UCF SENIOR DESIGN 2

ELECTRONIC CHESS TRAINER BOARD

Final Report

Group 7

Eric Roberts	Electrical Engineer
Brandon Dupoux	Electrical Engineer
Jean Melgarejo	Electrical Engineer
Saeed Rahaman	Electrical Engineer

Table of Contents

List of Figures

1.0 Executive Summary

Chess players know that there is no replacement for the tactile feeling and weight of moving a chess piece and analyzing a physical sixty-four square chess board. With the invention and advancement of the smartphone, the chess board has moved to the digital screen and has brought artificial intelligence to the game. Smartphones allow you to play chess against an AI and even players from all around the world.

Many people find the game of chess to be extremely complicated and daunting to learn. A digital chess board also allows beginning chess players to pick up the rules of chess intuitively. By lifting chess pieces off the board, possible moves for that chess piece are highlighted on the board with LEDs and the player gets a visual understanding of how each piece moves.

While chess smartphone applications offer numerous advantages for casual players, advanced chess players and chess enthusiasts lose the physical aspect of playing chess. Thus, the motivation for this project was to find a way to intuitively teach how chess is played on a physical board and to create an engaging experience. The advantages of chess played on a smartphone will be applied to a physical chess board. Users will able to see how pieces can move based on available spaces for a piece. Also, a game mode selector will be included so that a player can choose to play against an AI or play against another player.

The Smart Chess Board is a chess board designed with features to teach new players the fundamentals of chess and how the chess pieces move. When pieces are lifted, LEDs will light up to visualize where the chess pieces are allowed to move. After each turn made by the user, the chess board will be checked to see that a legal move was made. When an illegal move is made, the LEDs will blink red repeatedly until the illegal move is corrected. Different LED colors will also be used to differentiate between both players and the computer. Each type of piece will also have internal resistors to differentiate which piece the player wants to move.

The piece identification feature of this system is what sets it apart from others. Most systems like this need to start at an original starting point to keep track of which piece is where but our system will use piece identification allowing any starting position or scenario to be initially set. This feature makes this system much more complicated from its counterparts while bringing a very simple feature.

Another very important feature of the design is the physical appearance of the final product. While this feature isn't innovative or groundbreaking, it's still very important to consider to be able to sell customers a product that's attractive. The hardware and engineering can support several features but if the product doesn't have an appeal to it; then the product won't score big at the store. The engineering specifications can only go so far since the customer only sees the outside appearance of the chessboard.

Using this product, users will gain a thorough understanding of how the game of chess is played and how each piece can move. The interaction of the chess pieces and various components will allow users to gain a visual representation of the fundamentals of chess. This overall goal is in conjunction with the goals or remaining low cost and easily manufacturable. We would like to see this system easily affordable for schools and chess clubs to bring a new method of teaching the game to students.

2.0 Project Description

The purpose for this chapter is to provide some background about the purpose, goals, and rules that will be necessary to carry out the construction of the project. The first section will talk about the purpose and quick overview of the scope of the project. The next section will go over some technical requirements that will necessary to implement the design correctly. Section three goes over some of the engineering/marketing pros and cons of upgrading certain areas of the design compared to the overall scope of things. The next section will go over some of the basic rules associated with the game of chess as well as some history and strategies. Finally, the last section shall go over the software and hardware tasks required to complete the project and who is responsible for each task.

2.1 Purpose

The goal for this project is to design a low cost and lightweight smart chess board that can help players learn how to play chess. The smart chess board will also be compact and portable so that the board can be carried around while maintaining a minimalist aesthetic design. The main learning objective for this project is to understand and implement an embedded system with various sensors and to write software to incorporate all the hardware together.

A variety of features will be included in the design of the board. The smart chess board will run on a rechargeable battery and have an efficient battery life to last multiple games. A mode selector will be implemented so that the user can choose to play against an AI or another human player. An embedded microprocessor will be used to implement a simple chess engine. The chess engine will be able to light up possible moves' situations. A LED array will be included underneath the board to indicate a variety of information. To prevent illegal moves, the LED where the illegal move was made will blink several times. Different LED colors will be used to differentiate between two players. The AI will also have a unique LED color to indicate that the user is playing against the computer. The LEDs will also be used to indicate where the AI wants to play so that the user can move the chess piece.

2.2 Requirement Specifications

- Chessboard shall be made from a material that's sufficiently aesthetic to create a slick design. Chessboard shall maintain a lightweight design less than 10 pounds.
- Chessboard pieces shall have a unique electronic signature that can be used to differentiate between units.

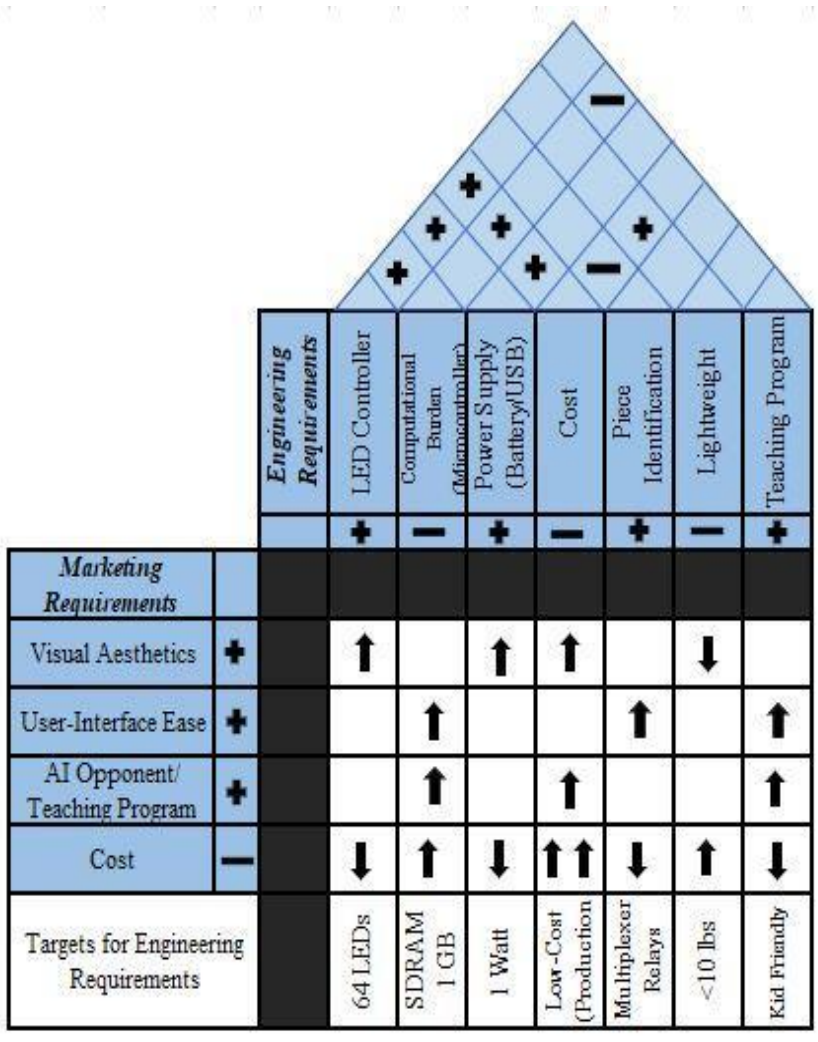
- Analog-to-digital converter shall be utilized to differentiate between different voltages given an input current on the pieces.
- RGB LED colors shall be implemented in each of the 64 quadrants of the chessboard.
- LED Controller shall be utilized to increase the simplicity of the design.
- Shift Register shall be utilized to select the row of LEDs that needs to be functioning in a timely manner to give a whole lit board appearance.
- Chess Piece Identifier shall be able distinguish between different type of units using multiplexers, relays, and microcontroller.
- Power Supply Battery shall be rechargeable and be able to maintain at least 5 full games on a full charge.
- Voltage Regulators shall be utilized to maintain consistent voltages throughout the design regardless of battery voltage.
- Chess game board shall be able to switch between two different modes (Emulator/Training).
- Micro Linux Computer shall be programmed in python and C if only one microcontroller is used.
- Microcontroller shall be programmed in C and Micro Linux Computer shall be programmed in python and C for chess emulator if that route is taken.
- Microcontroller shall have two user button inputs to switch between different game modes.
- Microcontroller shall include at least 25 GPIO.
- Chessboard shall include an Analog-to-Digital Converter integrated circuit with at least 10-bit resolution.
- Chessboard shall include PCB board for connections with the microcontroller, multiplexer, demultiplexer, ADC-IC, shift register, and LED controller.
- Microcontroller shall predict available moves whenever a piece is lifted by sensing that there is no longer an input.
- Micro Linux Computer shall maintain at least 1GB of RAM along with a microSD card.
- Chessboard shall be low cost in a production environment.

2.3 House of Quality

Using the House of Quality design tool, it was possible to create a relationship between the pros and cons of marketing targets and engineering specifications. To meet these targets, a set of needs must be met to incite the customer to purchase the product. Most valuable need for most products includes the end cost that the individual will have to pay for the product. Reducing this as much as possible is optimal. The second marketing target is the visual aesthetic of the chessboard. Improving the visual design could include adding features such as LEDs. The User-Interface will also be of significant interest since the customer will

be operating the chessboard. Creating a simple and user-friendly interface will be the cornerstone for creating a product that people want to buy. All successful businesses utilize simplicity in their design. The final marketing target is the performance of the AI opponent. Designing a neural network that adjusts playing style depending on the actions of the user and provides a level of difficulty for experienced chess players or an aspect of teaching for new chess players is the bulk part of the design that will be patented.

Figure 1 House of Quality



From an engineering perspective, the technical needs differ greatly from the marketing needs. There's more attention required for technical specifications to handle problems such as optimizing computational burden and increasing user simplicity. The overall cost of the product will dictate how sophisticated the technology is. The main objective here would be to have an efficient program that

can run a slick user-interface and handle the AI with sufficient time while being cost friendly. Since the chess board is designed to help teach the game to users, the final product will come with a power supply that can be recharged at the leisure of one's own house. The power supply will be the main limiting factor for the microcontroller, LED array, and Piece Identifier. Once the appropriate power requirement is calculated, the next step is to identify a microcontroller that can handle the computational burden associated with AI technology. Handling tradeoffs between how sophisticated the Artificial Intelligence is and the computational burden on the microcontroller will be very important. The final engineering requirement is creating a user-friendly interface that anyone can operate. Features would also include "smart options" that teach the user how to handle certain situations and the best play available considering all the positions of the units.

2.4 Chess Gameplay

Chess is a very popular game played throughout the world by millions and is known for its strategic gameplay.

2.4.1 Origins

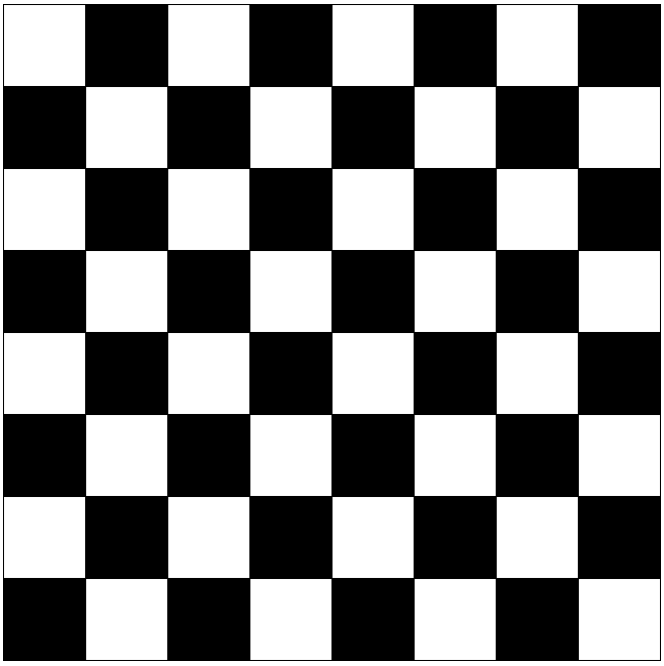
The origins of the earliest form of chess date back all the way to the Gupta Empire of India in 600 CE. [1] It is widely debated who exactly created this game, however, it is agreed upon that it was around this time era it came into existence. Fast forward to more present times during the 18th century when the game came full circle, its migration over to western Europe allowed for the game to gain steam. It was played at coffee houses in premiere cities like London and Paris. It's recognition was seen with the first ever official chess tournament that took place in 1851 held in London, England. [1] With the advent of this game in Europe, popularity exploded across the region. Official leagues began popping up and because transportation was becoming widely available and inexpensive the game traveled across all regions on the globe. Because of the strategies and complex nature of chess for a long time in history chess was reserved for the upper class which was revered mostly as more intellectual as the lower class. In present times we know this not to be true and almost every school in the US has a chess club and all people worldwide enjoy this game.

2.4.2 General Rules

A standardized game of chess contains an eight row by eight column gridded chessboard that has a total of 64 squares. Of those 64 squares, their colors alternate meaning that half of them contain a lighter color such as white or beige, while the other half are usually darker colors such as brown or black. It is a two-player game, where one opponent is up against another opponent. When the two

players face off against each other, the general convention is to have the board aligned in such a way that on each player's sides in the right corner of the board, a beige square shall be to the outermost part of it.

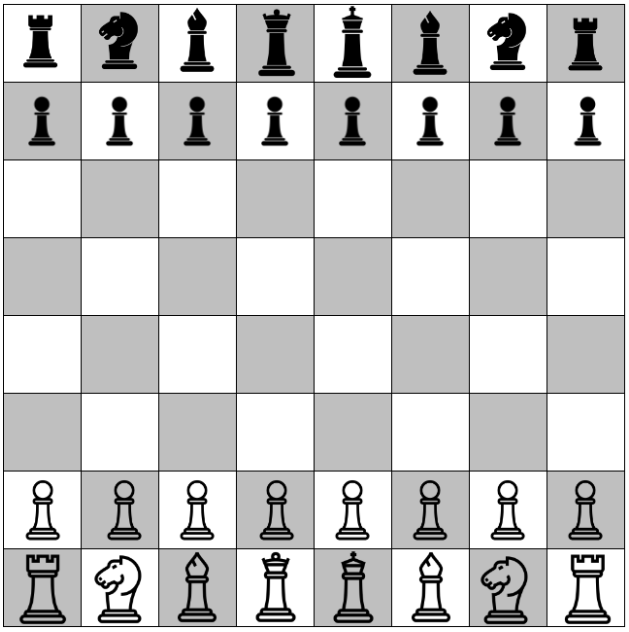
Figure 2 Standard Chess Board



Another convention used is the use of light and dark chess pieces, which also usually end up being divided into white and black pieces. Each player gets 16 pieces each, with the total number of pieces on the chessboard at the beginning of a game totaling to 32 pieces altogether. Each set of 16 pieces will consist of eight pawns, two knights, two bishops, two rooks, one queen, and one king. The significance of each piece is shown through its movements and consequently, its importance.

The main setup used for a game of chess is important to make sure all the pieces are in the right places. To begin with, the front row of each of the sides will contain the eight pawns all across. The back row will have the two rooks at the ends of each corner. Next, the two knights will have to be placed next to the two rooks. The two bishops come after that and are placed next to the two knights that were previously placed. Finally, moving on into the middle, the king and queen are placed as the last remaining pieces on the board. It is important to note that the king goes on the middle left tile and the queen goes on the middle right tile. Lastly, by convention, the white set always goes first.

Figure 3 Standard Chess Board Setup



The game is very strategic to say the least, and this holds true to its main objective. The main goal of the game is to get a checkmate on the opponent's king. Further discussion on how this occurs, and other end game scenarios will be discussed below in section 2.5.5 End of Game.

2.4.3 Chess Piece Movements

Before this game can be enjoyed, it is important to understand the rules and movements of all the pieces on the board. Each piece has a unique move or set of movements, whether they be used to overtake an opponent's piece or to keep a piece in check. The subsections below go into greater detail on each part's unique movements, special moves allowed, and the possible end game scenarios.

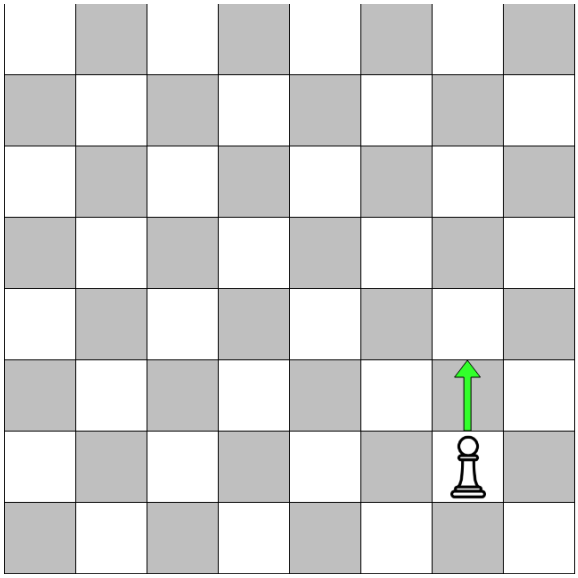
2.4.3.1 The Pawn

The pawn is the most common piece on the chessboard that has a total of eight pieces out of the 16 pieces on a player's side. This piece is generally renowned as the weakest and most expendable on the board, however, if used correctly it can prove to be an important part of different strategies in the game. Strategic moves can be anywhere from offense to defense and the pawns can be utilized to play significant roles to control the board.

The pawn is allowed to move one tile forward in a straight direction as long as the tile is not occupied by any other piece on the board. It is only allowed to jump two

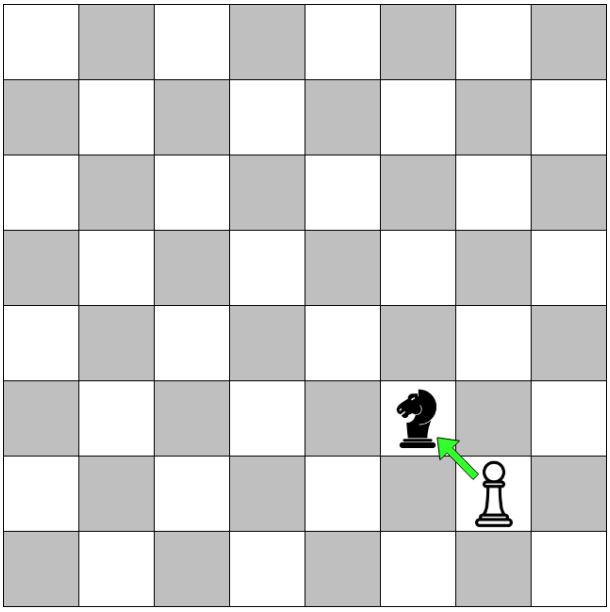
tiles ahead only when it is the pawn's very first move at the beginning of the game. Also, a pawn is not allowed to move backwards at all.

Figure 4 Pawn Movements



The pawn also has a special movement allowed only when attacking and this is known as a diagonal attack on an opponent's piece which would be on a diagonal tile in front of the pawn to the left or to the right side. The pawn then replaces the piece it has captured by moving into its tile which was previously diagonal from it.

Figure 5 Pawn Attacks



2.4.3.2 The Knight

The knight is one of the more complicated moving pieces on the board, moving in an L-shaped manner amongst the board. It is, however, a very special and unique piece because it is the only piece on the board that can jump over other pieces. The knight will always end up on a square that is opposite of the color it was just on after its move is complete. One thing to be noted is that just because the knight jumps over pieces, does not mean that the pieces it has jumped over get captured. It still would only capture the piece it would land on after its move. It goes without saying that the knight will not be able to land on a square that is already occupied by a piece that is its own color.

Figure 6 Knight Movements (Red Squares Only)

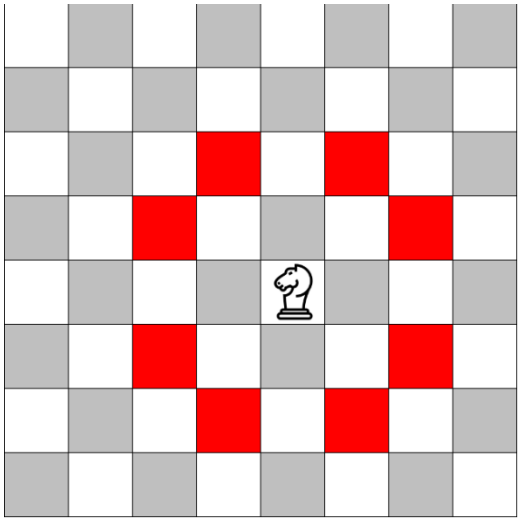
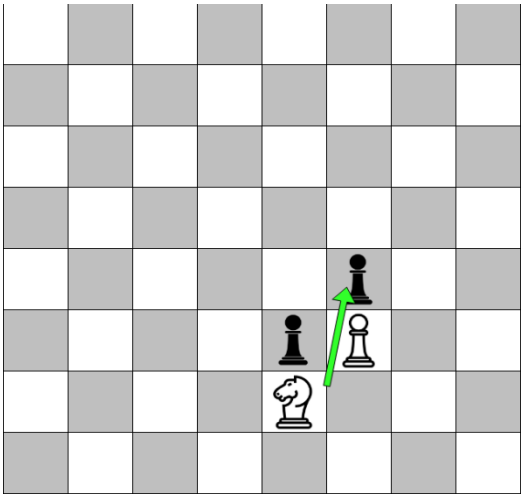


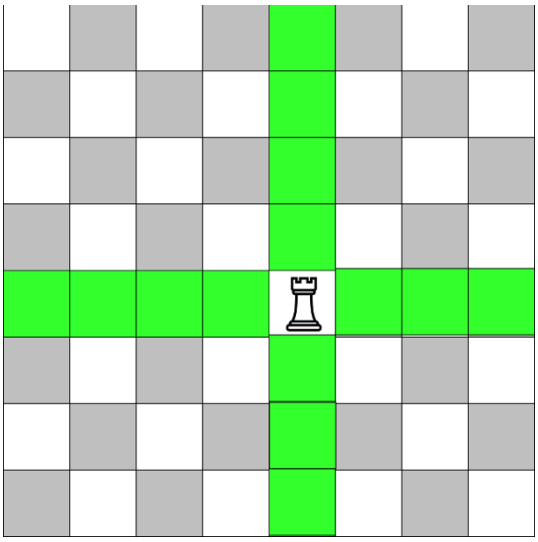
Figure 7 Knight Attacks



2.4.3.3 The Rook

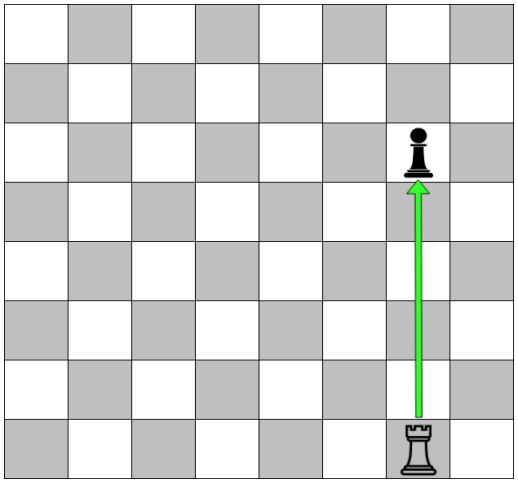
One of the stronger pieces on the board, the rook plays a great role in both offensive and defensive strategies due to its piece movements. The piece can move horizontally or vertically on the chess board. Its limits are if another piece which it cannot overtake, is within its path, or if it reaches the end of the board in its movement. Another advantage to using the rook is that it is the only piece on the whole chess board that can land on any square, regardless of the color.

Figure 8 Rook Movements



This does not mean it can leap over other pieces, as the rook is only allowed to move any number of tiles if it captures an opponent's piece or there are no pieces blocking its way.

