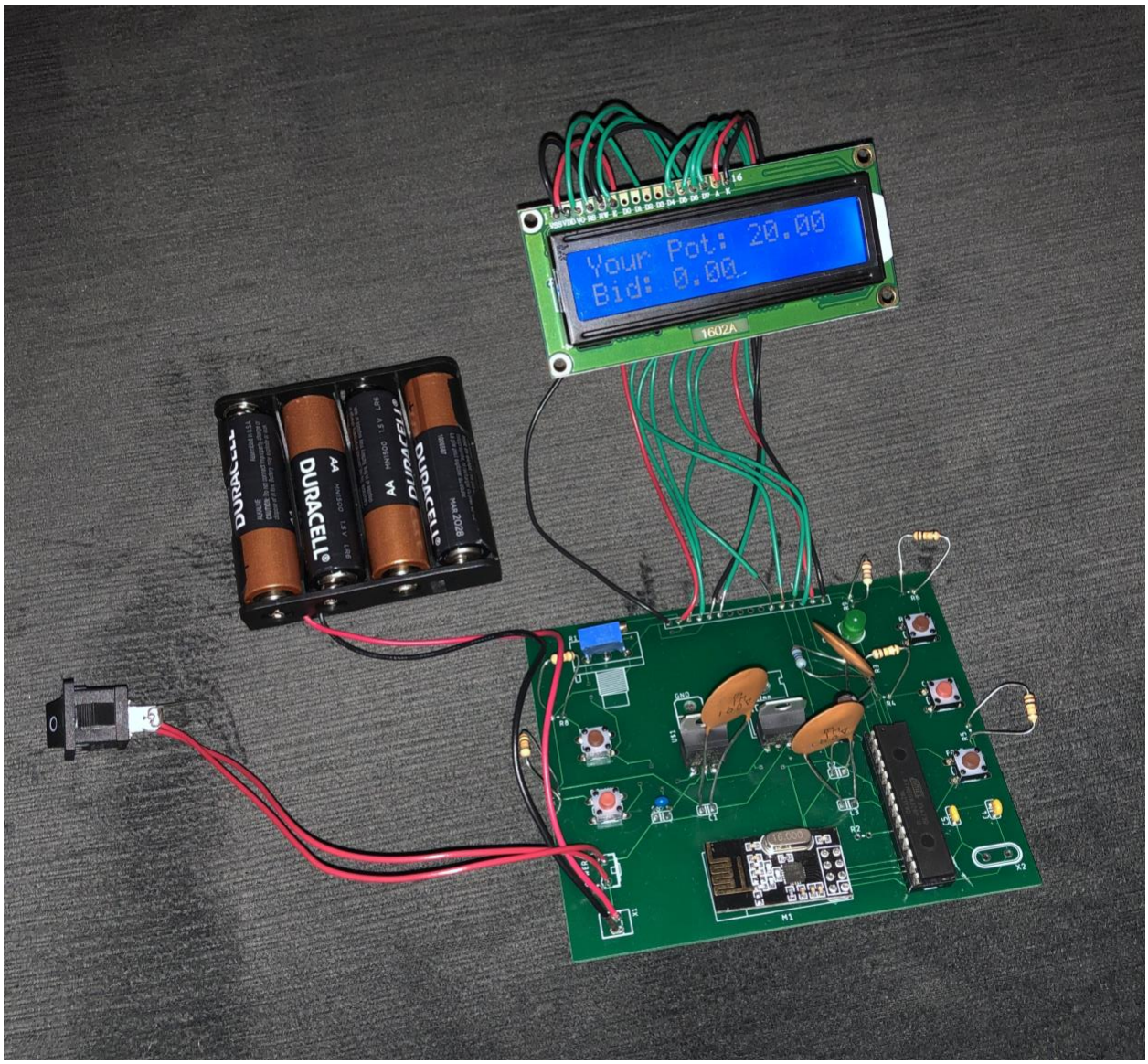

Figure 9: Player Device

Appendix D Code Snippets

```
if (call_press) {
  float player_pot_mem;
  char ret_msg[10];

  EEPROM.get(player_pot_store, player_pot_mem);
  EEPROM.put(player_pot_store, player_pot_mem - call_val);
  dtostrf(call_val, 5, 2, bid_amount);
  initial_amount = player_pot + player_bid;
  initial_amount = initial_amount - call_val;
  player_pot = initial_amount;
  player_bid = 0.00;
  EEPROM.put(min_bid_store, player_bid);
```

Figure 10: Example of EEPROM usage in code

```
void sendGameState(int player) {

  char buf[10];
  char call_bid[10];
  char raise_bid[10];
  float call_amount;
  float raise_amount;

  if (player == 1) {

    radio.stopListening();

    if(max_pot_g == p1_running_g) {
      call_amount = 0.00;
      raise_amount = 0.00;
    }
    else {
      call_amount = max_pot_g - p1_running_g;
      raise_amount = (2 * max_pot_g) - p1_running_g;
    }

    dtostrf((float)p1_g, 3, 0, buf);
    dtostrf(call_amount, 5, 2, call_bid);
    dtostrf(raise_amount, 5, 2, raise_bid);
    strcat(buf, call_bid);
    strcat(buf, raise_bid);
    radio.write(&buf, sizeof(buf));
    radio.startListening();
    p1_sent_g = 1;
    EEPROM.put(p1_sent, p1_sent_g);
  }
```

Figure 11: Sending player decision from player device to central hub

```
void updateScreen() {

  lcd_top.setCursor(0,0);
  lcd_top.print("P1 Pot: "); lcd_top.print(p1_pot_g); // probs take 2 out?
  lcd_top.setCursor(0,1);
  lcd_top.print("P2 Pot: "); lcd_top.print(p2_pot_g);
  lcd_bot.setCursor(0,0);
  lcd_bot.print("P3 Pot: "); lcd_bot.print(p3_pot_g);
  lcd_side.setCursor(0,0);
  lcd_side.print("Total Pot: "); lcd_side.print(comm_g);
  lcd_side.setCursor(0,1);
  lcd_side.print("Current Player: "); lcd_side.print(curr_player_g);
  lcd_side.setCursor(0,2);
  lcd_side.print("Last Play: ");

  switch(last_play_g)
  {
    case 1:
      lcd_side.print("None");
      break;
    case 2:
      lcd_side.print("Raise");
      break;
    case 3:
      lcd_side.print("Call");
      break;
    case 4:
      lcd_side.print("Fold");
      break;
  }
}
```

Figure 12: Snippet for updating screens on central hub