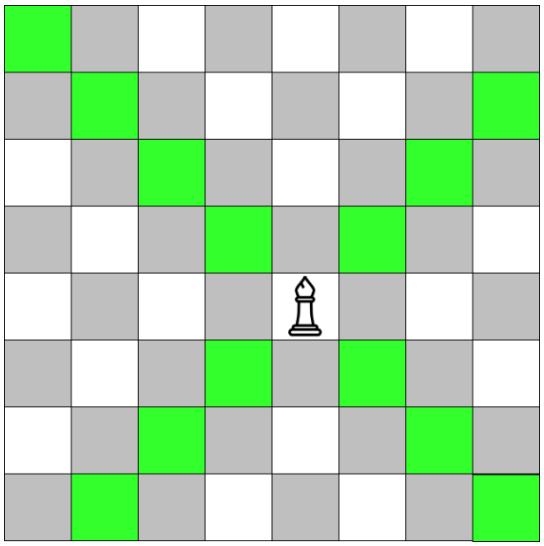
Figure 9 Rook Attacks



2.4.3.4 The Bishop

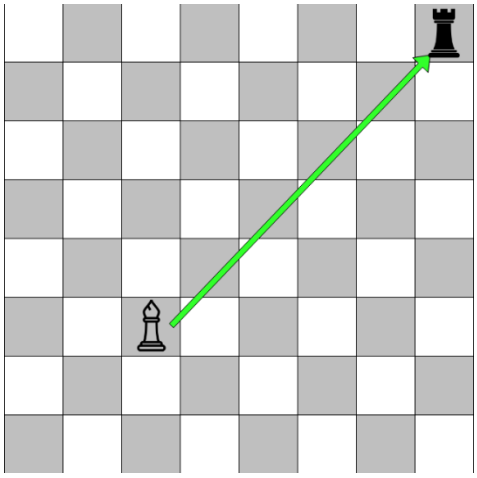
The bishop is defined through its diagonal movements. They have no restrictions in terms of distance and whether they are moved forwards or backwards.

Figure 10 Bishop Movements



However, just like the rook, if another piece is within its path, it will not be able to leap over that piece. It is restricted to only overtaking an opponent's piece in that case or just to be blocked if it is not able to take whatever piece is in its way. Since two bishops are what each player begins with, it means that one of the bishops will be restricted to the light-colored squares while the other bishop will be confined to the dark colored squares.

Figure 11 Bishop Attacks



2.4.3.5 The Queen

The queen is widely regarded as the strongest piece on the chess board. Not only can it control the most squares on the board in comparison with any other piece, but it can move forwards, backwards, and diagonally. There is no limit to how many tiles that it can navigate, however, it is possible for the queen to be blocked from movements. The queen also does not have the ability to leap over any pieces. In most cases, the queen is not risked too often just to overtake a piece due to the significance it holds in leverage to win.

Figure 12 Queen Movements

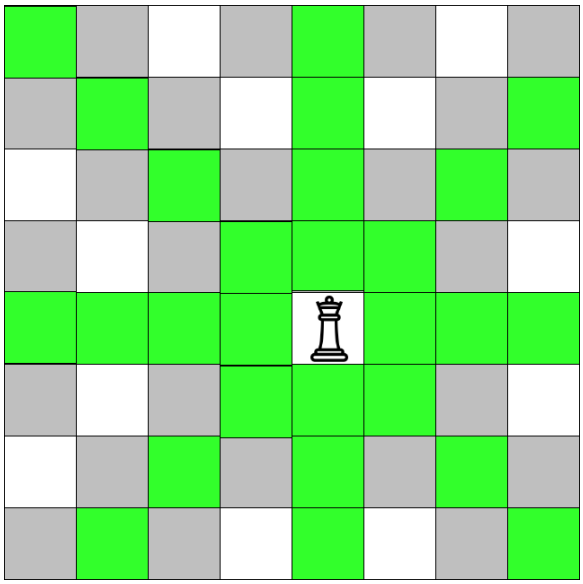
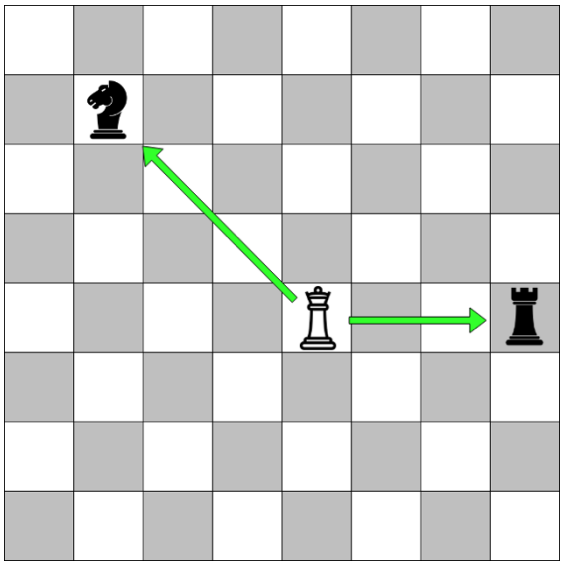


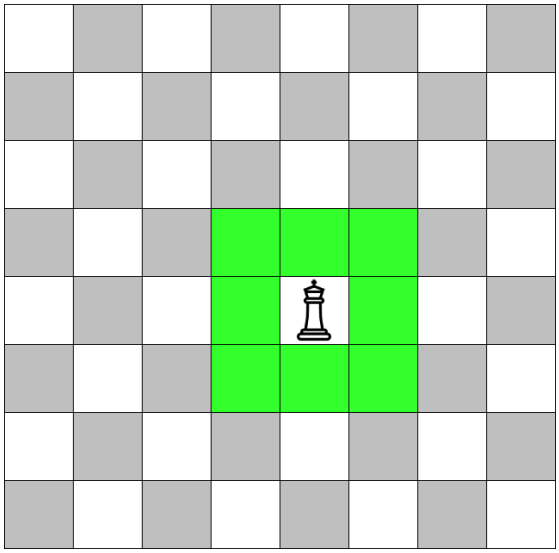
Figure 13 Queen Attacks



2.4.3.6 The King

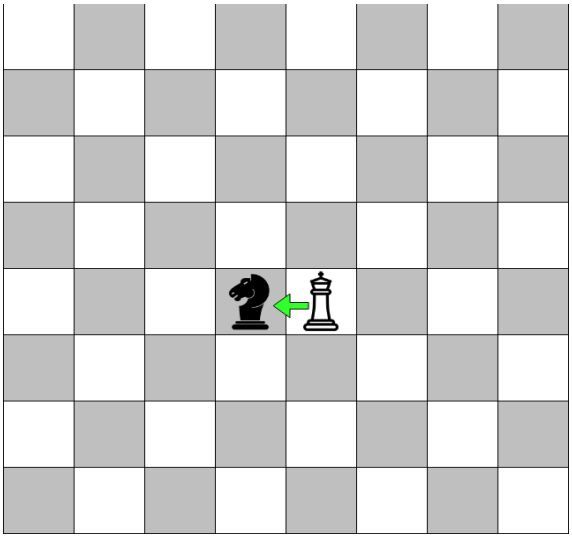
The most important piece in the game is the king. This piece is what must be safe at all times and can never be captured. If the king is in any danger of being overtaken, it must be moved immediately so that it is in the clear before any other piece can be moved. This is known as the king being in “check”. If there is no possible way for the king to be safe, then it is considered a checkmate and the game is over. This is further discussed in section 2.4.5 End Game. The king’s movements are confined to any direction (horizontally, vertically, or diagonally) as long as it is one square it moves.

Figure 14 King Movements



Just as its movements show, it also captures the other pieces in the same way, by being one square away in any direction. The only other restriction on the king is the fact that the king is not allowed to move to any square that would put the king in check. This ruling can also forcefully cause the king to have to capture or move from its spot if it is in check from another opposing piece next to it. Utilizing this forced move from check players have created and devised complex strategies to force the king into a position where it cannot win. These strategies are the one of the most studied and used in current practice.

Figure 15 King Attacks



2.4.4 Special Moves/Scenarios

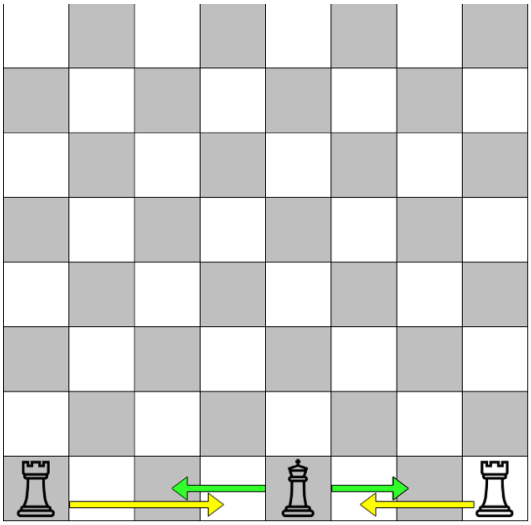
Like many games, there is always certain exceptions to the rules and that holds true to the game of chess. The special moves allowed are listed below.

2.4.4.1 Castling

The term “castling” is a maneuver that engages a player’s rook and king through a strategic act. The scenario may come about in a usual game where the king and rook have not moved yet, and no pieces are in between them [4]. This can constitute the possibility of castling which allows the king to move two squares to the left or to the right and then the rook is moved after to be on the opposite side of the king. It is important to note that the king must be moved first rather than the rook being moved because that could be the difference of the move being recognized as castling or the opponent claiming the move for only the rook. The move is valid once all these certain conditions are cleared:

- The king has not been moved previously
- The rook has not been moved previously
- The king cannot be in check
- There are no pieces in between the king and the rook
- The king does not encounter a square under attack with this move

Figure 16 Castling Movements



2.4.4.2 En Passant

The name itself gives away the move's capability which in French means "In passing". This move occurs when an opponent's pawn is moved two squares from its starting position. Just like the special move "castling", this move too has to meet certain requirements in order for it to be considered to be valid:

- The pawn being captured must be on an adjacent square and be the very first movement forward moving two squares.
- The attacking pawn must be on the same row as the pawn being captured.

Figure 17 En Passant Movements

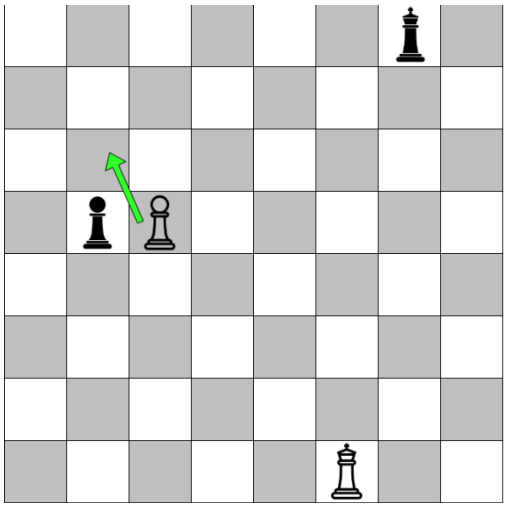
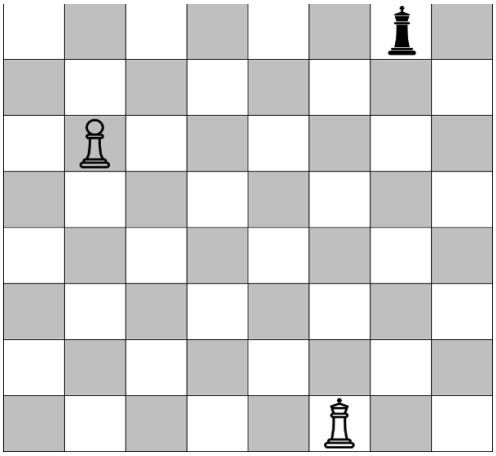


Figure 18 En Passant Result



2.4.4.2 Pawn Promotion

The final known special scenario in chess is known as a pawn promotion. This occurs basically when a pawn makes its way to its eighth rank. The player then has a choice of replacing the pawn by either a rook, knight, bishop, or queen that is of the same color. The new piece of choice will replace that pawn on the same square.

2.4.5 End of Game

In order to achieve victory, a player must obtain a checkmate on the opponent's king. This means that the opponent's king will be trapped and unable to make a single move without being in check. When that scenario takes place, then a victory is obtained. Listed below are the various scenarios that will setup or allow for an end of game.

2.4.5.1 Check

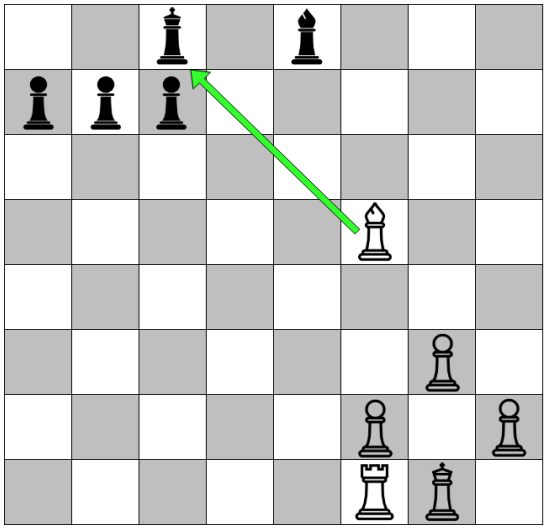
When a player is in "check", this means that the player's king is in current trouble of being captured by the other player's piece. Due to this urgent scenario, there are a few possibilities that can take place to get out of this:

1. The player moves the king to another square that will not keep it under attack.
2. The player blocks the attack by using another piece to put in between the king and the attacking piece.

3. The player captures the attacking piece by overtaking it with either the king or another piece.

The graphic below illustrates an example of a scenario when the black team's player would be in check. The white team's bishop can clearly have a direct path to capturing the black team's king.

Figure 19 End of Game: Check

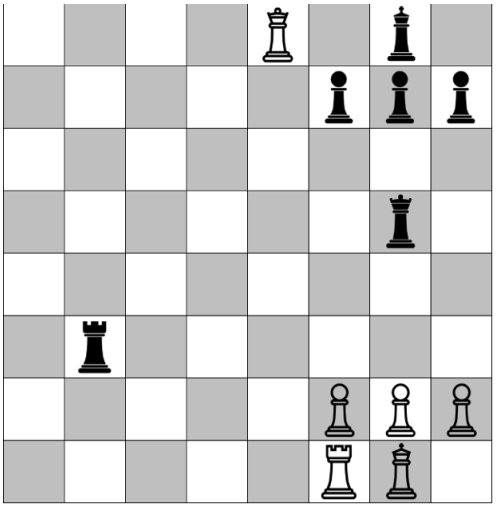


2.4.5.2 Checkmate

When a player is in check, the three scenarios presented in the 2.4.5.1 Check section show the possible ways to save the king from being captured. If there is no possible way to stop the king from being captured, then it is known as a checkmate. Checkmates can be presented in many different possibilities depending on what pieces are left on the board.

The example below is provided to show a basic scenario of one of the plethora of ways it can be achieved. Here it is seen that the white team's queen has got the black team's king forcefully trapped with no way out for the king to go since it is being enclosed by its own pawns in front of it. Even if the black team's king were to move horizontally, it would still be stuck in check because the queen can also move as many squares horizontally as it wishes.

Figure 20 End of Game: Checkmate

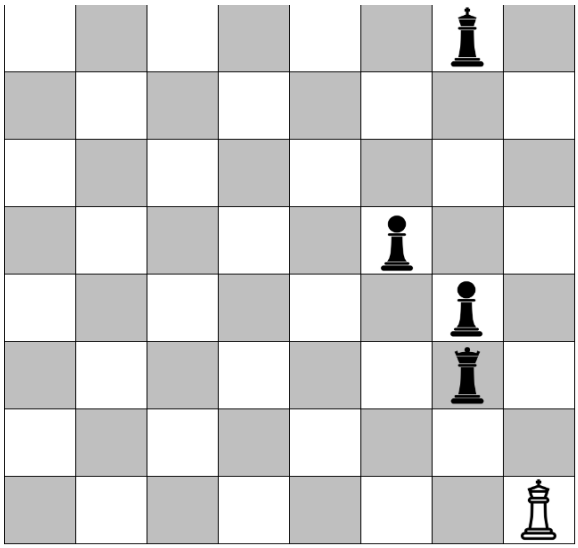


2.4.5.3 Draw

In chess, there are 6 ways for a game to end in a draw:

1. Stalemate: Probably the most common of the six. This result is basically when the king is not in check but is unable to move to any other square without being in check.

Figure 21 End of Game: Stalemate



2. Perpetual check: This occurs when an opponent repeatedly checks the other player's king and there are no possible moves left for the king without ending up in check [4].
3. Insufficient mating material: This occurs when neither of the sides possess enough pieces that will allow for either side to be able to checkmate either of the kings [4].

For reference, some combinations impossible to checkmate with are:

- King and two Knights against a King
- King and Bishop against a King
- King and Knight against a King
- Just the two Kings on the board.

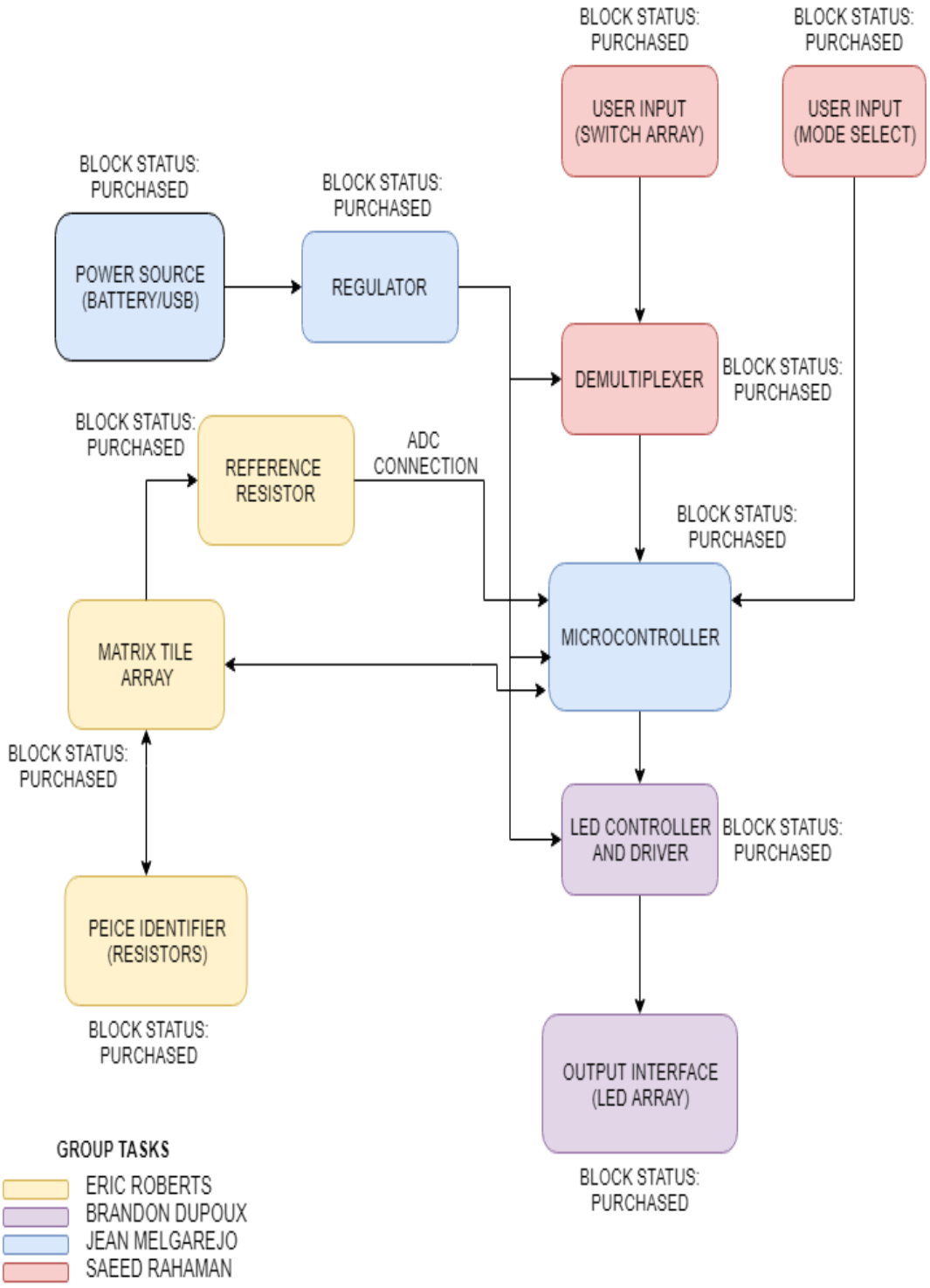
1. Repetition of moves: If three times in a game, a particular position takes place, then a draw may be claimed by the player.
2. 50 move rule: Within 50 consecutive moves, if a capture has not been made or a pawn move has not occurred, then a draw may be claimed by the player.
3. Draw by agreement: This occurs when both players decide that both sides positions are equal and come to an agreement to call the game a draw.

2.5 Task Division

To ensure that deadlines are met in the most efficient manner possible, the project scope was split up into software and hardware tasks. The color coding of the block diagrams displays who was responsible for what during the construction of the design. The first area of concern is the hardware required to implement everything the chessboard needs to do. This includes power source, voltage regulators, multiplexers, shift register, microcontroller, micro-Linux computer, LED driver, LED array, analog-to-digital converter, etc. The second part is the implementation of the software for all these integrated circuits to communicate properly within one another to complete the appropriate task. This includes user input, training mode, emulation mode, SPI communication, etc. Weekly meetings were also scheduled for the sake of communicating the whole project scope with one another and seeing if the software/hardware tasks were coming together.

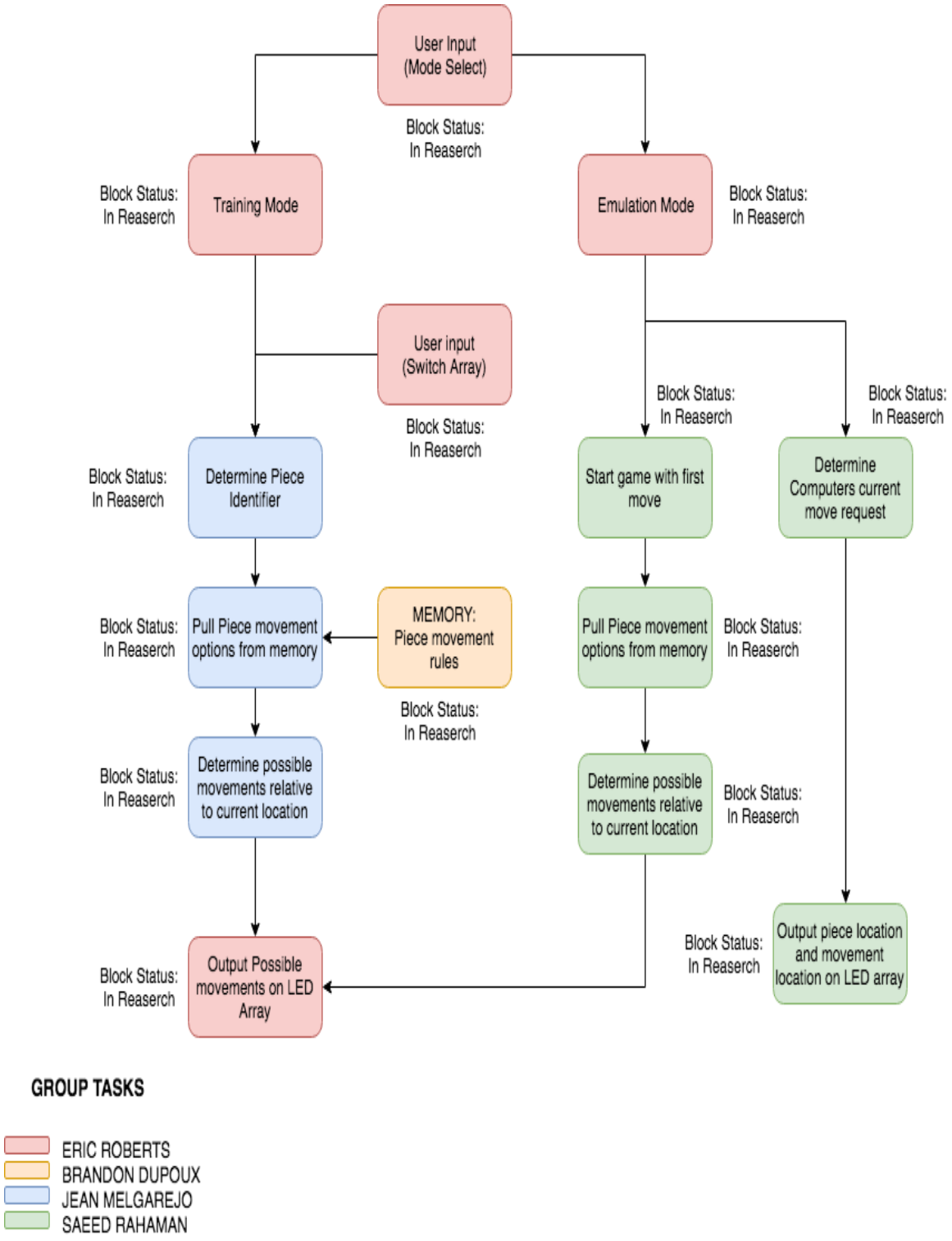
2.5.1 Hardware Block Diagram

Figure 22 Hardware Block Diagram



2.5.2 Software Block Diagram

Figure 23 Software Block Diagram



2.5.3 Software/Hardware Timeline

Table 1 Hardware Timeline

Timeline	Software	Hardware
May (21st-31st)	Research and development	Research and development
June(1st-15th)	Raspbian operating system configuration on micro SD card	Order 90 percent of the parts required
June(18th-29th)	Arduino IDE code for SPI protocol to communicate with the shift register and command it to light up a certain LED using the serial command prompt	Test circuit for Shift Register, LED array, and Arduino
July(2nd-13th)	Adafruit open source code for TLC5947	Test circuit for LED Driver and Arduino
July(16th-30th)	Adafruit open source code for ADS1115 for I2C communication protocol and final schematic design	Test circuit for Analog-to-digital converter with Arduino and photoresistor

3.0 Project Research and Investigation

The initial step in efficiently designing and constructing any project is the research phase. The Electronic Chess Trainer Board will feature unique aspects to it that will require thorough research before constructing. Basic materials such as a board and 3-D printed pieces will be needed along with various hardware requirements such as a microcontroller, microprocessor, relay array, led controller/driver, multiplexers, and PCB board. On the software side of things, features include an automated chess trainer to teach beginner players, chess emulator for playing against artificial intelligence, and piece identifier for when players want to emulate specific scenarios.

The first section shall go over pre-existing projects that were done in the past. This allows a solid foundation to be built on for designing a new and improved version of past projects. The next section shall go over relevant technologies that will be incorporated into the design. Properly researching the technologies will ensure a firm understanding is solidified throughout the group. Knowing how these technologies work helps continue onto the next section. The next section will encompass the selections for the parts that will be used on the physical design. This step is important, choosing parts that aren't electrically suitable or incompatible for the project will incur a huge loss on the timetable. The final section will include conceptual models and research potential approaches for completing certain tasks. A firm understanding of these different technological implementations will give us more versatility on how we can approach implementing different designs.

3.1 Existing Projects and Products

A system that integrates the modern technology of today with simplicity and learning opens the door to innovation. As with much of today's technology, many new projects and ideas stem from improvements based off existing technology already built. This holds true to the design of modern chess games, which also use similar technology such as LED's, programming, and other newer technologies to bring the game to life in a modern approach rather than traditional. Many of the newer versions of chess can be seen through the likes of apps on mobile devices such as iPhones or Androids, but essentially seem to be on hand-held devices rather than adding a physical touch to it. The given projects below are examples of existing ideas that implement, not only today's technology but also a traditional aspect regarding the physical pieces on the board as opposed to just a virtual side.

3.1.1 Square Off

Square Off is Kickstarter funded project whose main feature is a two-axis robotic arm underneath the board which is used to move the pieces automatically. This game offers the option of playing against people from an app over WIFI which means they can be anywhere in the world if they have the chessboard. The game is connected to the physical chessboard using Bluetooth technology. Another game mode that this platform offers is the AI mode where a player can go up against the computer in 20 different difficulty levels.

3.1.2 Smart Chess Board

A smart chess board was also designed by a senior design team at the Milwaukee School of Engineering in 2014. Their chess board design allows a player to play against an advanced AI. Their board uses LED's to signal where the computer wants to move as well as indicating where a piece can be moved when it is picked up. This is implemented using infrared sensors as stated by the group. The game is controlled through a touch-screen tablet that starts the game and selects whatever mode that is chosen by the player. Any sounds generated by this chess project were part of the chess engine that was used.

3.1.3 DGT Smart Board

A company called "Digital Smart Technology" develops a range of electronic chess boards. These chess boards have a variety of features and have a high build quality. The board is lightweight and thin, yet the board is made with high quality plastic and wood. For most of the features of the DGT Smart Board to function, the chess board must be connected via USB to a computer. Thus, the DGT Smart Board is not very versatile and portable. When the DGT Smart Board is connected to a computer, the chess board can be used as an input device on chess websites. Whether you play with another person online or a computer, the DGT Smart Board connects with the opponents moves. The Smart Board can also be used to record games to review and analyze later. You can also connect the Smart Board to a chess engine on a computer and play through them. Another interesting feature is the ability to broadcast a chess match using a free cloud service offered by the company. Other designs by DGT include wireless communication such as Wi-Fi and Bluetooth to reduce the physical USB connection needed for the Smart Board to work. However, the DGT Smart Board still relies on a desktop level computer being necessary for the advanced features of the Smart Board to function. Thus, the DGT Smart Board cannot be used for pickup games or be used for traveling. The DGT Smart Board also retails for around $500 and must be shipped from Europe [14].

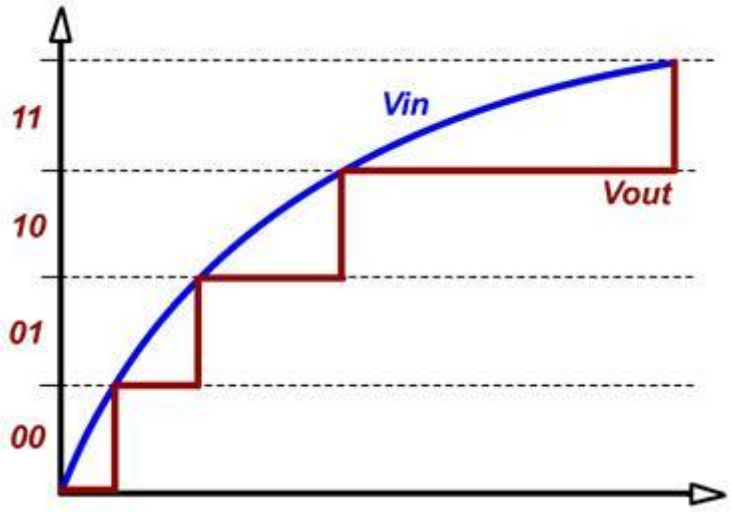
3.2 Relevant Technologies

The purpose of this section is to describe the technology relevant to the project that the group researched to get a better understanding on how the design of this interactive chess board would work. This section defines and describes hardware as well as components and features that the group needs a better understanding of to properly implement the technology into the design.

3.2.1 Analog to Digital Converter

The Analog to Digital Converter (ADC) is an integrated circuit (IC) that takes an analog signal and converts it to a digital signal. This IC can either be integrated into a microcontroller or be a standalone device. The analog signals that we are trying to convert are typically low voltages from a sensor. The ADC converts these signals into a binary number that ranges from a 0 to a maximum number that is defined by the resolution of the ADC. For example, we may have an ADC with 12-bit resolution. This means that our analog signal will be converted to a 12-bit binary number. If the minimum value is 0 we can calculate the maximum which is 212-1=4095. This means our analog signal will be converted to a digital number between 0 and 4095. The bounds of our analog signal are determined by the power source. If our ADC is being powered by a 5V source, then the ADC will only be able to convert analog signals less than or equal to 5V.

Figure 24 Analog to Digital Conversion Example



Determining how accurately the ADC will perform we will need to know the resolution, let's say 12 bits, and we need to know the power source value, let's say 5V. The conversion from an analog signal to a digital signal is proportional with respect to the analog value and digital value. If the analog signal is 0V then the digital equivalent will be 0. If the analog signal is 5V then the digital equivalent will

be 4095. If the analog signal is half of the max 5V/2=2.5Vthen the digital equivalent will be 4095/2=2047.5 (rounded up equals 2048). Knowing the maximum resolution and the bounds of our input power we can determine accuracy. The equations are upper bound minus lower bound divided by maximum resolution. In this case (5-0)/4095 = 0.001221V = 1.22mV, so the difference between 1024 and 1025 is 1.22mV which is very accurate.

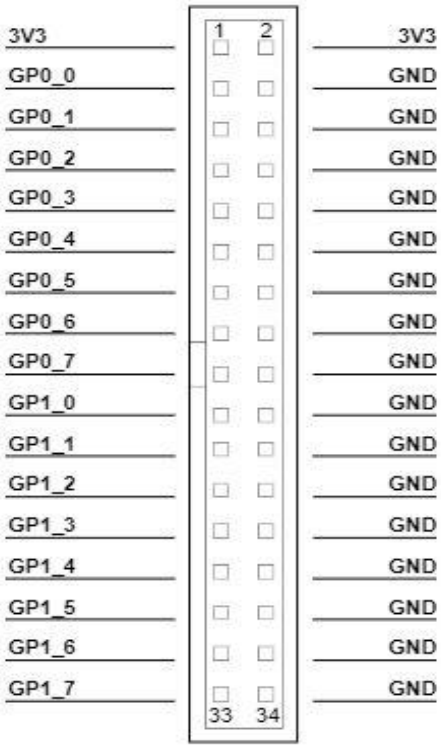
This technology will be the baseline for piece identification in this project. Each piece will have a unique resistance value and the ADC will detect the unique voltage based off that value and determine the playing piece from a predetermined table in the software.

3.2.2 General Purpose Input/Outputs

The General-Purpose Input/output pins are digital signal pins that can be programmed and connected to peripherals on an integrated circuit. GPIO can transfer either data or power and can act as both an input or output at runtime. For the project to run correctly, a microcontroller with at least 25 GPIO is required to handle all the peripherals and outputs associated with the chessboard. The microcontroller will then be feed, through an ADC input, a specific voltage that can be used to distinguish between different units on the chessboard. The microcontroller will also need to control the LED output on the quadrants given unique conditions. Obtaining a microcontroller with enough GPIO is virtually a necessity since the board will not operate correctly if it's missing an input/output. While Pulse Width Modulation would be nice to have on the microcontroller for more control with the LEDs it is not necessary.

If there is not enough GPIO, the PCB design can always include more shift registers which will increase the number of pins available. While including integrated circuits on the PCB board will help fix peripheral problems, more problems are added to the scope of the design. Adding GPIO in this manner effectively increases the complexity of the design and may cause future problems when constructing the final product. The easiest solution to this problem is the simplest; obtain a microcontroller or micro-Linux computer that has sufficiently enough GPIO to handle all the peripherals that need to be controlled by the processing unit of the design.

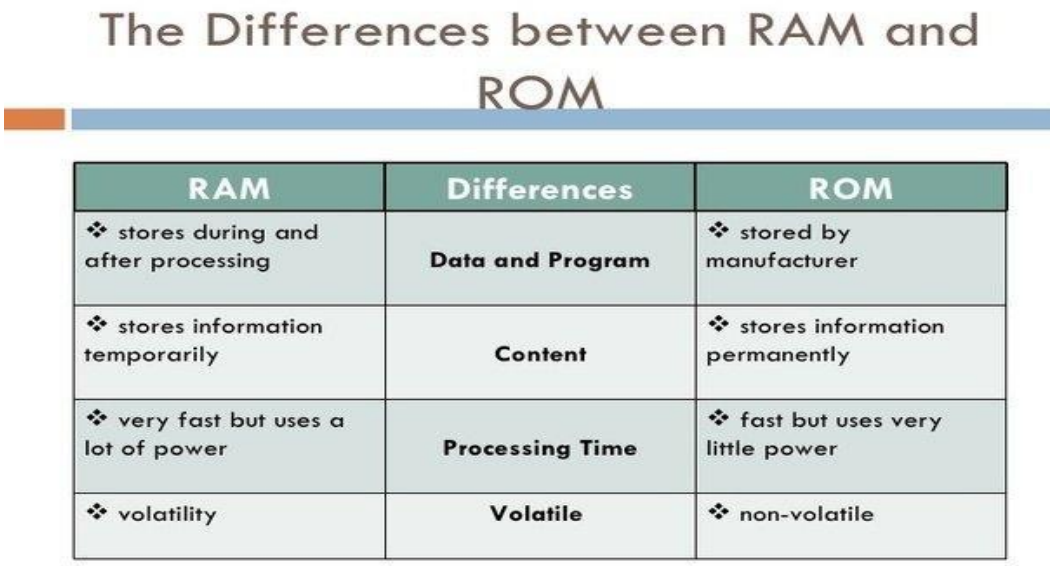
Figure 25 GPIO Connector



3.2.3 RAM/ROM

Random Access Memory (RAM) is the main memory that temporarily stores information close to the CPU for quick access. This increases the processing speed of the processor which allows for more information to be processed in a shorter amount of time although it also increases power consumption. Without RAM, computers would be very slow and would not run programs in a practical speed. That being said, there are disadvantages to them. The two main problems when you compare it to Read-Only Memory (ROM) is that it has more power consumption and it's more expensive. The benefits of ROM on the other hand, include permanent memory, cheaper per capacity, and lower power consumption. Using them together allows the computer to take the non-volatile memory of the ROM and brings it to the RAM to increase processing time. Optimizing these two memory devices allows a reasonable processing time as well as a consumer-friendly price. It's important to research a microprocessor that excels in this area to run the chess emulator part of the project.

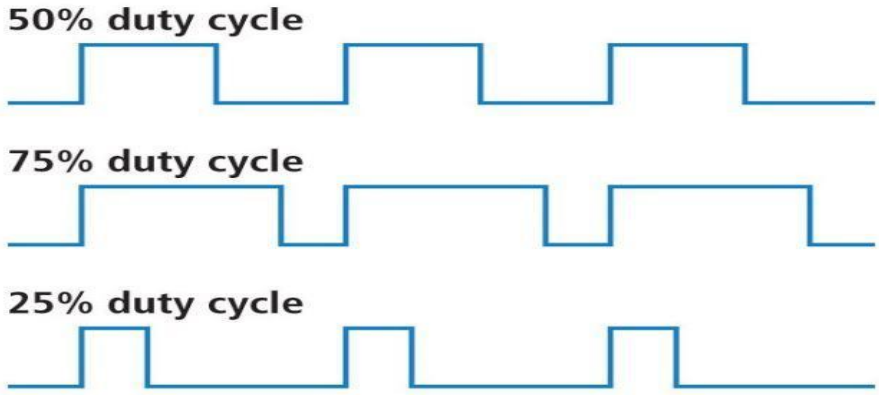
Figure 26 RAM vs ROM



3.2.4 Pulse Width Modulation

Electrically, creating an analog signal is impractical but by using high speed clock frequencies it can be done using a digital signal and a method known as Pulse Width Modulation. The most practical uses for PWM are led dimming and direction control in servo motors. It works on the principle of duty cycle controlling how much time a digital signal is "on" at 5 volts and "off" at 0 volts. If the signal has 50 percent duty cycle, then it represents a perfect square wave that's on half the time and off the other half. Increasing the duty cycle past 50 gives more on then off time and below 50 percent gives more off then on time. Utilization of the PWM will allow the RGB LEDs the freedom of emitting a chosen color based on duty cycle of the red, green, and blue LEDs.

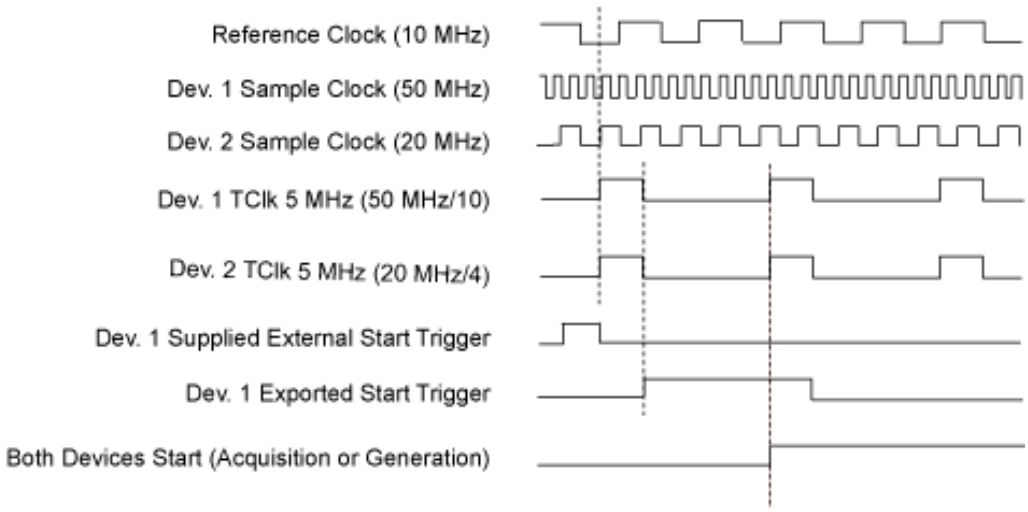
Figure 27 50%, 25%, and 75% Duty Cycle Examples



3.2.5 Clock Frequencies

Using a quartz-crystal circuit gives a computer an oscillator that sets the tempo for the processor. Clock speed is measured in pulse per seconds. For example, if a processor runs at 1.6MHz then that means that 1.6 million pulses are generated per second. On the computational level, higher clock speeds allow for more instructions per second. That being said, a higher clock speed doesn't necessarily boost performance. Performance speed is based on several things such as type of microprocessor, bus architecture, instruction nature, and random-access memory. For example, a 16-bit processor works slower than a 32-bit processor at the same clock speed. Choosing a microprocessor that has enough performance speed to run a chess emulator is important. Since total performance is based on several factors however, a high clock speed may not be necessary. On the other side of things, a high clock speed may be necessary for the LED driver and shift register. Since there are 8 individual rows of LEDs that need to be processed one by one, a high clock speed is necessary to create the illusion that all 8 rows are on at the same time.

Figure 28 Examples of different Clock Frequencies



3.2.6 Voltage Regulator

Voltage regulators work by limiting the output voltage to a specified amount regardless of the input voltage or load condition. There are two types of voltage regulators, switching and linear. High degree of voltage accuracy isn't necessary for the scope of this project aside from the ADC input. Therefore, linear voltage regulators will be enough and are the most cost-effective option. The chessboard will feature low power devices which makes the linear voltage regulator even more

appealing regardless of the power dissipation and efficiency. One variable that should be considered when implementing this device is the dropout voltage. The dropout voltage is the minimum amount of input voltage higher than the output required to correctly regulate the voltage to a fixed level. Since the chessboard will feature several devices that will be feed a unique voltage value it will be necessary to select a regulator that limits the voltage accordingly.

Figure 29 Linear Voltage Regulator Circuit

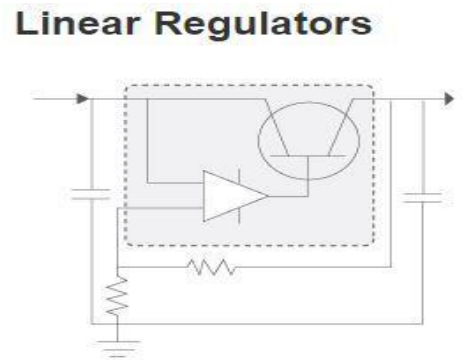
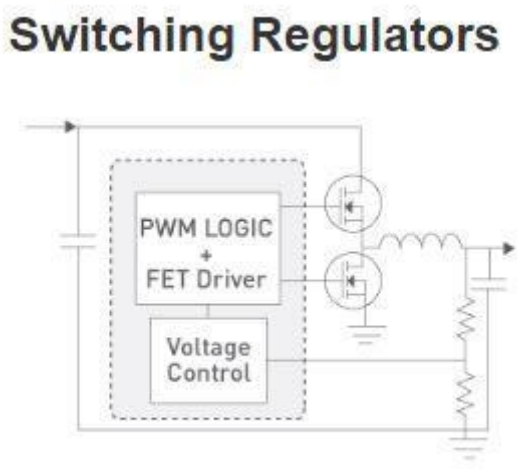


Figure 30 Switching Voltage Regulator Circuit



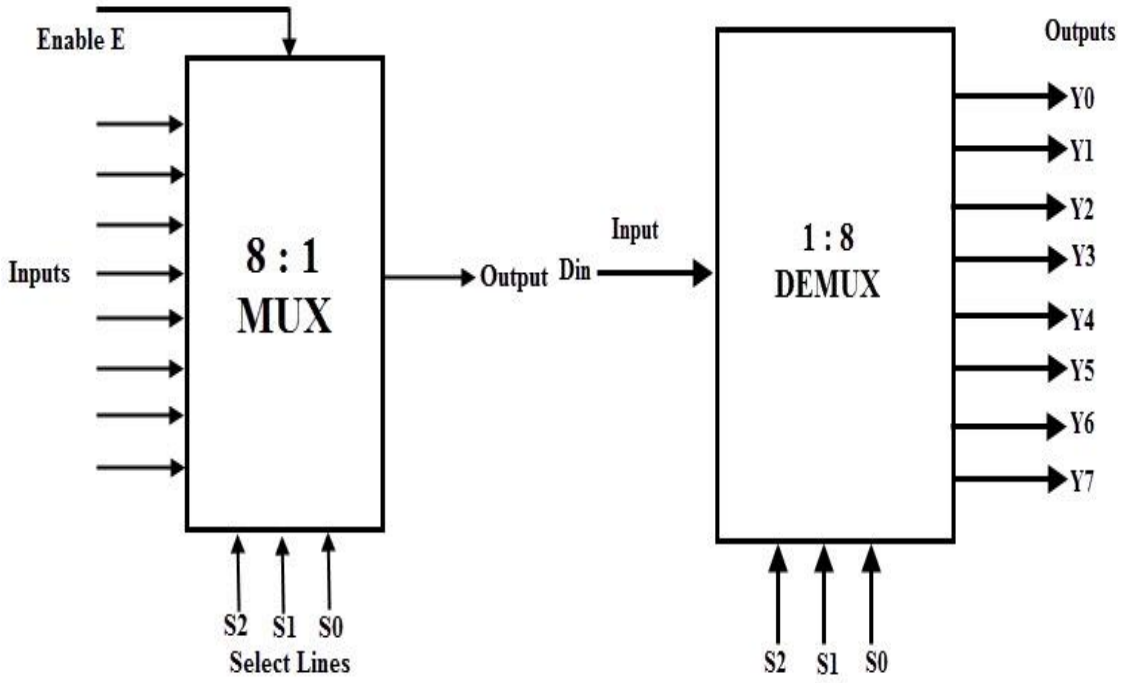
3.2.7 Multiplexer/Demultiplexer

Our design integrated Multiplexers (MUX), which are integrated circuit devices that act as a digital switch between multiple outputs. Multiplexers usually range in sizes between four and sixteen inputs and one output. The switching occurs when there is a change of input, which is in binary form. The select terminals correspond with the number of outputs. For example, three bits can have eight possible binary

equivalents. This means the input of the Multiplexer has eight terminals and there will need to be three select terminals to switch the device. One other feature that the Multiplexer has is the enable pin. The enable pin allows daisy chaining of multiple Multiplexers as well as disabling a Multiplexer if needed. Figure 3-9 below shows the basic block diagram for an eight input, one output Multiplexer.

A Demultiplexer (DEMUX) will also be used for this design. A Demultiplexer is an integrated circuit very similar in design to the Multiplexer. In fact, most Multiplexers today are combined with a demultiplexer and there is an extra terminal that toggles the function from multiplexer to demultiplexer. A demultiplexer is the opposite of a multiplexer. It has one input and 8 outputs. The outputs are selected via three select terminals. Figure YYY below shows the basic block diagram of a one input, eight output demultiplexer.

Figure 31 8:1 Multiplexer/Demultiplexer Block Diagram



3.2.8 LED Sensing Hardware

A major feature of the smart chess board is the ability to show users where a piece can move. To accomplish this task, the board must know when a user wants to move a chess piece and what chess piece the user wants to move. To accomplish the first task, several options were discussed. Pressure sensors and magnet switches were considered. Pressure sensors work based on the amount of pressure they can detect. Pressure sensors have a threshold they must reach. Once the threshold is reached, a current is sent to the microprocessor. Magnet

switches work using magnets to trigger the switch. Usually, the switch is open and when the magnetic field from a magnet is detected, the switch closes allowing current to flow through to the microprocessor. Figure 3-11 below illustrates the concept of a reed switch. To accomplish the task of determining which piece the user wants to move, the hardware will be discussed in a later section.

There are two proposing ideas on how to detect when a chess piece is selected. One method is to use pressure sensors underneath each square of the chess board. To activate the sensor, the piece must be pushed down on. When the sensor is activated, a current will be sent to the board indicating that the pieces is being activated. As the pressure increases, the resistance in the sensor decreases, causing the output current to decrease. The downside of pressure sensors is the cost of implementation. The cheapest sensors that measure sensors are around five to eight dollars. The cheaper sensors are also very inaccurate in detecting pressure. This could cause problems if pressure is accidently detected when a chess piece wasn't pushed down. More expensive sensors are more accurate but can cost up to twenty dollars which would exacerbate the total cost of the product. Another potential problem is being able to sense the pressure from underneath the chess board. The thickness and material of the board could interfere with sensor's ability to detect pressure. Coupled with the inaccuracy of the sensor, pressure sensors do not seem to be a viable method of determining when a piece is being selected by the user.

Another method to detect when chess pieces are selected is by using magnetic switches. Magnetic switches, such as reed switches are extremely cheap ranging from twenty-one to thirty-five cents for a single switch. The reed switches are also very compact with an average length slightly longer than a standard resistor. The reed switches work by using the magnetic field from a magnet. When a magnetic field interacts with the reed switch, the switch closes allowing current to pass through. This type of reed switch is known as "Form A" or "normally open" because the switches are in an open position when there is no magnetic field present. Reed switches can also work in the opposite direction. These reed switches are called "Form B" or "normally closed" because they are closed when there is no magnetic field present. From B reed switches are more expensive than Form A reed switches costing around twenty dollars for one switch.

Thus, our design will incorporate workarounds to accommodate Form A reed switches. Each chess piece will have a small magnet attached at the bottom. This will cause each switch to be closed and allow current to be sent through. Thus, when a piece is lifted, the magnetic force will disappear, and the switch will open causing the current to disappear. When the current disappears, the chess board will know that a piece is lifted and to activate the LEDs needed to highlight possible moves for the lifted piece. To account for the squares of the chess board where there are no pieces, the chess pieces will have different value resistors in them as well. These resistors will be used for identification of the pieces. A range of current

values will be used to identify the pieces and prevent the LEDs from triggering falsely.

Another design is to use two magnets for each switch to simulate a Form B reed switch. With a magnet underneath the chess board covering the reed switch and a magnet in the chess piece, a Form A reed switch will act like a Form B reed switch. When the two magnets are present, the magnetic forces of each magnet cancel each other out causing the reed switch to be in the open position. When a piece is lifted off the board, the reed switch will close allowing the current to travel through. With this setup, the LEDs will activate when a current is detected. Using resistors of piece identification, the magnitude of the incoming current will indicate which piece is being selected and the proper LEDs will light up.

Another problem is how to identify pieces in the player versus player mode. One part of the problem is how to know which piece the user wants to move if magnet switches are used. The ADC won't be able to detect the current because the chess piece is lifted off the board and not contacting the board. One solution is to configure the system to work in reverse. When the game begins, all the chess pieces are making contact with board. Thus, the magnet switches will all be closed, sending a current to the ADC. When a chess piece is lifted off the board, the current for that piece will disappear. The ADC will know what type of piece is lifted up based on what input current is missing. However, this creates another issue when a chess piece is captured. One solution is to have to an area on each side of the chess board for the captured pieces to go to. Magnetic switches will be placed underneath these areas and when a captured piece makes contact, the current will travel to the ADC and the ADC will still get input from the captured piece. The input from the captured piece is to prevent the ADC from not detecting more than one current and activating more LEDs than necessary.

Table 2 Pressure Sensors vs Magnetic Switches

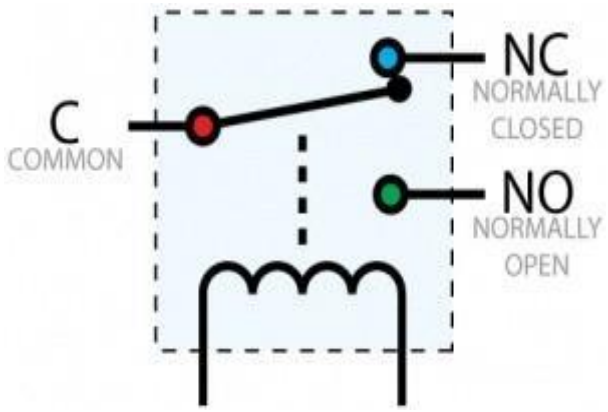
Features	Pressure Sensor	Magnetic Switches
Cost	$7 - $20	$0.20 - $10
Accuracy	Low Accuracy	High Accuracy
Maximum Current	1mA - 2.5mA	0.5mA - 1A

3.2.9 Relays

A relay is an electromagnetic switch. A relay is controlled by a small electrical signal that generally closes the switch to allow a much larger signal to pass though. To flip this switch a small number of current passes through a coil of wire near the

switch and creates a magnetic field. As the field becomes larger, it will be great enough to toggle the switch. The switch itself in general can hold a much larger signal than needed to toggle it. Some uses of a relay allow a microcontroller with very low output current to control large current driven devices like motors and lights. Other uses of relays are for switching testing applications. For example, a microcontroller can test many resistors with one ADC if every resistor first comes through its own relay. Then the microcontroller can switch on the relay when it wants to test that resistor. We can test many devices at once now automatically by controlling relays. Figure 3-12 below displays the internals of a relay.

Figure 32 Internals of a Relay



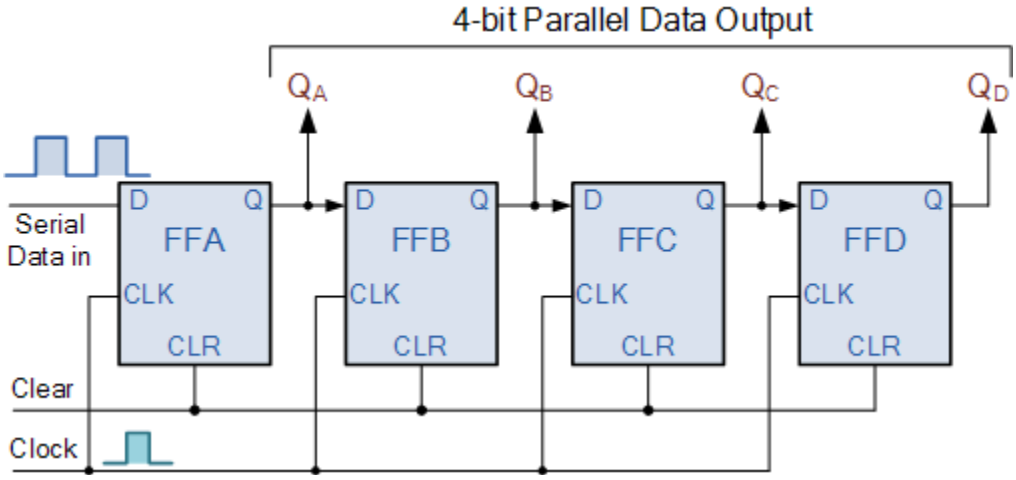
3.2.10 Shift Registers

A shift register is an integrated circuit (IC) that takes a binary data input and outputs that number along a digital pinout. The shift register outputs are usually connected to peripherals and the device itself can control multiple peripherals at one time. This device acts like a multiplexer, where you have an input and you can select one output based on a binary number corresponding to it, but instead the shift register can output multiple pins at a time in any combination. It can do this by only using about the same number of pins on a microcontroller that a multiplexer would use. Shift registers are very versatile components in the engineering world.

The way a shift register works is simple. There are three to four pins that control the device. A data pin, a clock pin, a latch pin, and sometimes there is an enable pin to. The enable pin allows you to switch off the shift register without cutting the power. The enable pin usually also clears the memory. The data pin provides the one or zero that we are trying to set on the device. The clock is the periodic signal where data is written to memory on the rising or falling edge. The latch signal takes the binary number created and stored in the memory and outputs it on the output pins. The components built in to a shift register are shown below in figure 32a. The block diagram describes a four-bit data output. This is a small-scale version; the standard size of a shift register that we researched is an eight-bit output system.

In the diagram below the latch system is nonexistent. The latch pin is something that isn't needed to properly function a shift register but it is an added benefit because you can program it while its outputting a different value. This is very helpful for our project because of the fast-paced changing of the shift register values.

Figure 33 Block Diagram of Shift Register



This device is best understood in an example. If I want to output a 10 on the device, I would clear the device first. Set data high when the rising edge of the clock passes then set data low for the next rising edge of the clock. Before the next rising edge after that I would set the latch high to output the memory. The memory starts from the lowest significant bit (LSB) and shifts the left. So, the initial 1 we set moved to the left on the next rising edge and the LSB was set to a zero. We can keep shifting these values to left until there are not more output pins, but nothing will output until we latch. Once we latch the value holds on the output until it is latched again with new values or the data is cleared.

Shift register applications are wide and many, but we will talk about relevant applications to our project. The most common application that shift registers are used for is to increase the number of GPIO pins off a microcontroller. By using three or four pins on the microcontroller we can control a shift register that can theoretically have up to an infinite number of output pins. The research done with shift registers on out project includes the LED array. The best way to select LEDs more than one at a time is a shift register. In fact, the LED driver we are looking at is just a shift register with other LED driving components around it.

3.3 Strategic Components and Part Selections

In this section, strategically selecting components that meet the demands of the project shall be researched. Once an analysis of each part is completed, the parts will be compared, and the final part selection will be based on electrical characteristics and cost.

3.3.1 Microcontroller

For the design of this project, it will be important to analyze the microcontroller specs and find a model that can suit the needs associated with the design scope. Even though the CPU is the heart of the microcontroller its importance isn't going to be a focal point for the selection. The constraints associated with the Electronic Chess Trainer Board will involve focus in specifications such as volatile memory, non-volatile memory, clock frequency, general purpose inputs/outputs (GPIO), and analog-to-digital converters (ADC). Specifically, the GPIO, RAM/ROM memory, and clock frequency is of importance.

There are two options available for the implementation of the Electronic Chess Trainer Board. One option includes using a single micro-Linux computer with a substantial amount of general-purpose inputs/outputs (GPIO) and enough computational power to run the desired programs. The GPIO are necessary for all the peripherals that the microcontroller will need to analysis and the computational power needs to be enough to handle the chess emulator as well as the trainer program. Since this approach literally incorporates a small computer into the design; there is much more flexibility in terms of total functionality, but this approach also increases the complexity of the design. Examples of increased complexity includes the addition of integrated circuits in the case that the micro-Linux computer doesn't have enough hardware functionality.

The second option would utilize a microcontroller and a micro-Linux computer that are specialized to perform in their intended functions. The microcontroller would control all the peripherals and have the necessary GPIO for the LED controller, multiplexers, demultiplexers, shift register, and piece identifier. This would reduce the need for a strong CPU on that microcontroller and simplify the system. The micro-Linux computer would focus solely on the computational burden associated with programming a chess emulator and teaching program. This second microprocessor would focus more on RAM/ROM memory and clock speed while ignoring the other specifications. The objective of this microprocessor is solely to operate the chess emulator, function as a teaching program for new users, and communicate with the microcontroller.

3.3.1.1 Arduino MEGA 2560

In order to ensure that the microcontroller will have enough GPIO for all the peripherals the Arduino MEGA 2560 was considered. It contains 54 digital I/O pins of which 15 of them are PWM enabled along with 15 analog inputs. While this is severe overkill for the scope of the project this ensures there will be enough pins to handle all the peripherals. The DC current supplied on the digital I/O is 20 mA and the 3.3 Volt pin handles 50 mA. The operating voltage is 5 volts, but it is recommended to input voltage of 7-12 volts. The memory available on flash (256 KB) and RAM (8 KB) should be enough to handle the simple inputs and outputs the microcontroller will feed. One potential problem, however, is the clock speed of 16 MHz which has a chance of not being fast enough for the 8 by 8 LED array to appear lit up simultaneously. Another serious problem is the cost of the microcontroller which sits at $38.00 plus shipping and handling.

3.3.1.2 Raspberry Pi Model B+

Using a microcontroller with a microprocessor is one such option to implement the chessboard design but there's also the option of running everything on just one micro-Linux computer. The Raspberry Pi Model B+ should have everything required to take that route. It sits at 40 GPIO although it has no ADC which is necessary for the project, so some ADC IC's would need to be incorporated separately on the PCB board. The DC current draw on the GPIO should not exceed 50 mA all together or 16 mA per individual pins. The operating voltage is 4.75 volts to 5.25 volts, but 5.00 volts is recommended. The main advantage of using this microcontroller is its ample memory (SDRAM: 1 GB) and processing power (700 MHz, 32-bit) and the fact that it has an operation system which is required to run the chess emulator. The final price is $24.95 which would be cost effective considering the other option requires a microcontroller along with a microprocessor.

3.3.1.3 MSP430FR2355 Launchpad

A more realistic microcontroller with enough GPIO and processing speeds to handle all the constraints of the project scope is the MSP430FR2355. With a $12.99 plus shipping and handling price it's also more cost effective than the MEGA 2560. Even though it's a familiar coding environment for the person in charge of the microcontroller, however, it requires slightly more work to code when compared to the Arduino. The MSP430FR2355 has 44 GPIO with 12-channel 12-bit resolution ADC which may be required to differentiate between 12 units from both teams. The DC current supplied on the digital I/O is 25 mA. The operating voltage is 1.8 Volts to 3.6 volts. The memory available on FRAM (32 KB) and RAM (4 KB) is questionable on whether it will be able to handle the program file size.

On the other hand, the 24 MHz clock speed and 16-bit processor should be enough for the LED array to appear simultaneously lit.

3.3.1.4 Microcontroller Selection

In order to simplify the design of the project it was decided that utilizing the Raspberry Pi Model B+ would be the wise decision since there would only be one micro-Linux computer with additional IC's as necessary as opposed to a microcontroller and a microprocessor. Another issue with using two devices as opposed to one is that both devices would have to communicate correctly between each other. That increases complexity in the design and additional accommodations would be required to run things correctly.

Table 3 Microcontroller Comparison Table

Features	ArduinoMEGA 2560	Raspberry Pi Model B+	MSP430-FR2355 LaunchPad
GPIO Count	54 Pins (15 PWM)	40 Pins	44 Pins
Analog-to-Digital Pin Count	15 Pins	0 Pins (Analog-to-Digital Converter IC Required)	12 Pins
Pin Current Max	Digital I/O: 20mA for all pins	Digital I/O: 50mA for all pins	Digital I/O: 25mA for all pins
Bit Count Processor	8-Bit Processor	32-Bit Processor	16-Bit Processor
Memory Storage	Flash: 256 KB, RAM: 8KB	MicroSD Card: 16 GB, SDRAM: 1GB	FRAM: 32 KB, RAM: 4KB
Clock Speed	16 MHZ	700 MHZ	24 MHZ
Total Cost	$38.00 Plus Shipping and Handling	$24.95 Plus Shipping and Handling	$12.99 Plus Shipping and Handling

3.3.1.5 Software

An important feature of the chess board is the ability to play a game of chess against a computer. To accomplish this feature, a chess engine is needed. A chess engine is a software program that can analyze a chess board to make the best possible moves. The chess engine knows what moves can be made and which

moves are illegal. Typically, chess engines are designed to run on desktop level hardware and don't have a graphic user interface. However, there are chess engines available that utilize a graphic user interface. For the purpose of this project, a graphical user interface is not needed.

Chess engines run on computers which can perform calculations much more efficiently than the human brain. Chess engines are generally difficult to beat because they can perform millions of calculations per second. Thus, they can analyze almost every possible outcome for any configuration of pieces on the chess board. However, chess engines are software and difficulty levels can be programmed into the chess engine. Chess engines can also be very complicated if needed. Depending on the stage of the game, a chess engine can employ different software techniques to employ different strategies suited for the game. A chess engine can determine when a game reaches the middle phase or the endgame phase. Because almost every scenario in the endgame has been played before, most chess engines use an endgame table base. A table base is a database of scenarios and moves to play in each case [2].

There are two options for implementing the chess engine. The first option is developed a custom chess engine to run on a cheap microprocessor such as an MSP430. Developing a chess engine would require high level algorithms that are beyond the software capabilities of the team such as various search trees and pruning techniques to find the best possible move. A popular technique is to use the minimax theorem [3]. The minimax theorem prioritizes reducing the gain the opponent can make in a move. A chess engine assigns a value to each chess piece based on the chess piece capability. Opponent chess pieces are marked with negative values. Thus, at the beginning of the game the chess engine keeps a value of '0.' The chess engine then makes a move based on which move will increase its score and keep the opponents score low.

However, developing a custom chess engine would limit the maximum challenge a computer could give. Due to the lack of algorithm analysis skills in the group, the most basic chess engine would only be intelligent enough to move random pieces. A random chess piece would be selected and based on what piece is selected, a random number generator would be used to determine what position on the chess board the chess piece moves to. Each move would be checked to make sure that the chess engine makes a valid move. With this setup, the chess engine would not prioritize taking pieces because every move would be completely random. This design would work to teach someone who has never played chess before. However, as the user becomes more familiar with how the game of chess is played the user can easily defeat the chess engine. Therefore, the difficulty of the chess engine would become the limiting factor to how the board can be used.

Another option for implementing a chess engine is to use an open source chess engine. While this option does not need a custom chess engine, an open source

engine does need a microprocessor that can run an operating system. Almost all open source chess engines are programs written in some variant of C and run on an operating system such as Windows, Mac OS, or Linux. There a few chess engines such as Stock fish that can run on mobile operating systems such as Android. There are chess engines that are written in other languages such as Python, Java, and Pascal. However, most chess engines are developed using C or C++ because the C language is very fast and efficient when performing complex algorithms. The C language is very well structured and extremely portable between different devices and operating systems. For computers, C++ is recommended, but for low computational microprocessors, C is preferred because the C compiler is more efficient at converting the C code to a machine language.

Using a pre - built chess engine is more efficient for several reasons. Using an open source chess engine saves time because the chess engine does not have to be designed and programmed from scratch. With an open source chess engine, simple modifications can be made to fit the design purposes of the project. There are several factors that must be considered when choosing a chess engine. First the chess engine must have a quick response time. If the chess engine takes too long to make a move, the game becomes stagnant and the User will become impatient. The chess engine must also have a high accuracy. The chess engine should not make illegal moves otherwise this will confuse the User and ruin the game. Finally, the chess engine should be powerful yet lightweight to run on the microprocessor chosen.

3.3.2 LEDs

In order to select an LED, it is important to first identify what exactly is the purpose of the LED in the design. In particular, the LED for this design will indicate a number of different things to alert the players during gameplay. First off, the LED will light up whenever a prediction is being made by the chess engine. There will have to be multiple LED color possibilities to indicate different movements and errors.

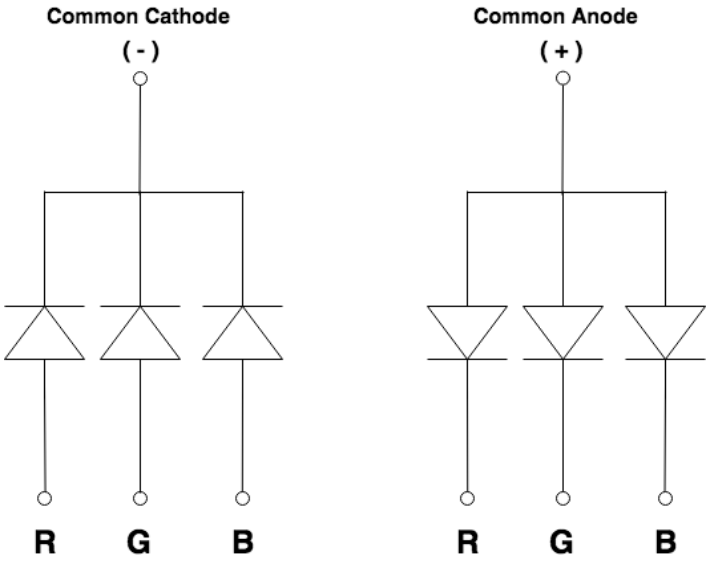
LED color scheme:

- Green - Player one's piece movements
- Blue - Player two's piece movements
- Red - Invalid movement

With this color scheme indication, it is evident to see that multiple color LEDs will be needed to in order to satisfy the requirements. Aside from that, it is also important to note how many LEDs will be needed in order to satisfy the populating of the full chess board. The initial thought is to have a single LED under each of the 64 squares on the board. After realizing that we would want multiple colors to indicate certain scenarios, it became apparent as to what type of LEDs would be chosen.

It was settled that the RGB LEDs would serve our purpose best since it provides a red, a blue, and a green LED all in one. RGB LEDs can either be common cathode or common anode. Common cathode means that when the circuit is setup, the common anode terminal will be tied to ground. Common anode means that when the circuit is setup, the common anode terminal will be tied to a voltage input rather than ground. A visual representation of the terminals and physical component of an RGB LED is shown below:

Figure 34 RGB LED



The LED can also have other attributes that contribute to its selection. Another consideration when it came to choose the exact LED was the way that the light would emit from a top view underneath another material. The RGB LEDs will be underneath the chess board, so the clarity of the light and its distribution of the light must be considered too. This is where the option of a clear or frosted tip RGB LED came up. The clear tip underneath the board would provide more of a point of light, as if it were a single beam focused on a pinpointed area. This is not what would be desired for a whole square underneath a chess piece because the visibility of light from the RGB LED would prove to be insufficient. However, the option of the frosted tip proves as a better choice in this situation. A frosted tip RGB LED will be able to have a more well-rounded dispersed display of light from underneath the board as opposed to the clear tip which would prove to be sufficient for this design choice.

After figuring out the exact type of LED we would want to implement, it was important to seek another route as a backup. Our initial thought to this design was to have an LED for each color underneath each square before inquiring on a 3-in-1 RGB LED option. Although this route would clearly be more expensive and contain more parts, it was still an option to be looked at. The table below

summarizes the possible LED's that was researched for implementation to our design:

Table 4 LED Comparison Table

Color	Type	Current Max (mA)	Voltage Max (V)	Pins	Quantity	Price
RGB	Frosted	20	3	4	100	$8.96
Red	Frosted	20	2	2	100	$5.69
Green	Frosted	20	3	2	100	$5.80
Blue	Frosted	20	3	2	100	$5.82

3.3.3 LED Display

The original idea was to go with another part at first that would perform the same functions, but it was brought to our attention that it would be in our best interest to use an actual LED driver as a part to control the LEDs that would light up the board. In searching for the LED driver, we first had to figure out what kind of LED's we would be using, as well as how many and how frequently amongst the board that they would be placed. When going about choosing the LED's, we would need enough colors to represent different functions of the game on the board. For example, player 1 will have a specific color such as green when the piece is lifted up and it's possible moves are revealed. If an incorrect move is made, then the tile below it would get a red LED to be lit to alert the player that it is not a valid move and so on. After thinking this through, we realized we would need multiple colors and it would not be wise to go through with buying separate colored LEDs. It was decided that the use of RGB LEDs were the smartest decision in terms of space and feasibility. An RGB LED is basically an LED that contains red, green, and blue LED's in it. They are able to use the three colors by themselves or these colors can be combined to produce a multitude of different hues of light. After making the decision to use these RGB LEDs, it made sense to go ahead and search for an LED driver that would support the use of these RGB LEDs.

The LED driver that the group came across was from Texas Instruments and its part name is the TLC5947. This part is a 24-Channel, 12-Bit PWM LED driver with an internal oscillator. The 24-Channels are very important when choosing the driver since we are driving RGB LEDs rather than a regular one colored LED. This part in particular served the purpose of having enough channels to drive all the

LEDs needed for a row of 8. Since there is red, green, and blue LED's in one RGB LED, then it is necessary to have a channel to control each of those colors separately. Therefore, 24-channels are necessary for this design. It is also important to note that the current max of the RGB LED's are important to note when making sure that the part will be able to supply enough to have them well lit. The max current output on this part totals to 30 mA which is enough to distribute to the LED's which have a current max of 20 mA.

The only issue with the TLC5947 was the speeds it could run at. After it was tested on a breadboard along with the RGB LEDs and the chess engine running in the background, it was determined that something faster was needed. This led to the final decision to go back to using shift registers to control the columns of the LED matrix.

Figure 35 TLC5947 Application Layout (TI Permission granted)

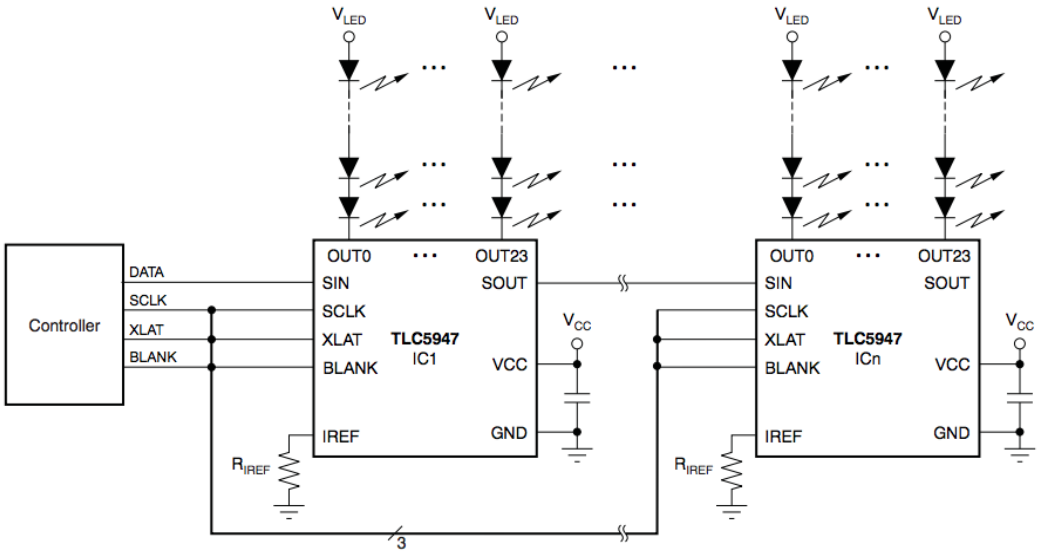


Table 5 LED Display Comparison Table

TLC5947	TPIC6B595
ALL IN ONE Package	Standalone shift register (Need 3)
24 Channels	8 Channels
PWM Control	No PWM
12bit programming per channel	1bit programming per channel
1ms To program 24 channels	5us To program 8 channels
$5 each	$1.50 each

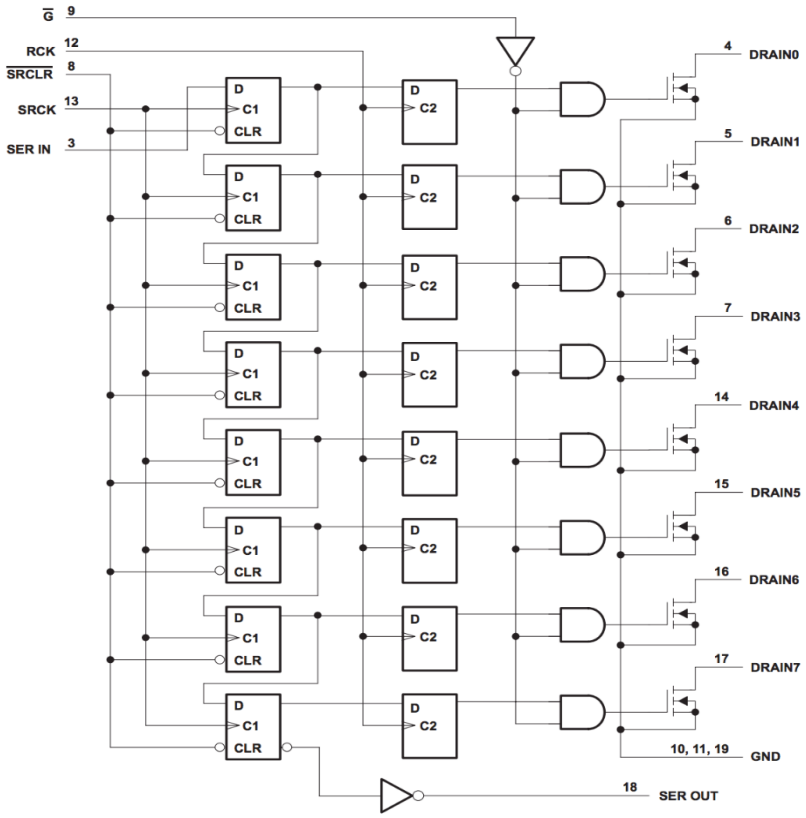
3.3.3.1 TPIC6B595

This shift register is the part that ended up being chosen for our final design which is the TPIC6B595, also made by Texas Instruments. This part is an 8-bit shift register that can transfer data on both the shift and storage registers on the rising edge of its clock. This part comes in two possible package choices. First choice was the SOIC (Small Outline Integrated Circuit). This package is a much smaller footprint and must be soldered to a board in order to be tested. The other choice of package option was the PDIP (Plastic Dual-in-Line Package). This package contains a larger footprint and does not need to be soldered in order to be tested. The PDIP package serves the best purpose for testing purposes. This package can be implemented right into the breadboard and fitted on so that it can be connected in conjunction with the LED driver to control the RGB LEDs.

Other possible 8-bit shift register options that could be implemented into the design are listed in Table 3-4. The main areas of concern when looking to choose the 8-bit shift register was the number of outputs and the continuous output current. It was important to make sure that enough current would be able to output so that the RGB LEDs, which have a max current of 20mA, would have enough current margin to work with when multiple LEDs are lit up. Also, since the 8-bit shift register will be interfacing with the LED drivers (8 BJT's) to control the RGB LEDs, we needed to make sure that there were enough outputs on the shift register to support our design choice. Having 8 outputs does not give us enough outputs to control all of the columns of the RGB LED matrix, however, we can daisy chain 3 TPIC6B595's together to make 24 outputs which satisfies our requirements. Daisy chaining involves just sequencing the three shift registers together so that they will be connected and run in accordance with one another.

Most importantly, using these shift registers greatly reduces the amount of time needed to program them with the chess engine that's running in the background. As listed in the table, it takes the TPIC6B595 about 5 us to program as opposed to the TLC5947 that takes 1 ms to fully program. This ultimately led to our choice of using the shift register over the TLC5947.

Figure 36 TPIC6B595 Internal Layout (TI Permission granted)



3.3.4.1 SN74HC595

This SN74HC595 shift register was another option that could be implemented into the design. Keeping options open for different parts play a huge role in design for flexibility and allow for solutions to possible part incompatibilities causing a show stopper. In our case, this part came in handy since it can work in accordance with the LED driver which was a part that was more cemented into our design as compared to the 8-bit shift array possibilities.

The reason for the choice of 8-bit shift register to be this one came from the compatibility of the RGB LED's and the overall design. The LED driver at the beginning was the least interchangeable part of the parts concerning the LED scheme of this design. Initially, common cathode RGB LEDs were going to be used as opposed to common anode RGB LEDs. This however changed but benefitted the implementations of shift registers in the design to control the columns as opposed to the LED driver we initially would use.

In the diagram of the SN74HC595 circuit layout, it shows how there are no N-channel MOSFET included internally on each output that would drive the source to ground. This is handy since we will be using the common anode RGB LED instead. This part comes in different style packages just like any other shift register parts that could be used. The first possible choice of package styles was the SOIC (Small Outline Integrated Circuit). This package is a much smaller footprint and must be soldered to a board in order to be tested.

The other choice of package option was the PDIP (Plastic Dual-in-Line Package). This package contains a larger footprint and does not need to be soldered in order to be tested. The PDIP package serves the best purpose for when it comes to component testing. This package can be implemented right into the breadboard and fitted on so that it can be connected in conjunction with the LED driver to control the RGB LEDs.

Figure 37 SN74HC595 Internal Layout (TI Permission granted)

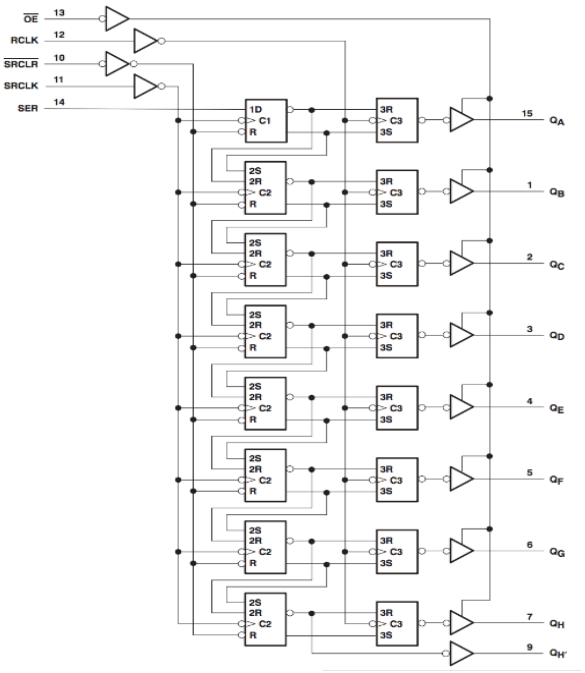


Table 6 Shift Register Comparison Table

Register	Outputs	Supply Voltage Max (V)	Continuous Current (mA)	Output Voltage Max (V)
TPIC6B595	8	7	500	1
SN74HC165N	2	7	25	0
TPIC6A596	8	7	1000	1.1
TPIC6A595	8	7	350	1.1

3.3.5 Development Board (LED Driver)

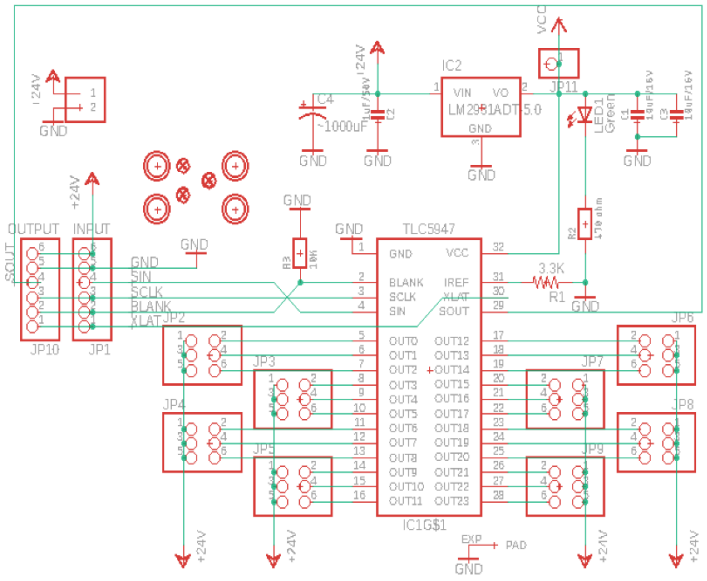
The LED driver selected for this design is the TLC5947. Many small microelectronic parts come in different package sizes which is important to consider. Many of the package sizes automatically assume the part to be soldered

to a board, which is usually the case. In our case, however, we need the piece to be tested before being soldered. Preferably, the part would fit ideally into the breadboard, so it could be used in testing.

The TLC5947 allows for samples to be given out by Texas Instruments for free, up to 5 of the parts. The main issue with this is that the all the parts (priced or free sample) all were not of any help for testing purposes due to their package specs. Two options were presented to choose from for this part. First was the HTSSOP (Thermally Enhanced Thin Shrink Small-Outline Package) to consider. This part has a very small footprint, with all the pins that would be tested in the millimeter range of values. The other package available for this part was the VQFN (Very Thin Quad Flat Non-Leaded Package). This package was also extremely small, which would cause a problem with testing. Either of these options could have been chosen but it would require extra tedious work with soldering to a board which is unnecessary when other options are available.

This brought our decision to choosing a development board to be used for testing of the TLC5947. The development board used was the Adafruit TLC5947. This board provides an easier way to test the TLC5947 because it provides it soldered to a board already, along with other components to regulate the voltage to 5V for a set input voltage. The most helpful aspect of this development board is the ease of soldering wires to the through holes provided for each of the 24 outputs in the TLC5947. This can be seen represented in the schematic for the J2 through J9 options.

Figure 38 TLC5947 Adafruit Development Board (Adafruit Permission granted)



3.3.6 Linear Voltage Regulator Selection

Most, if not all, of the devices required to complete this project are Low-Power IC's. Some integrated circuits may require a specific amount of voltage in order to operate correctly. Since the design will utilize a rechargeable battery, it's important to regulate the voltage of this battery to an acceptable amount to run all devices smoothly. The regulator chosen to for the task was the LM7805ACV. If for some case more voltage is required, then the LM7812ACV can be utilized.

Figure 39 LM7805ACV Schematic (TI Permission granted)

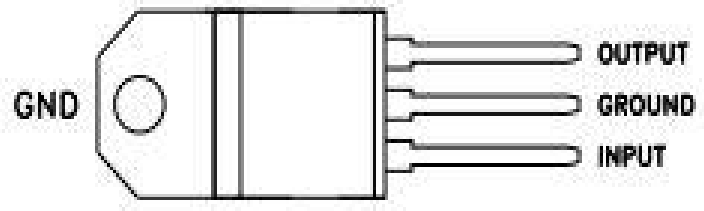


Table 7 Linear Voltage Regulator Specifications

Regulator	Output Voltage	Output Current	Input Voltage	Operating Temperature	Cost
LM7805	5 Volts	1.5 Amps	7 V -35 V	0 C - 125 C	$0.63
LM7812	12 Volts	1.5 Amps	7 V - 35 V	0 C - 125 C	$0.63

3.3.7 Analog-to-Digital Converter IC Selection

Going the raspberry route unfortunately doesn't include ADC pins on the microcontroller so incorporating a separate ADC integrated circuit on the PCB board will be required. Fortunately, this can easily be done with the raspberry and the hardest part about this is selecting the correct resolution for the job.

The MCP3008 offers 10-bit resolution giving us values 0-1023 to work with. It's programmable to provide four pseudo-differential input pairs or eight single ended inputs. The differential nonlinearity (DNL) and integral nonlinearity (INL) specifies the device at 1 LSB of analog accuracy. It offers 75-200 ksps max at input voltages 2.7 - 5.0 volts but the supply voltage can range from 2.7 - 5.5 volts. Typical standby current can be as little as 5 nA to 2 uA and typical active current is 320 - 500 uA at 5 volts. The price tag runs at $3.75 plus shipping and handling.

If a higher bit resolution is of a concern, then the ADS1015 may be a better option. The resolution for this ADC is 12-bits allowing us to work with 0-4095 values. It has

less inputs compared to the MCP3008 with two pseudo-differential input pairs or four single ended inputs. The differential nonlinearity (DNL) and integral nonlinearity (INL) specifies the device at 0.5 LSB of analog accuracy. The throughput offered ranges from 128 sps - 3300 ksps at input voltages 2.0 - 5.0 volts but can supply 2.0 - 5.5 volts. Typical standby current can be as little as 2 uA and typical active current can range from 150 - 200 uA. The price tag runs at $9.95 plus shipping and handling.

If even higher bit resolution is needed, then the ADS1115 may be required. The resolution for this ADC is 16-bits allowing us to work with 0-65536 values. It has less inputs compared to the MCP3008 with two pseudo-differential input pairs or four single ended inputs. The differential nonlinearity (DNL) and integral nonlinearity (INL) specifies the device at 1 LSB of analog accuracy. The throughput offered ranges from 8 - 860 sps at input voltages 2.0 - 5.0 volts but can supply 2.0 - 5.5 volts. Typical standby current can be as little as 0.5 uA and typical active current can range from 150 - 300 uA. The price tag runs at $14.95 plus shipping and handling.

Figure 40 ADS1115 Layout (TI Permission granted)

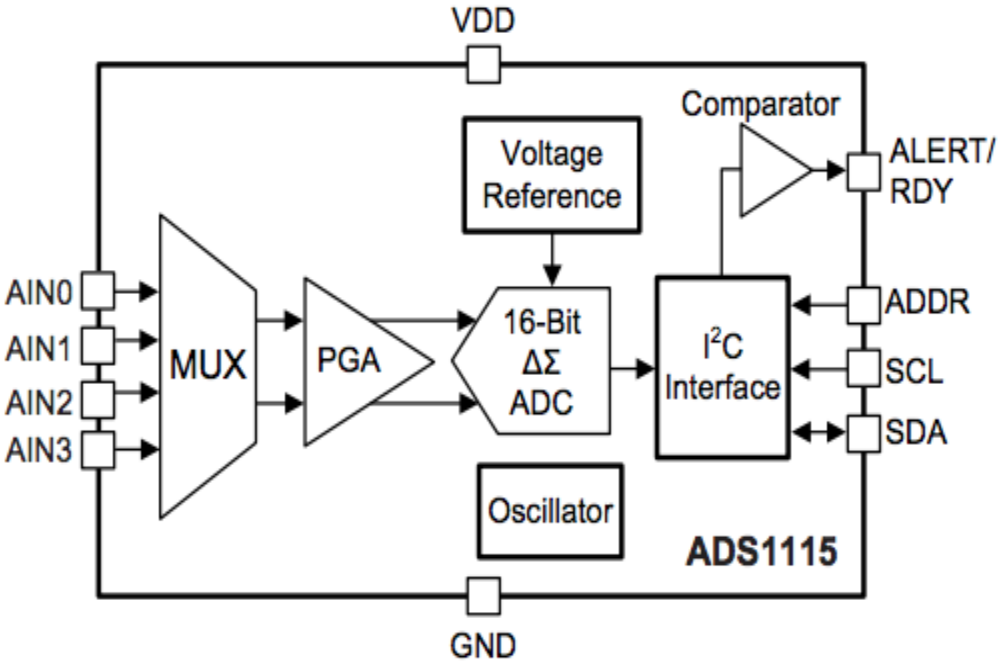


Table 8 ADC Integrated Circuit Specifications

Feature	MCP3008	ADS1015	ADS1115
Resolution	10-Bit	12-Bit	16-Bit
Inputs	8 Singles	4 Singles	4 Singles
Accuracy	1 LSB	0.5 LSB	1 LSB
Throughput	75-200 ksps	128 sps - 3300 sps	8 - 860 sps
Current Draw	Standby: 5 nA - 2 uA, Active: 320 uA - 500 uA	Standby: 2 uA, Active: 150 uA - 200 uA	Standby: 0.5 uA, Active: 150 uA - 300 uA
Operating Voltage	2.7 - 5.5 Volts	2.0 - 5.5 Volts	2.0 - 5.5 Volts

3.3.8 LED Drivers (Row selection)

Initially, the thought was to have shift registers also control the 8 rows that would be switching at a rate of about 60 Hz. This rate was chosen so that it would be undetectable by the human eye. When the final microcontroller was chosen to be the ATMEGA 2560, it came with more than enough GPIO pins that we could spare 8 of them to control the shifting through the rows.

This led to our selection of the 2N2222 BJT's that are connected in each row to act as a switch to allow for each row of the LED's to turn on when selected. This BJT provides up to a max of 800 mA which we would only need a max of 720 mA at worst case scenario for the LED's being turned on. Also, the switching speeds only need to be around 16 ms which this part would achieve. Lastly, it was very cost efficient since these BJT's are provided to us for free from our labs, so we had leftovers that we implemented into the design rather than having to buy more parts.

3.4 Research

Some topics throughout this project were more involved than looking up reference designs. This section highlights areas and topics that did not have readily available information that we could use to base our design on. This constituted for serious researching and development to achieve our requirements.

3.4.1 Chess Piece Identification

One of the most crucial features of this smart chess board is for the computer to be able to determine what and where a certain piece is on the chess board. This technology is around because in most smart chess boards all of the pieces from both teams have a designated starting position. The switches embedded in the chess board can determine when and where a single piece gets moved to at a time. This technology only works because the computer knows the original starting position. If we accidentally set up the board incorrectly the computer cannot do its work properly and we can get all sorts of errors.

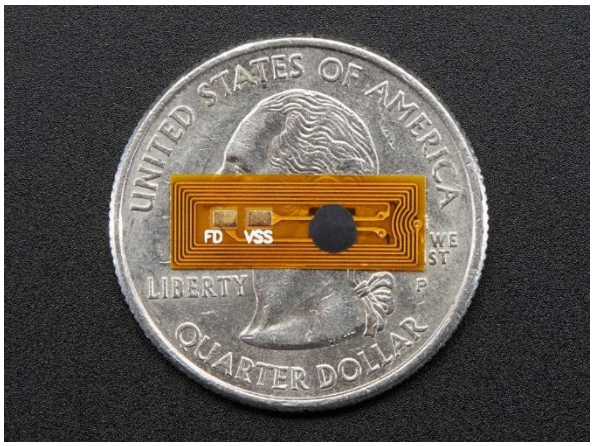
This smart chess board design will incorporate a piece identification function that can determine the location of a piece and what kind of piece it is. The point of this function is that two players can set up the board however they like, and the computer can still display the possible movements of the pieces. The software required to have the computer in real time be able to track every piece in any state and any starting position we determined to be too extensive and therefore is not in the scope of this project. The focus of piece identification is for training purposes or for the physical user to use.

Extensive research went into what technologies are available for this technology. Out of the dozen ideas, four main ideas stood out from the rest. These ideas are detailed in the sections below. All these four systems utilize different systems but all of them in the end relay the information back the microcontroller for processing.

3.4.1.1 Radio Frequency Identification System

The first system researched by the group was a Radio Frequency Identification (RFID) tagging system. This technology is new compared to the others. RFID systems have two main components, the tag and the reader. The tag is a small lightweight electromagnetic powered device that stores a unique signature. The size of these tags, shown below in the figure below, is small enough to be embedded in the bottom/inside of each chess piece. This design would incorporate passive tags which use the radio waves from the reader to power the device and allow the signal to be read. The reader is a significantly larger device than the tag. The reader contains the coils used to emit the radio signals and the technology to read the signature of the tag from those signals. These RFID systems run on a high frequency (HF) radio band. Common applications for RFID include Smart Card Readers for entry and exit of regulated areas, automatic vehicle tolling systems, animal tracking/identification. Medicine has even gone far enough to implant these tags in humans that contain information about that person including ID and even credit cards.

Figure 41 Size of the RFID tags



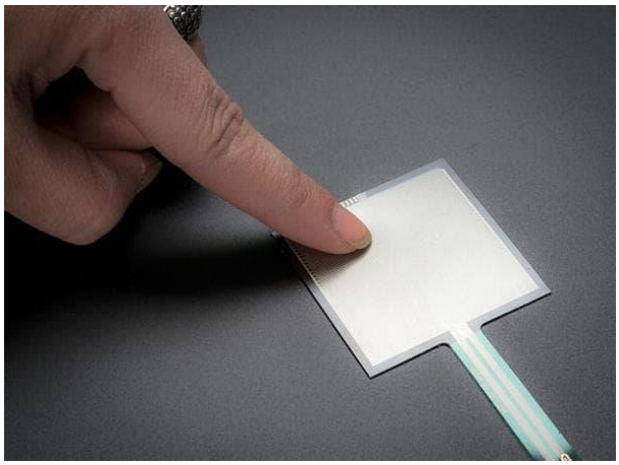
The implementation of this technology in the smart board would have been the best choice if there were no budget constraints. A RFID tag would be planted in each chess piece. Each piece would have its own unique signature code. A reader would also have to be installed underneath each tile on the board so that no matter where the pieces land they can always be read. This design is optimal because the RFID system will work through the tiles because the radio signals can penetrate the tiles.

The biggest issue with this system is cost. The cost of a single RFID tag is about one to two dollars. We need 32 tags so between 32 and 64 dollars. The RFID readers, however, run about $30. We need 64, one for each tile. Do the math and this is about $2000. Extremely expensive in terms of this project. We also researched the idea of only using one RFID reader with 64 separate antennas instead of 64 readers to lower the costs. The biggest issue with this is multiplexing radio frequency signals. Maintaining signal integrity is difficult and increases the price of the system. We determined that treading that direction would lead to increased costs and development problems in the long run.

3.4.1.2 Pressure Sensing System

The second system we research extensively during development of this project was a pressure sensing system. This system would utilize 64 pressure sensors in each of the 64 tiles on the board. The figure below displays a common pressure pad size. Each chess piece would have material removed or added so that each unique piece has a unique weight. The pressure sensors can detect the weight of the object and relay the information back to the microcontroller.

Figure 42 Pressure Sensor Size



This system involves the most mechanical design and would bring unique challenges up front. There would have to be an invisible bed under the tiles and LEDs but still have contact with the tile so that the chess pieces can be measured properly. There would have to be a very tight tolerance in the weight of the hardware used above the pressure pad so that the measurement is accurate from tile to tile. The Pad does not allow light to pass through it, so it would have to be staged in a way where the LED light can pass through the tile and not be hindered by the pressure pad. The number and complexity of the mechanical and hardware related issues to overcome is the main reason we didn't decide to go with this system. The cost of the pressure pads also ran about 8 to 10 dollars each. This would amount to over $500 dollars.

3.4.1.3 Visible Light Sensing System

The third identification system involves the use of photodiodes. Photo diodes in short detect the amount of light entering the sensor and convert it to an analog voltage that can be interpreted by the ADC of a microcontroller. The figure below displays a photodiode. A photodiode would be implanted in every tile within the chess board. A hole would need to be drilled in every tile so that the tile does not hinder the light trying to enter the sensor. On the bottom of each type of chess piece there would be a different color material attached. This material would be determined based on the amount of light that reflects off it. This is the basis on how we would differentiate between the pieces on the board.

Figure 43 Photodiode

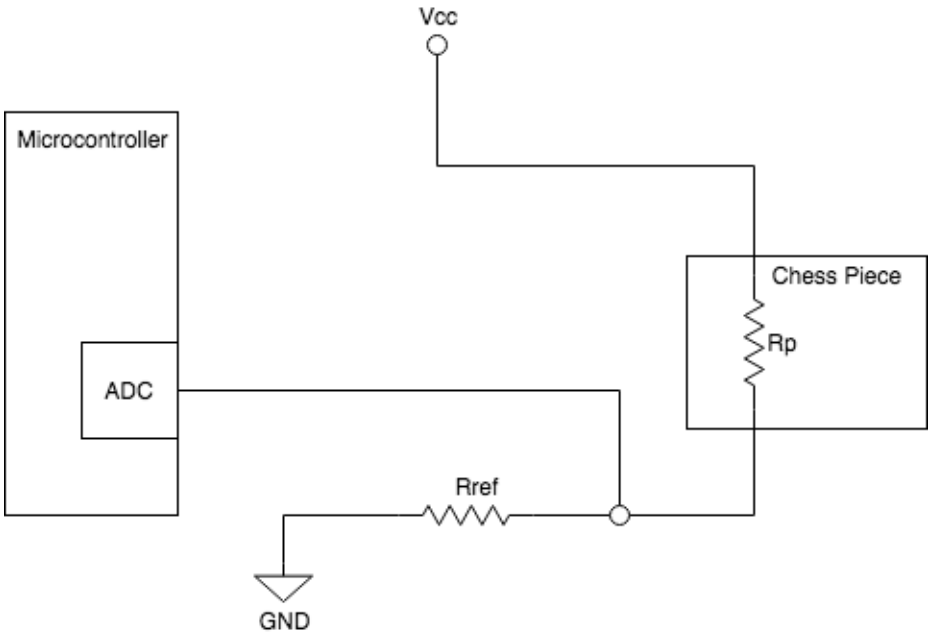
Issues with this design are still complicated enough so that we didn't go with this system. First off, as protective system would need to be in place so that ambient light does not interfere with tiles where there are no pieces. The ambient light would also penetrate the opaque tiles and interfere with the sensors that have pieces on top of them. The software would get complex because there would be so many scenarios of reflected light with the same piece because the LED will output so many different colors and levels of brightness. The number of unknown variables with the light sensing and the sheer perfectionism needed to make sure all 64 tiles on the board work the exact same and detect light the same is the reason we did not go with this design.

3.4.1.4 Sampling Circuits

The fourth and last option we researched for the chess piece identification system was along the lines of a resistive circuit matrix. Every tile on the chess board would have two conductive contacts on it. These contacts underneath the board would be connected in a matrix like fashion. Each of the 32 chess pieces would have the same contacts underneath the chess pieces, arranged in a way so that they line up with the tile contacts. All the pieces and tiles would be in the same position. Within the chess piece the two contacts will relate to a resistor whose value is unique to the type of chess piece. A total of 12 unique pieces.

The initial theory behind this was that we can create a switching circuit matrix that can select a specific tile based on a switch that is activated. This circuit would in its basic form be a simple test circuit with a DC source passing current through the resistor within the chess piece and then through another reference resistor. The ADC on the microcontroller can then sample the voltage between the two resistors. See the figure below. Using voltage divider, since we know the source voltage, the sampled voltage and the value of the reference resistor we can mathematically determine the value of the resistor in the chess piece. The microcontroller can now determine which piece was selected. Initially we were planning on using push button switches in the tiles to activate a tile.

Figure 44 Basic Identification circuit for one tile



With that method you would select a piece by pushing down on the piece and activating the switch. The switching matrix circuit could be changed via the microcontroller and the circuit can change to any configuration quickly. The big problem with this design is instead of push button switches the team decided to go with reed switches, magnetically controlled switches. This cause a significant change to need to be made because the reed switch is “activated” when the chess piece is removed from the board. If the piece is removed from the board then we cannot sample the circuit. This idea no longer works.

With the change from push button switches to reed switches brought changes to the identification system. Since we would be using the reed switches in an “activated” state while there is a chess piece resting on the tile, and the way that tile would be selected by the user is lifting the piece of the board, therefore “deactivating” the switch. This is the exact opposite of the way the previous concept was laid out. After more research the final concept the team came up with was very simple. Get rid of the switching matrix and connect all the tiles together in parallel. Sample the entire system at once. We continually sample the board and once a switch is “deactivated”, indicating a piece has been selected and lifted off the board, we will sample the system again. By comparing the difference in samples before and after the event we can determine the value of that resistor. We know the location as well based on the event of a switch deactivation.

This system ends up being a little less complicated than building a whole switching circuit matrix. It’s cheaper too. The contacts in the chess pieces and the tiles will

now serve a double purpose in that they will be magnetic now and connect the resistor to the board and activate the reed switch.

3.4.1.5 System Comparison

Out of the four systems we did extensive research on only one stood out as a feasible option in all the ways we were looking for. The main factors we looked at while doing the research was cost, development ability, and implement ability. Table 3.4.1.5-1 shows the cost comparison between the four systems. The first two systems costs were way out of the budget so that was one deciding factor for why we did not implement those systems. The third and fourth system were very reasonable and were the encouraged systems we continued to develop. The development feasibility comparison is shown in table 3.4.1.5-2. We rated the feasibility based on a scale 1-10. One being not possible at all and ten being very easily done with reference designs already in place. The first two systems scored in the positive range due to the reference designs and examples available. The third and fourth systems scored low due to the lack of examples and complex hardware/software development needed.

Table 9 Piece Identification Cost Comparison

System	Item 1 Cost ($)	Item 2 Cost ($)	Total Estimated Cost	Rating (1-10)
RFID	RFID Tags (3)	RFID Reader (30)	$2016	1
Pressure Sensor	Sensor (10)	Various weights (30)	$670	3
Photodiode	Photodiode (0.1)	N/A	$6.50	9

Table 10 Development Feasibility Comparison

System	Explanation	Rating (1-10)
RFID	This development is very feasible, reference designs are readily available. Lots of examples to work from. The hardware itself is easy to use and setup. This system is easy to implement with the microcontroller so it scores in the high range.	8
Pressure Sensor	The development is feasible. Reference designs are available. There are tutorials and other examples available to work from. The connection to the microcontroller can get complex so this system scores in the upper middle range.	6
Photodiode	The development is difficult. Changes in ambient light, colors of the LEDs and brightness of the LEDs will all affect the nominal value of the photodiode. Differentiating the difference between tiles and locations on the board will be hard due to the uneven lighting on the board. The design itself will be fairly simple as the hardware itself is simple but the software for this system to work properly would be a project in of itself. The software is very complex so this system scores very low.	2
Sampling Circuit	The development is moderately difficult. There are very few reference designs and no examples. The hardware development behind the scenes is fairly difficult. But not a show stopper. The software development will be fairly easy as it is just ADC manipulation and multiplexer manipulation. The hardest part of this system would be calculating the best resistance values to gain the best ADC resolution. This system scores in the lower medium range.	4

Table 11 Implement Ability Comparison

System	Explanation	Rating (1-10)
RFID	The installation and prototyping of this system would be complex. The RFID reader antennas under each tile must be close enough to the surface so that we get a good read from the tag, but it can't be too close because the LED still needs to be far enough from the tile so that the light distributes evenly. There is a possibility that both features won't work together perfectly, and we would have to cut back on one of the two to get a happy medium. This system scores medium low due to this.	4
Pressure Sensor	This systems installation gets a nearly impossible score. The size of the pressure pads would cover almost the entire size of the tile. Light cannot pass through the pad. There would have to be a complex mechanical system in place to separate the pressure pad from the tile to allow light through while also distributing weight evenly on the pad, so it is consistent. This needs to happen consistently time 64 tiles. The mechanical nightmare of this issue is the reason this system gets a score of almost impossible.	1
Photodiode	The implement ability of this system is very simple. The tiles need a hole for the sensor to sit flush in. The sensors are so small that LED light distributing through the tile won't be hindered at all by the sensor. The chess pieces will only need a colored material installed on the bottom. This system scores high because of this.	8
Sampling Circuit	This system scores in the medium high range. The installation of the system is very similar to the photodiode system except we are adding conductive terminals. These terminals won't interfere with any LED light or reed switch systems. In the chess pieces we would need the resistors and magnets installed. Keeping the magnets in consistent locations across all the pieces is a challenge.	6

The comparison tables provide a logical system to determining the best system to use. Based on the sum of the ratings we see what might work the best. The higher the score the better the overall system rating. The first system, RFID, the total score was 13. The second system, pressure sensor, scored 10. The third system, photodiode, scored 19. The last system, sampling circuit, scored 20. The highest score is the system we went with. Even though the last two were a close match the photodiode proposes development challenges that could result in a system that does not function in the end. The sample circuit system does not have any foreseen issues like that. The identification system that this project will be based on is the fourth system, sampling circuits via an ADC.

3.4.3. Microcontroller vs Microprocessor

When designing the smart chess board, the original plan was to use a microcontroller such as the MSP430 developed by Texas Instruments. However, upon researching chess engines, it was discovered that most chess engines run on an operating system. Microcontrollers are not able to run operating systems because an operating system needs a microprocessor to run. To use a microcontroller, a chess engine would need to be written in C or Assembly to be able to run. However, writing a chess engine is a complex task and requires hardware resources such as RAM and memory that are not sufficient on a microcontroller. One advantage of a microcontroller is that most microcontrollers have flash on board. Flash allows the microcontroller to return to its previous state in the event of a crash. A microprocessor does not have flash and thus must be rebooted if a crash occurs.

The next design involved both a microcontroller for the hardware and a microprocessor for the chess engine. Having both a microcontroller and a microprocessor would increase the cost of the smart chess board significantly. Also, connecting both devices would be challenging as most microcontrollers and microprocessors use separate languages to interface with other hardware components.

Upon further investigation of devices such as the Raspberry Pi, it was determined that the Raspberry Pi was able to perform both functions as a microcontroller and microprocessor. This would reduce of the final cost of the smart chess board and keep the hardware components and wiring to a minimum. Going with a microprocessor keeps the hardware design simple and easy to implement with various other hardware components.

Table 12 Microcontroller vs Microprocessor

Features	Microcontroller	Microprocessor
Cost	~ $5 - $20	~$35
RAM	128B - 10KB	~ 512 MB
Memory	Limited - around 16 KB	Unlimited - SD storage
Power	Low Power Mode	No Low Power Mode
Flash	1 - 60 KB	None

3.4.4 Operating Systems

This section lists the research of different operating systems to run on the microprocessor for the chess engine. Each operating system was tested to see if it will run on the microcontroller and how stable each operating system was. The efficiency of the startup time for each operating system was considered. A quick operating system that boots up quickly is beneficial to responsiveness of the system in order to make the smart chess board intuitive. The operating system must be able to load quickly when the smart chess board is turned on.

3.4.4.1 Windows IoT Core

The Windows IoT Core operating system is a slimmed down version of Windows 10 designed to run on low power microprocessors such as the Raspberry Pi. There a few downsides to using this operating system. This operating system requires a license which costs thirty-five dollars. Also, the operating system will only run applications that are available through Microsoft's app store. This is because the operating system has been modified from the original Window 10 to only run on ARM processors. Thus, the chess engines will not run on the Windows IoT Core operating system because the chess engine programs are not available on Windows application store. The Windows IoT Core operating system is also relatively new compared to other operating system and many bugs may appear during the development of the smart chess board. The operating system requires a lot of storage. The recommended hard drive space for the operating system is two gigabytes [10].

3.4.4.2. Ubuntu Core

Ubuntu Core is a lightweight version of the Ubuntu operating system. This version of the operating system is designed to run on less powerful microprocessors such

as the Raspberry Pi. The Ubuntu Core operating system is based on the same Linux kernels as Ubuntu, unlike the Windows IoT operating system which has been significantly rewritten. Thus, any program that can run on the desktop version of Ubuntu, can run on the Ubuntu Core operating system. Therefore, all the chess engines will be able to run on Ubuntu Core because they are all available for Ubuntu. The operating system is also free, unlike the Windows IoT operating system.

The image size of the operating system is also very small at around three hundred and fifty megabytes. Because the size of the operating system is so small, Ubuntu Core is also very secure. Since Ubuntu Core shares a lot of code with the desktop version of Ubuntu, there is an enormous amount of documentation for Ubuntu Core. This illustrates that there won't be any headaches when installing the chess engine and hardware components. Updates to the operating system can be done wirelessly if needed and the operating system supports full rollbacks if an unexpected issue arises [11].

3.4.4.3 Kali Linux

Kali Linux is another Linux operating system that can run on low power microprocessors such as the Raspberry Pi. Like almost all Linux distributions, Kali Linux is a free operating system. Kali Linux is a cyber security-oriented Linux operating system. It comes standard with password crackers, encryption programs, and security software. Unlike Ubuntu Core, Kali Linux has not been redesigned to run on low power microprocessors. The boot up time for Kali Linux is significant on a Raspberry Pi and requires eight gigabytes of storage for the operating system alone. Research shows that while Kali Linux can run on a microprocessor, the operating system is not suited for the needs of a smart chess board [12].

3.4.4.4 Raspbian

The Raspbian operating system is a Linux operating system that was designed specifically to run on the Raspberry Pi microprocessor. The operating system is based off another desktop Linux operating system, Debian. There are two version of the Raspbian operating system. One version comes with a graphic user interface that makes navigating the operating system extremely easy and intuitive. However, the size of the operating system is about four gigabytes. The other version of the operating system does not come with a graphical user interface. The command line is the only tool available to navigate around the operating system environment. The size of the operating system for this version is very light at about four hundred and fifty megabytes.

Raspbian is the most popular operating system used for the Raspberry Pi. The Raspberry Pi is also the most popular microprocessor used for doing it yourself

(DIY) projects. Due to the popularity of the Raspberry Pi and Raspbian operating system, there is an enormous amount of documentation to help with implementing the operating system and the chess engine for the smart chess board. The chess engine will be able to run on Raspbian because Raspbian uses many of the same kernels as the desktop counterpart, Debian [13].

3.4.4.5 Android

Another operating system that can run on microprocessors is Android. Android is an open source operating system that is based on various Linux kernels. Android was initially developed by Google and is primarily used on mobile devices such as cell phones and tablets. Recently, Android has expanded to wearables and smart speakers. Android is a very power operating system that uses very little hardware resources. However, chess engines are not designed to run on the Android operating system. One option is to use a smartphone app to simulate a chess engine. However, Android applications are locked down in an .apk file. Therefore, the chess engine in a smartphone app is not able to be modified. To do so would require an extensive knowledge of Android apps and how to decrypt application files which is beyond the skill set of the team members.

3.4.4.6 Other Operating Systems

Numerous other operating systems were researched for consideration. However most other operating systems have been created for very specific functions and are not very versatile. CentOS is a Linux based operating system that is designed to run simple servers and wireless networks. Other operating systems such as OpenMediaVault and OSMC are designed to host home media servers. Finally, operating systems like RetroPie are used to develop mini, portable gaming consoles.

Table 13 Operating System Comparison

Operating Systems	Memory	Speed (seconds)
Windows IoT	2 GB	~45
Ubuntu Core	350 MB	~20
Kali Linux	8 GB	~25
Raspbian	450 MB	~10

3.4.5 Chess Engines

This section lists the research put in to various chess engines. Selecting a chess engine that is efficient is an important aspect and requirement of the design specifications. When comparing chess engines, the processing power of each engine was measured using central processing unit analyzer software. The time it took for each chess engine to make a move was also measured. The chess engine with the quickest time to make and move and the least processing power use will be the chess engine that will be implemented.

Each chess engine tested must also have a method of outputting the move data. The move data is needed by the shift register to highlight the LEDs indicating were the chess engine wants to move. When the chess engine makes a move, the chess square with the piece that needs to be moved will be highlighted and the position of where the chess piece will go will also be highlighted. Thus, the move data must be able to be translated into data that the shift register can use to highlight the appropriate LEDs.

The chess engine is only needed when the user wants to play against the computer. There will be a mode selector modified into the chess engine. When the user wants to play with another user, the mode selector will be toggled, and the chess engine will be turned off.

Each chess engine was tested in a virtual machine running the Raspbian operating system. This is to best simulate how long and how much power the chess engine will use in the final product. However, because a virtual machine is being used to run the Raspbian operating system rather than a Raspberry Pi, the speed of the chess engine might be slower when tested. This is due to the fact that a virtual machine must share hardware resources with the main operating system running on the computer. During testing, the main operating system was Windows 10.

3.4.5.1. Stockfish

Stockfish is the most advanced chess engine on the market currently. Stockfish is an open source chess engine that can run on Linux, Windows, MacOS, Android, and iOS. Stockfish has an enormous variety of difficulty levels and has been proven to beat Grandmaster level chess players on the highest difficulty setting. The chess engine was originally designed by Tord Romstad, Marco Costalba, and Joona Kiiski. Currently, the chess engine and applications are being developed and maintained by the chess community and Stockfish forums.

Due to the difficult of the Stockfish chess engine, the chess engine may not be suitable for the smart chess board. The smart chess board is marketed towards beginning chess players who may not fully understand the game and the rules of

chess. Thus, the Stockfish engine must be set to one of the lowest difficulties if implemented. If the chess engine is too strong for beginner users, it will be difficult for beginning chess players to learn the game and ruin the enjoyment of playing chess. If the Stockfish engine is implemented, another feature to be implement would be a way for the user to modify the level of the chess engine.

Stockfish is one of the most popular chess engines available. Due to the popularity, there is lots of documentation on how to modify the chess engine and how to deal with problems with the chess engine taking too long or causing overheating.

3.4.5.2 Houdini 6

Houdini 6 is known as the second most powerful chess engine behind Stockfish. Houdini 6 was recommended by various master chess players. However, Houdini 6 is not a viable chess engine for the smart chess board. The Houdini 6 chess engine only runs on Windows and is not compatible with Windows IoT or any Linux distributions. The hardware requirements for Houdini 6 also go beyond the capabilities of microprocessors such as the Raspberry Pi. Finally, the Houdini 6 chess engine is not free and costs about one hundred dollars.

3.4.5.3 MicroMax

MicroMax is an open source chess engine that was developed by Harm Geert Muller. The MicroMax chess engine is written using the C language and is one of the smallest chess engines ever made [19]. The MicroMax chess engine is written in about 130 lines and is less than two kilobytes in size. The small size of the chess engine allows the chess engine to run on a microcontroller such as an Arduino or a MSP430 rather than a microprocessor such as the Raspberry Pi.

The MicroMax chess engine is configurable to reduce or increase the memory usage and the time the chess engine takes to make a move. The minimum memory the chess engine needs is about 72 kilobytes and can makes moves in under two seconds. In certain scenarios the chess engine took about a maximum of four seconds. With the chess engine set to a low RAM usage setting, the max CPU load was three percent at maximum. As the RAM usage increases however, the CPU usage can reach full capacity at one hundred percent. However, beyond those configurations, there is nothing else in the code that can be changed.

Because the chess engine focuses on using as little memory as possible, the readability of the code has been severely sacrificed and is unreadable except to the author and the most advanced software engineers. To overcome this issue, there are comments within the code which give a brief yet vague understanding of what each line of code does, and Harm Geert Muller has a website that hosts an

enormous amount of information detailing different chess engine techniques used by the chess engine.

Besides the small size of the chess engine, another benefit to using MicroMax is the ability for different game modes. MicroMax supports playing against another player or against the computer. However, one downside is that the computer level is not adjustable. The computer plays at one level and cannot be customized to be made stronger or weaker. However, the computer difficulty is not too strong that beginner players do not have a chance of winning. The chess engine also supports move checking on both sides. The chess engine cannot make an illegal move and the chess engine will not make a move until a valid move is made by the user.

3.4.5.4 Faile

Faile is another open source chess engine written in C by Adrien Regimbald. The chess engine is free to use so long as credit is given to the owner of the chess engine. The Faile chess engine is extremely old with the last version being made available almost eight years ago. Due to the age of the chess engine, there is very little documentation about Faile other than the comments of the code and the documentation provided in the chess engine download [20].

The Faile chess engine checks if user has made a valid move. The chess engine does not make a move until a valid move is entered. The chess engine performed relatively efficient with an average time of ten seconds. When using a chess book, the engine seems to be able to calculate moves much quicker. The Faile chess engine uses about thirty percent of the CPU power and is very resource intensive.

The chess engine does not a player versus player mode. Thus, the code will have to be modified to include a player versus player mode. The extremely sparse documentation provides little detail about how the chess engine works and the comments within the code are not very robust. Designing new functions to incorporate a player versus player mode will require an enormous amount of unit testing and debugging of the existing code to understand how the chess engine works.

The Faile chess engine was designed to run on 32-bit Windows operating systems. There is a Linux version as well. However, the chess engine has not been updated in years and the chess engine does not compile under the modern C compiler standards. Thus, a graphic user interface known as Xboard was needed to test the chess engine. This is not ideal for the requirements of our chess engine. The Faile chess engine is unable to compile in the command line environment and needs a graphical user interface to function properly. This would make it impossible to send data about where a user plays to the chess engine because there is no GUI for our chess board. While the Faile chess engine performs reasonably well, it cannot be considered for the smart chess board.

3.4.5.5 GNU Chess

The GNU chess engine is an open source engine that is extremely robust. The GNU chess engine was originally created in the mid-1980s. Over the last three decades, various people have modified and refined the GNU chess engine. Other open source chess engines have been incorporated into the GNU chess engine such as Fruit, Cobalt, and Gazebo all of which are high level chess engines [21].

The GNU chess engine has an enormous number of features. The GNU chess engine also includes an entire webpage of detailed documentation for every feature and aspect of the chess engine. The GNU chess engine supports a player versus player mode as well as a player versus computer mode. The GNU chess engine can run in either the terminal or through a graphical user interface such as Xboad. The GNU chess engine is also able to record games and output an analysis of the game to a text file.

During testing, the GNU chess engine seems very competent. The GNU chess engine includes legal testing of each move and never makes an illegal move. If the player makes an illegal move, the GNU chess engine sends an illegal move message and waits for a legal move to be made. The GNU chess engine difficulty cannot be changed. The GNU chess engine difficulty can also vary depending on the resources given to the engine to use such as RAM and CPU power. The GNU chess engine is powerful yet efficient. It performs most moves is about five seconds and uses about fifteen percent of the CPU power.

3.4.5.6 Sunfish

Sunfish is an open source chess engine written by Thomas Dybdahl Ahle. It is the only chess engine tested that is written in Python instead of the C language [22]. The Sunfish chess engine have very detailed documentation on GitHub and seems to be very flexible. The Sunfish chess engine is also very lightweight, like the MicroMax chess engine. Although, the Sunfish chess engine does not employ many chess engine techniques to calculate a move. Sunfish is also very barebones and leaves a lot of features unimplemented.

Despite having detailed documentation, there does not seem to be an active community devoted to further developing the Sunfish chess engine. The chess engine does not support a player versus player mode. However, unlike Faile, the source code for Sunfish is extremely readable and can be modified easily. This is because Sunfish was designed to be modified and used to test different search algorithms and features. The Sunfish chess engine supports terminal commands as well as input through a graphical user interface.

Although Python is a slower language than C in general, the Sunfish chess engine performs reasonably well. The Sunfish chess engine makes moves within four seconds and uses only five percent of the CPU to perform calculations. However, because the Sunfish chess engine is written in Python, a microprocessor such as the Raspberry Pi will be needed to run the engine.

Table 14 Chess Engines

Chess Engine	CPU Load	Time
Stockfish 9	< 3 %	~ 20 seconds
Houdini 6	N/A	N/A
MicroMax	3%	~ 2 seconds
Faile	30%	~ 10 seconds
GNU Chess	15%	~ 5 seconds
Sunfish	5%	~ 4 seconds

3.4.5.7 Final Chess Engine

The chess engines have been narrowed down to two possible candidates. The chess engine that is chosen will depend on other parts of the design. If a microcontroller is used, then MicroMax will be the chosen chess engine. MicroMax and Sunfish are the only chess engines that are lightweight and efficient enough to run on a microcontroller. MicroMax would be chosen over Sunfish due to its responsiveness and that it is written in C. If the Sunfish chess engine were used, a Python compiler would also need to be installed to the microcontroller to compile the chess engine.

If a microprocessor is chosen, then Stockfish 9 will be chosen. MicroMax is also another prime candidate for a microprocessor due to the lightweight design of the chess engine. However, Stockfish would be used due to its adaptability in computer difficulty and the immense documentation.

One downside to the microprocessor is that there are fewer input and output ports on a microprocessor compared to a microcontroller. Depending on the number of devices connecting to the microprocessor, a microcontroller may be needed.

3.4.6 Mechanical Design

The research done on the mechanical design is in this section. The term mechanical in this section is used very lightly. The things that fall under this category include the physical box design and construction and the other non-electrical materials we used in this project.

3.4.5.1 Materials

Most of the research done on the mechanical design was in materials. Specifically, the material of the chess board housing. The tools available to students at UCF for milling, cutting and shaping are more than adequate for the design we are looking to. The computer numerical control (CNC) milling machine and laser cutter machine in the UCF innovation lab are the machines of choice for our precision hardware components. We are limited to materials that these machines can work on. Those materials include wood, plastics and soft metals.

The wood material that we would use is plywood. Plywood is a manufactured wood that uses pine chips and sawdust glued and compressed together in sheets in a random fashion. These sheets are then crossed and pressed into layers. Plywood is a very durable and stable material. Over time humidity does not change the state of the material in length or width. If moisture does get in the wood, it can expand in the depth of the material. Wood is very easy to work with and almost all tools can cut or shape wood. The types of plywood that can be purchased vary on thickness and layers. The more layers of plywood the stronger and more stable the wood is.

There are many different types of plastics and we only compared the cheapest and most available. Polyethylene terephthalate (PET) is the most common plastic and therefore the cheapest. This material is hard, durable and is not affected by moisture. It is basically as easy to machine as plywood but because the plastic is not as prone to chipping during milling. This material can get more precise machining done. The plastic comes in all colors, thicknesses and sizes. The other plastic researched is acrylic. This plastic is harder and more brittle than PET, but it is clear, it acts just like glass, but it is easy to machine just like PET. It is prone to chips, cracks and breaks though.

The softest metal, that price does not become an issue, is aluminum. Aluminum sheets are fairly inexpensive and common enough to buy at a local hardware store. They come in many thicknesses and sizes. They are the most precise material here when it comes to manufacturability. They are also the hardest to machine due to how tough this material is. Aluminum can also be conductive depending on the alloy so this can become an issue.

To summarize the differences between materials. Wood and plastic are the top choices due to their easy manufacturability. They both run about the same price and function the same. Plastic is more stable over time than wood, but wood is a little easier to work with. Acrylic plastics are ideal for the clear/opaque tiles in the board itself. When it comes to the design the choice of mechanical materials will lie in the preference of the team.

4.0 Standards and Design Constraints

In this section, industry standards and design constraints relevant to the project will be thoroughly investigated. Researching the standards and design constraints will ensure that the project implementation goes smoothly, and all devices are within their datasheet specs. The first section shall cover engineering standards that are relevant to the technologies used in the project. The second section shall cover some design constraints that need to be dealt with to successfully implement the design. Taking into consideration these two sections should help avoid some future headaches that could arise.

4.1 Standards

It will be vitally important to properly research the standards associated with devices that will be utilized on the chessboard. The IEEE standards shall be used since the scope of the project deals mostly with electrical devices and concepts. Another important standard organization that should be considered is the American National Standard Institute (ANSI) which sets the standards for increasing America's impact on global competition as well as ensuring that American quality of life is sustained by incorporating safeguards and universal compatibility. Without these standards, simple tasks such as wireless communication with phones would become increasingly complex with individual communication companies incorporating their own engineering design which may not be compatible across all devices. Wireless communication has its own set of standards (IEEE 802.11) which requires infrastructure to be built under unique parameters. These parameters are what allow all cellular devices to be compatible when communicating with cell towers regardless of phone provider.

4.1.1 (IEEE 1118.1) Standard for Microcontroller System Serial Control Bus

IEEE 1118.1-1990 is a standard used to provide a protocol for the serial bus interconnection of independently manufactured devices. This standard is a communication protocol that helps with making devices that are limited in their reprogrammability stay consistent throughout devices to ensure compatibility across manufactures. Specifically, the protocol is optimized in areas such as instrumentation, distributed data acquisition systems, and control devices. The standard starts with mandating a protocol for multi-drop bit-serial communication between interconnected devices.

A key feature that distinguishes the bus in this standard compared to other bit-serial data-communication buses is that it is designed for optimal interdevice,

intrabuilding, and intrasite interconnection of microcontrollers. Messages that are less than 255 bytes and selections of signaling rates in the 50-500 KBaud range are examples of such optimized features in this standard. Bus characteristic restriction is another feature in the standard that's utilized to simplify network implementation. While the standard is more technologically based than application based, it is still useful for applications in laboratory, measurement, and tests.

For the scope of this project it's very important to consider this standard since it's the basis on how the whole microcontroller development board will communicate with the integrated circuits soldered onto it. While the whole point of using microcontrollers is to simplify the tasks of all the integrated circuits into one "brain"; it's still useful to understand the architecture of how these intrasite interconnections work. It's not needed to understand the foundational logic behind the communication of these circuits, but comprehension may help in communicating with other integrated circuits when the microcontroller requires additional integrated circuits for more functionality.

The first thing to note before explaining the advantages of comprehension of this standard is to research some key concepts that aren't explained in the standard. The most important of such said concepts is the Open Systems Interconnection Model (OSI Model). This model is a seven layer "schematic" that standardizes a communication protocol for computing systems without considering the technical internals of the devices. The first three layers can be looked at conceptually as "media" layers; being in that users don't have much control over the physical technical details associated with it. The first layer of this conceptual model is the physical connections. The physical layer specifies the voltage levels, voltage changes timing, data rates, transmission distance, and device connections. It determines whether the transmission mode is going to be simplex, half duplex, and full duplex. It also determines what type of network topology it will utilize such as busses or meshes. The next layer in this model is the data link. In the data link layer, the responsibilities are split into two sublayers which increases reliability of data transmission. The Medium Access Control (MAC) Layer handles controlling how devices gain access to transmission mediums and with data permission when transmitting. The second sublayer, Logical Link Control (LLC) Layer, is responsible for correcting errors in transmission and frame synchronization. The third and final layer in the media layer is the network. In the network layer transfers variable length data sequences known as packets from one node to another. Typically, this data is transferred in networks with many nodes (or addresses) interconnected within one another.

In the last four OSI Model layers, the user starts having more control over functionality and control. The fourth layer of the model is the transport. In the transport layer data sequence delivery from source to host is of importance. Typically, control over reliability is changed through flow control, segmentation, and error control. Five classes define the transport functionality ranging from TP0

which has the fewest features to TP4 which is less reliable but features more error correcting. The session layer is the next one in the list and includes control over connections with computers. The main thing in this layer is the establishment, management, and termination of connections between devices. It's important because it's responsible for the closing of sessions, session checkpointing, and recovery. The presentation layer is the following layer in the conceptual model. It represents data in a new format that makes it acceptable between applications and network formats. It is typically known as the syntax layer because of data is transformed and compressed into different syntaxes and semantics. One such example includes converting EBCDIC-coded text into an ASCII-coded file. The final layer of the OSI Model is the closest one to the actual programming by the user. Meaning that this layer of the OSI Model falls outside the scope of the standard and can be different depending on the software application at hand. Now that all the layers for open system interconnections are known; the IEEE 1118.1 has improved readability.

Table 15 OSI Model Layers

Layer		Protocol Data Unit (PDU)	Function
Media Layers	1. Physical	Symbol	Transmission of data over a physical medium.
	2. Data Link	Frame	Reliability framework for data frames between two nodes.
	3. Network	Packet	Multi-node managing network for addressing, routing, and traffic control.
Host Layers	4. Transport	Segment, Datagram	Transmission of reliable data between networks using error correcting techniques such as segmentation, acknowledgement, and multiplexing.
	5. Session	Data	Communication management between networks for multiple back and forth sessions.
	6. Presentation		Conversion between networking algorithms to device applications using character encoding, data compression, and encryption.

	7. Application		Final layer available to the end user which uses a high degree of application programming interface (API) to program communication on the software level.

The first key concept to take note of is the generic bus service (GBS) which handles services for the upper-level application. In this optional set of standards, the microcontroller is given a high-level of data and command interchange procedures that are split up into a memory and task model. This in return provides maximum flexibility that supports many architecture types. This flexibility is created by using four octet-addressable memory spaces and another section for input/output spaces. The addressing size is split up into two sections, the page (16-bits) and the offset (8-bits or 16-bits).

Table 16 Memory and I/O Model

	Address Size	
Name	**Page**	**Offset**
Code Space	16-Bits	16-Bits
Data Space	16-Bits	16-Bits
Scratch Pad Space	16-Bits	8-Bits
I/O Space	16-Bits	8-Bits
Special Function Register Space	16-Bits	8-Bits

For the task model, there needs to be an environment where messages can be exchanged between tasks. The upper limit for the number of tasks GBS typically recognizes is 32 within a device with task 0 being the GBS. Using a task number and an optional function ID, master devices can identify which tasks are which and send a command message to tasks on slave devices. The slave devices response by returning a message.

For the GBS model, there are five basic services that the user can control. The first of which is the device control services which allows manipulation of the device such as software resets, slave device offline, request device information, memory permission, and time setting. The second service is memory access and handles the upload/download of data and code as well as the manipulation of memory and registers. The third service is access to input and output addresses of slave devices. The fourth includes access to slave device tasks such as the creation and

deletion of tasks, pausing tasks, obtaining task ID, reading task functions, and communicating task messages. The final service given by the GBS model is user implementation. This includes defining extensions in the command code and call for user-defined procedures.

While an understanding of the physical and software aspects of how devices communicate within one another isn't vitally important for the design of this project; it's still extremely helpful to know the nature behind the communication protocols of these devices since similar protocols are used to communicate between integrated circuits.

4.1.2 (ISO/IEC 9899) Standard for Programming Language in C

The ISO/IEC 9899 is an international standard used to provide specifications on the representation and rules of the C language. The utilization of this standard is only necessary if one microcontroller and microprocessor are used to implement the design. Otherwise, the non-standardized python language will be the only utilized programming language. The standardized areas in C include but are not limited to the representation of the language, semantic rules for interpretation, representation of input/output data processed by C programs, etc.

The C standard is used to define how the C language works. The C standard includes an enormous library of headers and functions that are native to C and don't need to be rewritten. These libraries can simply be imported into the file. There are various libraries for reading input and output files, performing complex mathematical calculations, and modifying strings and other data types. The C standard also includes numerous guidelines on the naming conventions of functions and variables. The C standard also defines what type of variables can be made such as int and char. The C standard also sets guidelines for the C compiler and how the C code gets compiled.

4.1.3 Universal Serial Bus

The Universal Serial Bus, or USB, is a standard that was developed in 1996 by various companies including Microsoft, IBM, Intel, and Norton. The USB standard sets mandatory specifications for connectors and how the connectors interact with the computer and other devices. The USB standard allows peripherals such as keyboards and storage devices to connect to any computer. Without the standard, peripherals made by different manufacturers would have their own cable connector and only work with certain computers. Computer manufacturers would be unable to include every proprietary connector on their computer and be forced to choose what connector to implement.

Since the introduction of the USB standard in 1996, there have been three major generations of the standard. The first generation of the USB standard, referred to as USB 1.0, was released in 1996. The first-generation standard had a data transfer rate of 1.5 Mbits/second to a max speed of 12 Mbits/second. Higher speed rates were used for devices such as hard drives and floppy disks while lower speeds were used for average peripherals such as mice. The second-generation standard of USB was released in the year 2000. The new generation aimed to increase the data transfer rates up to 480 Mbits/second. In 2008, the third generator of the USB standard was released. The third generation of USB devices have a max data transfer rate of 5 Gbits/second and reduce the power consumption required by USB peripherals [8]. Due to the high transfer rate, the third generation USB standard can be used as a video output port replacement for other standards such as HDMI and DisplayPort. The third generation of the USB standard also increased the power output capability. The new generation of the USB standard can also be used as a charging port for mobile phones and laptops due to the increased power delivery output. The capabilities of the new generation USB standard can be seen in the current market as mobile phones and laptops come with a third generation USB port, known as USB C, as a charging port, peripheral connector port, and a video output port all in one.

The USB standard allows peripheral devices to powered through a USB connection which eliminates the need for an individual power source for a peripheral. An important objective of the USB standard was to be fast and efficient. To the naked eye there is no latency between the input of a peripheral, such as a keyboard, and the action of the peripheral happening on a computer screen. USB devices can also be plugged into a computer and removed without having to reboot the computer. USB devices are also able to adjust their own settings by taking advantage of the processing power placed inside USB devices. Overall, the specifications and requirements set by the USB standard allow any user to have confidence that a USB device is guaranteed to work with their computer with minimal effort and configurations.

There are a few limitations to the USB standard. USB peripherals cannot interact with one another. A USB peripheral can only interact with the computer. Also, USB connections are physical, thus they cannot be used to connect devices within a proximity to each other. Finally, as a manufacturer or developer, the use of the USB standard requires the USB device to have hardware and software to monitor and control the data speeds and connection issues. Also, manufacturers must pay annual fees to be allowed to use the USB standard and USB logo for their devices.

4.1.4 Chess Board Layout

The basic 8 x 8 chess board is the standard board layout that has been used since the beginning of chess circa the sixth century and is used in all major chess tournaments. The 8 x 8 layout involves four rows of pieces. At the start of the

game, the top two rows and the bottom two rows are where the chess pieces are placed, and the game begins from there. Over the centuries the game of chess has had many variants. Variants such as hexagonal chess and spherical chess use a different board layout and double chess uses a 16 x 16 board. Other variants of chess use different rules and have different starting positions for each chess piece. In 1924 [5] the World Chess Federation was founded to organize major chess tournaments. The World Chess Federation is responsible for the rules of chess and has made the 8 x 8 chess board layout the standard.

4.1.5 (IEEE 1625) Standard for Rechargeable Lithium Ions

The IEEE Std 1625-2008 describes how rechargeable lithium ion batteries should be designed and what specifications the batteries need to be made to. A manufacturer must label the battery according the maximum voltage that the circuit can reach. The circuit for the battery must also have a tolerance with a four-sigma confidence. The battery must also work under a range of ten degrees Celsius to forty-five degrees Celsius.

The standard also describes the safety precautions that are required for the rechargeable batteries. If the battery reaches temperatures outside of the recommended range, the battery must include the proper circuitry to throttle to the output current to prevent overheating and combustion of the battery. The battery casing must also be made of proper materials to prevent unnecessary damage to the battery.

The host device that the rechargeable battery is connected to is responsible for checking that too much voltage and current is being supplied. The IEEE standard states that the host system is responsible for checking that the battery is not over charged and the host system must include proper discharging circuitry for the battery. The host device is also responsible for keeping the rechargeable batteries within the temperature specifications.

The IEEE standard for rechargeable lithium ion batteries also details specifications and requirements for the charging adapters. The charging adapters should charge the batteries sufficiently while not damaging the safety circuits inside the battery [9].

The rechargeable batteries will be used to power all the components for the smart chess board projects. The batteries must be able to power the Raspberry Pi and all other components required for the project such as the LEDs, reed switches, and multiplexers.

4.1.6 Universal Chess Interface (UCI)

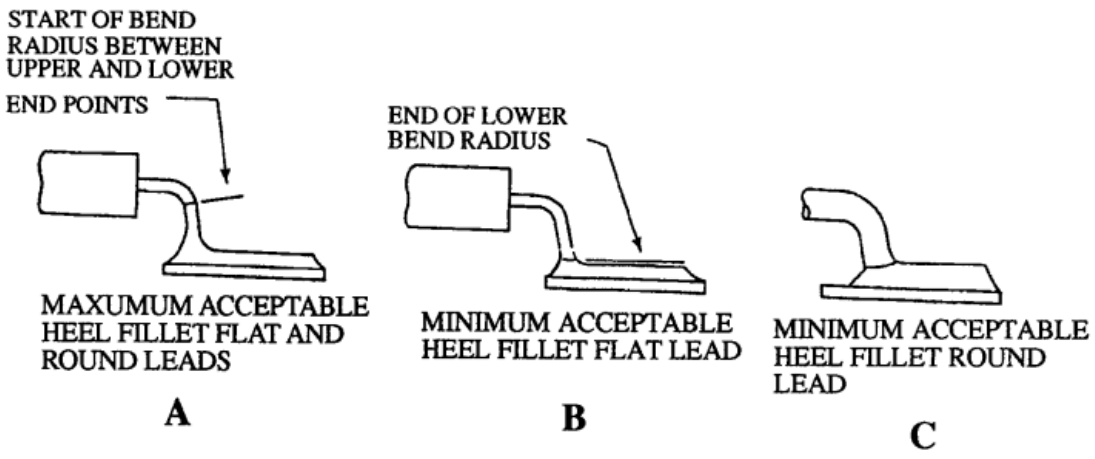
The Universal Chess Interface is a protocol written by Rudolf Huber and Stefan Meyer - Kahlen in November of 2000. UCI is an open communication protocol designed to allow chess engines to communicate with graphical user interfaces. The Universal Chess Interface is an open protocol that can be used by anyone without licensing fees. The UCI protocol was designed to standardize how chess engines interact with users. Thus, any chess engine can be used with any user graphical interface without have to use custom programs and debugging tools.

The Universal Chess Interface sets the guidelines for how the chess engine communicates and what functionalities the chess engine must have. All communication with the chess engine is done through standard input and output text commands. The UCI guidelines list through all the various commands that can be used in a terminal. The guidelines describe all the commands the graphical user interface sends to the chess engine and all commands the chess engine sends to the graphical user interface. Move commands are also standardized so that reading the moves made by the chess engine can be read and understood by any graphical user interface or person. Fail safes are also designed into the guidelines. Chess engines only begin calculating when the “go” command is received, and chess engines ignore unknown commands received by the GUI or terminal [15].

4.1.7 Soldering Standards

A soldering standard has been defined by the National Aeronautics and Space Administration (NASA) under the PDF document name “NASA TECHNICAL STANDARD: SOLDERED ELECTRICAL CONNECTIONS.” For this document, it delves into many of the correct techniques that come about when attempting to properly solder. This document contains techniques that apply to both surface mount parts and to through hole parts which apply to this current design. Some background knowledge and terms must be defined before this document could be correctly understood. To begin with, the term fillet as defined by NASA, is a smooth concave buildup of material between two surfaces. For example, a fillet of solder between a conductor and a solder pad or terminal [17]. This can be seen visually using the diagram provided below portraying a proper soldering method.

Figure 45 Soldering Heel Fillet Application (NASA Permission granted)



To better explain this diagram, this is showing the different cases of minimum and maximum acceptable solder application. In example A, it exhibits the maximum heel fillet flat and round leads allowed before it is considered too much applied solder. Basically, if any more solder is added to example A, the heel fillet will run into the possible issue of no longer working correctly and becoming damaged. In example B, it shows practically the opposite of example A. In this case, it is showing that any less solder than that amount could be a potential problem for the part working the way it should. Lastly, in example C, it shows the same thing example B did, but this time it is showing it for round lead rather than flat lead like in the last example. All these examples pertain to the gull wing package style for surface mount styles which help since most chip parts used in this design all have that style of package.

NASA's soldering standard document also gives pointers and directions for proper soldering by hand and the correct way to handle the components. One of the key things that NASA pinpoints relevant to our design in the PDF is that the solder should be cooled under room temperature when heated and applied. Something such as pressurized air is pinpointed to never be used when trying to let solder cool down after it has been heated. The main reason for this is because if the solder is cooled too quickly, it could cause some unwanted issues to the board. Some of these issues include faulty connections for the fillets and also fragile or brittle solder joints. So, when choosing to apply whatever amount of solder to cover it, then it is important to flow the heated solder around the conductor and then over the pad. This follows from the diagram previously shown above. No motion should be present during application of solder or while the solder is being cooled between the conductors and the pad. Due to the fillets getting possibly ruined or possibly having an obstruction of the joints if they were not set properly.

If the above conditions have been satisfied with properly soldering pieces to the board, then next comes the visual inspection [17]. The purpose of visual inspection is to make sure that the quality of the board and pieces have been maintained throughout the soldering process. This plays into board functionality and making sure that the board will work as expected. There also is a possibility sometimes that visual inspection may not be able to take place. In this situation, alternative methods such as fiberscope optics, X-rays, and other methods are possible solutions to this problem [17]. For example, a situation such as this could arise when there is a part that is present and say, the pads are located underneath the part prior to arrival. In the design for this project however, this shall not be an issue because any piece in our design that will be implemented will be satisfied and be cleared upon meeting the requirements of a typical visual inspection.

Moving forward from passing a basic visual inspection test in order to validate our parts used in the design, other tests can be performed to further validate our design. One important test is the use of the continuity test for the board. This test involves using the digital multimeter probes in order to ensure that the pins on the board or part is correctly connected to all the areas that pin should be connected to through the means of the solder. Most times, pins will not only be in one area. They can be routed through the board, many times through vias that go through multiple board layers and out to the bottom of the board to other devices. This continuity test will verify that the board design is working for pin connections.

4.2 Constraints

When designing any product, there are various constraints that challenge the design and require workaround solutions. Constraints refer to the challenges that arise during the design stage of the product implementation. Specifications for the product may limit the type of parts that can be implemented and require creative solutions to meet the goal. The constraints of the project are listed and described in detail in the following sections.

4.2.1 Design Constraints

There are several constraints to think about while developing and building the smart chess problems. Constraints refer to the design challenges that arise and the design parameters given. One design challenge is determining what material the chess board will be made from. The build quality of the chess board must be durable and have a clean design while not drastically increasing the cost of the final product. The chess board must also use a material that will allow the LEDs to show through. However, the board must allow the LED to light up a single chess square and not allow light to bleed through the board and affect other squares. The

bleed through effect of the light would confuse players, thus not being intuitive to the player. Possible materials being considered are acrylic and glass.

The small spacing between each square on the chess board also presents another design constraint. Each square needs to have a magnetic switch, thus the PCB board needs to be carefully positioned so that each switch fits underneath each square. Also, the size of the chess square determines the size of the chess piece. Each chess piece must have a magnet to activate the magnetic switch and a resistor to identify what piece is on the chess square. The magnets in each chess piece must also not affect the surrounding reed switches. The magnets need to be strong enough to only trigger one reed switch. If a magnet can trigger multiple reed switches, then the action of a piece being lifted will not be detected because there will be other magnets affecting the switch. The circuit design of the switches must also be able to take care of recognizing whenever a chess piece is lifted up and preventing the "ghosting" effect. The sizing of the board and pieces affect the design layout of multiple pieces involved in the design.

Another design constraint is the accuracy of the components. To create an intuitive user experience, the system must operate smoothly. When the user lifts a chess piece, the LEDs pertaining to where the chess piece can move should light up. If they delay in LEDs is too large, the flow of the game will be affected. The detection of when a chess piece is picked up by a user must also be accurate. False detections would cause LEDs to light up unnecessarily which would be unintuitive and confuse the user(s). The reed switches must also be able to quickly detect when a chess piece is moved. In the case that the user immediately knows where they want to move a certain chess piece, the user will pick up the chess piece and place it down within a second. Within this time frame, the microprocessor must be able to detect that the reed switch is closed, highlight the proper LEDs for that piece, and detect when and where the piece has been placed. To accomplish this, there must be no latency between the switch, the LED driver, and the microprocessor.

4.2.2 Economic Constraints

Another constraint for this project is cost. Unfortunately, this project is not funded by any sponsor. Thus, the group members of this project are responsible for buying all the parts needed to build the smart chess board. Due to limited funds, the goal of the project is to keep the cost down while still maintaining an aesthetic design and high build quality. The low cost also limits the features that can be added to the smart chess board. Choosing the right parts for each function is also important because parts that are not viable will still increase the total cost of the product.

Another reason to keep the cost of the final project down is for marketing purposes. If the final project design becomes a marketable product, a low cost would be beneficial for marketing because a low MSRP would attract customers. A smart

chess board product would compete with regular chess boards in the marketplace. If the smart chess board is priced too high, customers will forgo the features and benefits of a smart chess board because it is too expensive compared to a regular chess board.

4.2.3 Time Constraints

Time constraints are the biggest problem of any project because all projects have a deadline. The deadline for the smart chess board project is November of the Fall 2018 semester. The design of the project began in June of 2018 during the Summer semester. Within the six-month timeframe, the final design must be sufficiently benchmarked and meet all required specifications set by the team and the senior design course.

The time constraint greatly affects the features that can be included in the design of the final project. There were various features that were discussed and were in the original design plan. However, certain features such as moveable chess pieces for the computer, would require research and development time that the group decided would not fit within the time frame of the project. If there is time available these features will be reconsidered.

4.2.4 Manufacturing Constraints

There are also numerous manufacturing constraints to take into consideration. The smart chess board is designed to be a portable device that can be played anywhere until the battery life of the device reaches zero. Thus, careful consideration must be taken when choosing the material that the board will be made from. If the chess board is made of flimsy material, the smart chess board will become damaged during transportation or from accidental drops. Accidental drops by the user might cause significant damage and make the chess board unplayable. A strong material that will withstand unexpected environments and still allows the LEDs to shine light through must be chosen.

Another manufacturing constraint is the total size of the smart chess board. The smart chess board is being designed and marketed as a portable device. Thus, the smart chess board must be portable and lightweight so that transporting the device is relatively easy. To achieve a lightweight design, multiple factors must be taken into consideration such as the type of material and the size of the chess board and chess squares on the board. Another consideration is the placement of the chess pieces when the smart chess board is not in use. Due to all the hardware underneath the chess board, the smart chess board will not be foldable. Therefore, there needs to be compartments designed into the board to store the chess pieces.

4.2.5 Safety Constraints

When designing the smart chess board, proper care must be taken to ensure that the user will not get electrocuted. The smart chess board will house resistors inside each chess piece. These resistors will be used to identify which chess piece is being lifted by the user by identifying which resistor value is missing. For this system to work, contact will need to be embedded in the chess board for the resistors to make contact. Thus, the contacts must be designed so that the contact will not be live and prevent accidental electrocution if a user accidently touches the contact.

The batteries must also be taken into consideration when designing the smart chess board. If the batteries are not able to supply enough power to all the hardware components, certain hardware components could be shorted out or become a fire hazard. There must also be a method of determining if the batteries are safe to use and are not prone to swelling. Swollen batteries can also pose a potential fire hazard. The batteries must also not oversupply current to the parts. An oversupply of current and voltage poses a major fire hazard and possible explosions.

4.2.6 Health Constraints

The only health constraint with the smart chess board is associated with the chess pieces. The smart chess board is designed as a board game. Board games are usually played in familial environments. In these environments, there may or may not be children under the age of six. The chess pieces pose a hazardous health constraint to children under the age of six. If a piece is swallowed, there could be serious health consequences such as a ruptured esophagus or stomach. A swallowed chess piece can also damage the intestines and make defecation difficult. Careful attention must be made that young children do not swallow any chess pieces.

4.2.7 Miscellaneous Constraints

After reviewing the purpose and goals set by the group for the smart chess board, it was determined that there are no political, social, or environmental constraints. There are no ethical constraints that are associated with a chess board game. There are also no political constraints associated with the game of chess. Chess is played internationally between people of all nationalities. Finally, there are no environmental constraints associated with smart chess board. The chess board is made from environmentally friendly materials and has no significant impact.

5.0 Project Design and Architecture

This section contains the necessary design documentation, drawings and information needed to fully understand the functions and operation of the project. This section includes the mechanical and electrical hardware designs as well as the software design and integration with the hardware. This section focuses on the technical design elements while explaining some of the reasons for the design and the purpose behind it all.

5.1 Chess Board Housing

The purpose of the housing is to protect the components inside while making the product look aesthetically pleasing. The board housing will be made from wood because it is easily worked with and the group has prior experience with it. This box has an opening in the bottom for batteries, a power port on the side to use instead of batteries, ports on the side for a power switch and mode switch, and an opening on the top for the board itself. The box can be taken apart easily for troubleshooting by unscrewing the top and bottom faces. The game board, which is the top face, is milled from a piece of wood.

The figure below displays the drawing of the top face of the game board. The open spaces in the surface area for acrylic tiles. The surface of the tiles is frosted down with sand paper to hide the components underneath while still allowing light to bass through from the underside. Clear acrylic was the chosen material because of its easy of machining and manipulation. The acrylic tiles will have holes drilled into it for the contacts needed in the piece identification system. Under the board will be a honeycomb system made from strips of interlocking wood to block LED light from traveling between adjacent tiles. The box will be big enough to hold the PCBs, wires and all other components.

Each of the tiles on the chessboard will be 1.4 inches square. Frosted enough for a suitable opaqueness/light distribution and machined for contacts on the surface. The figure below displays the tile drawing. Each square is measured to obtain a realistic idea of what measurements need to be taken when machining the chess board. The dimensions of the length, width, and height are displayed in the figures below and give a detailed illustration of a prototype chess board.

Figure 46 Drawing for the Top Face of the Chess Board

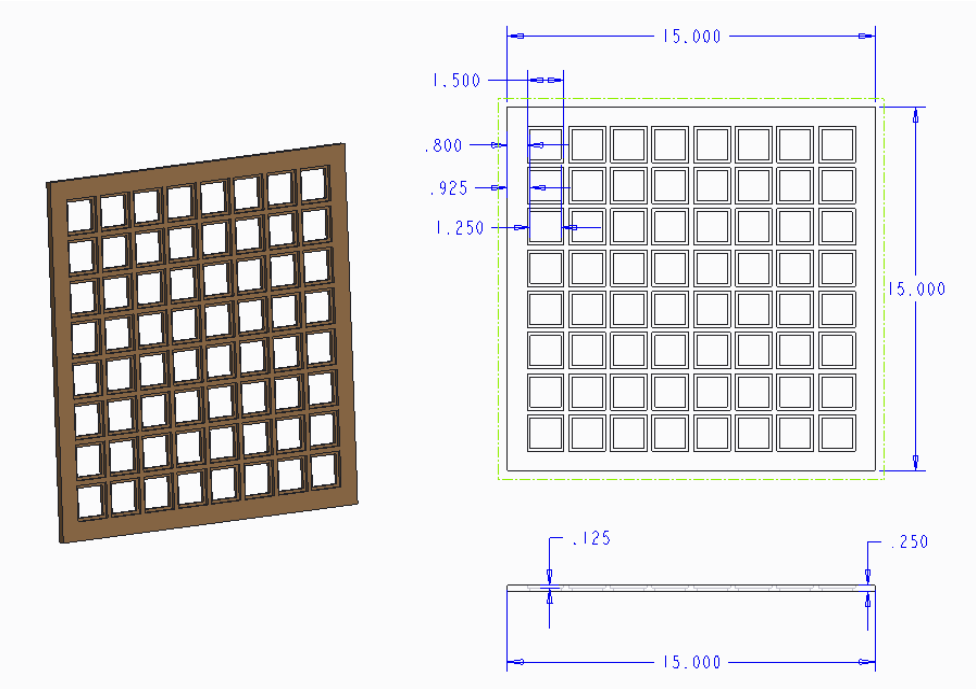
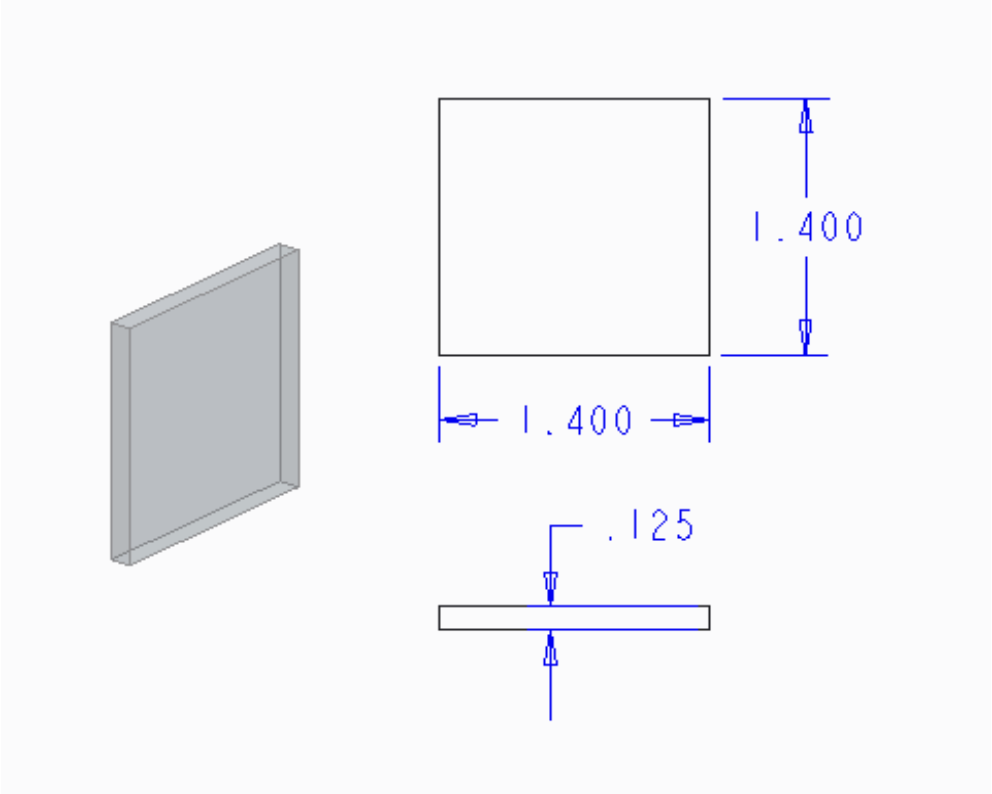


Figure 47 Acrylic Tiles Used for each of the Chess Tiles

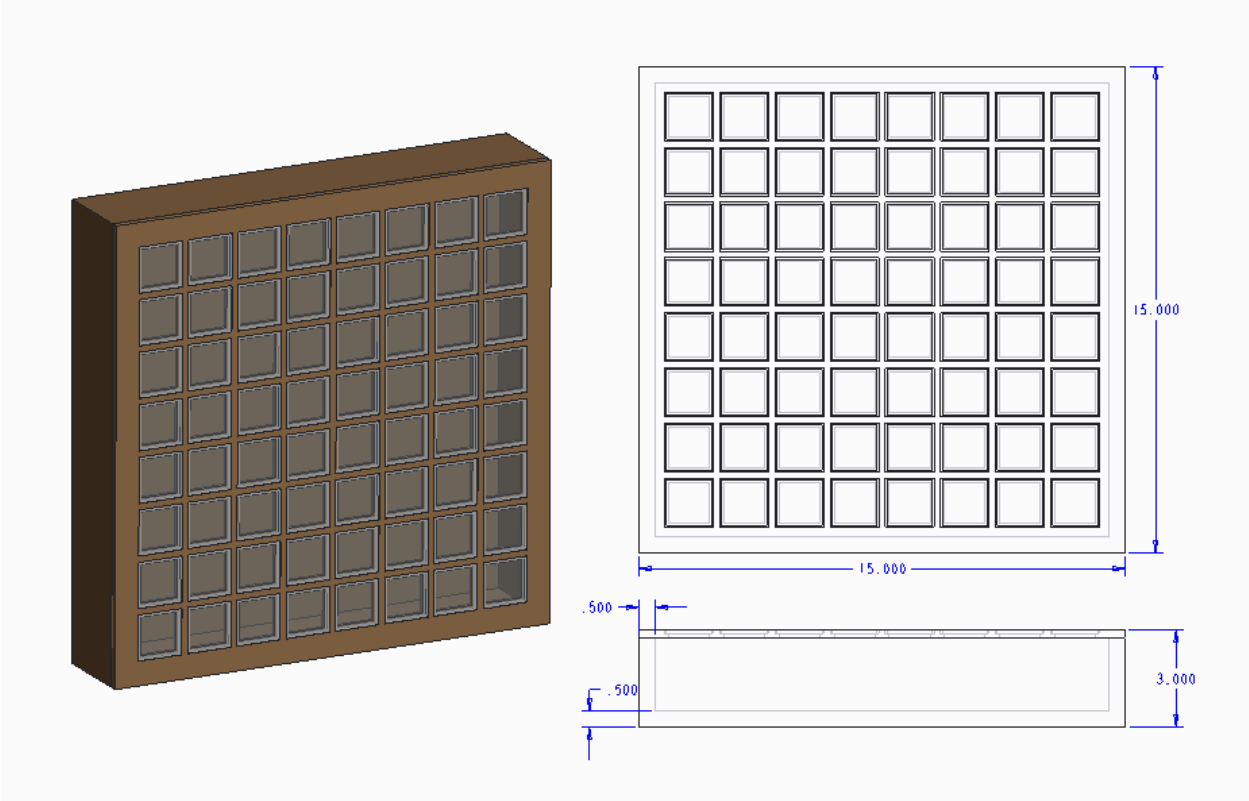


The box holding all the materials will be 15 inches by 15 inches square. The depth of the box will be 3 inches. During the fabrication of this chess board, the size may need to increase based on the final size of the components and PCB. The Table below shows a summary of the most used dimensions on the chess board. The figure below displays the final assembly drawing of the project housing.

Table 17 Chess Board Housing Dimensions

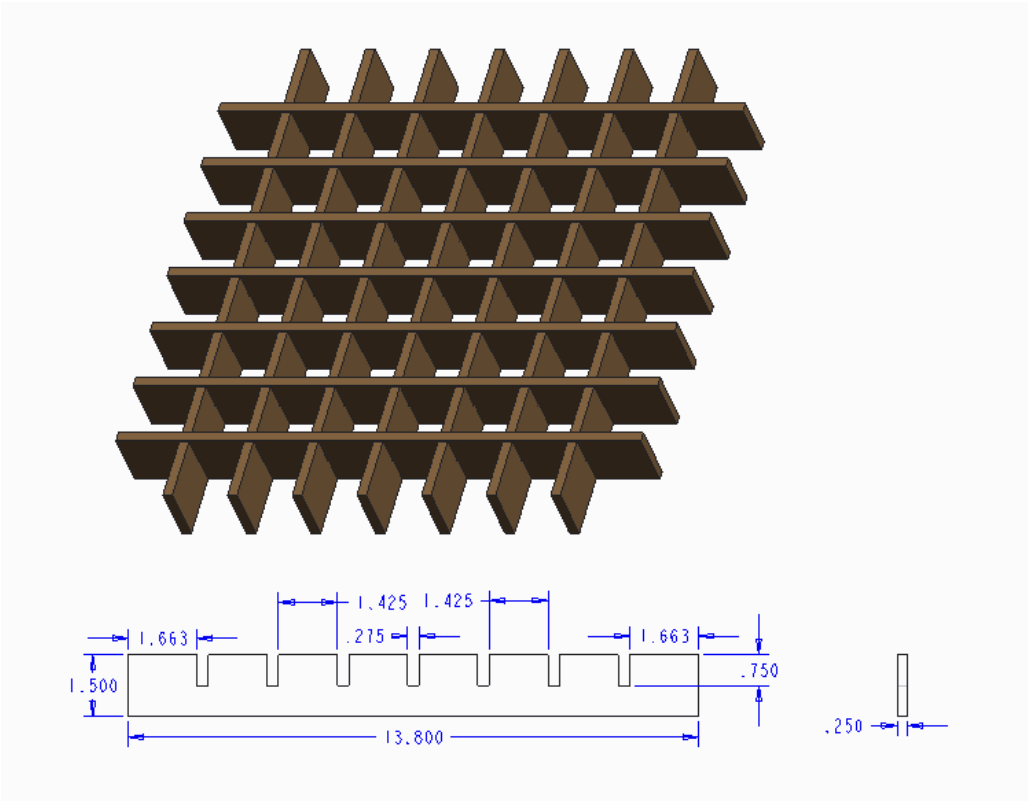
Component	Dimension
Chess Board Tile	1.4” by 1.4”
Chess Board Surface	18” by 18”
Surface Tile Indentation	1.5” by 1.5”
Chess Box Length and Width	18” by 18”
Chess Box Depth	4”

Figure 48 Final Assembly Drawing for the Project Housing



The honeycomb structure used underneath the surface of the board is shown in the figure below. The honeycomb structure will be permanently installed to the underside of board. This structure will be the mounting points for the larger PCB design holding the LEDs, Reed switches and other major components that directly interface with each tile.

Figure 49 Drawing of Honeycomb Structure



5.2 User Interfaces

The user interface in its basics are the features and functions that the user directly interacts with. In this design of the interactive chess board there are two main user interfaces, LEDs for display and Switches/Buttons in the tiles for function. Each of these interfaces are the backbone to the project. They need to be easy to use and understand as well as robust, so they can be reliable. This project has a very simple interface with only two modes, training mode and emulation mode. Both modes utilize similar interfaces, but they differ slightly. This section will break down the user interfaces into the type, function and mode.

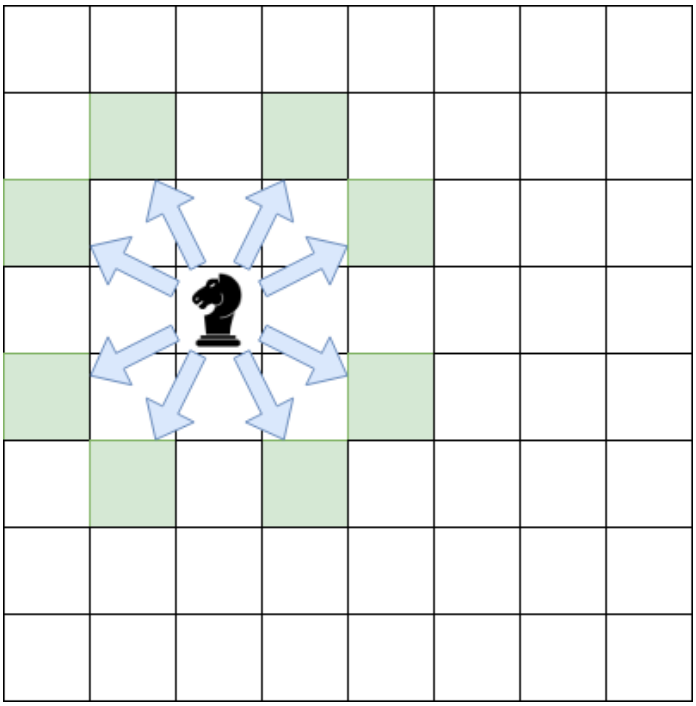
5.2.1 LED Interface

The LED user interface is one of the core interfaces for this interactive chess board. This is the only way for the game, and extension the microcontrollers, to communicate with the user is via the LED display matrix. The goal of this interface is to provide a clear and easy medium of communication to the user via multi-colored visual cues and patterns.

5.2.1.1 Training Mode

Training mode will utilize the LED interface in a fairly simple way. Training mode allows the user to set the game pieces in any configuration they would like. This means they can set up the game in the original starting position like a normal game of chess would go. They can also set up different chess scenarios for training and practicing purposes. Training mode offers the most flexibility. No matter what the user intends to do in this mode the LED interface will react the same. Upon lifting a chess piece from a tile, the action will activate the switch corresponding to that tile, the microcontroller will light up all the tiles that the piece can move to or it will light up all the pathway that the piece can move along. Depending on the piece selected. This interface in its basics will teach the user what positions the piece can move to in a game a chess. The figure below displays a knight and the possible movements it can make during a very simple training mode scenario.

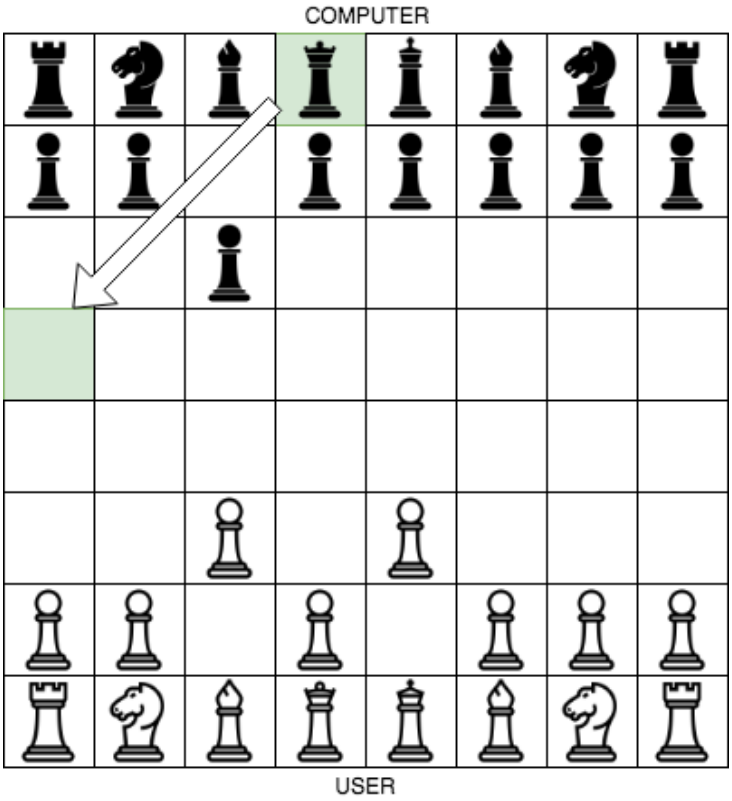
Figure 50 Training Mode Scenario Displaying a Knight and the Possible Movement Options it Can Make.



5.2.1.2 Emulation Mode

Emulation mode utilizes the same functionality that training mode utilizes and more. This mode will react in the same way that training mode reacts in that if you select a piece on a tile, the possible movements of that piece will light up on the board. The difference, whereas in training mode you can select any pieces from either of the two teams, in emulation mode you can only select the team the user is playing. The computer controls the other team itself and will not show you the possible paths. Instead the computer will light up the tile of the piece it wants to move and light up the tile of the place it wants the user to move it to. The user will have to move the piece manually, but the LED interface gives the computer an easy to understand tool for communicating to the user. The figure below displays the beginning of a possible emulation game scenario; the computer is communicating with the user to move the bishop from the starting square to a new square.

Figure 51 Emulation Scenario Displaying the Computer Controlling the LED Interface to Communicate to the User Where it Wants to Move the Next Piece



5.2.2 Physical User Interfaces

The other core interface that is needed for the user to be fully engaged in an interactive chess game is a physical user interface to determine the course of the game by the user. In this project there are four physical interfaces that the user can change and operate to their liking. There are reed switches in every tile of the board. These reed switches are toggled under a magnetic field. The magnetic field will come from magnets installed in each of the 32 chess pieces. These reed switches will also provide the location assistance needed for the piece identification subsystem. The last physical interface the user has control over is the mode select switches. These switches have two positions and gives the user control of whether they want to play chess or checkers, and whether they play that game mode in PVP or PVC mode.

5.2.2.1 Chess Player Vs Player Mode (PVP)

With the mode select switch in the PVP mode position, the computer will run the PVP mode software. The training mode software keeps a table of the possible movements that a piece can take based on the rules of the game. This mode utilizes the reed switches located in under each tile in the board and the magnets located in the bottom of each chess piece. The chess pieces can be installed in any configuration on the board. When a piece is lifted from the board the reed switch under that tile deactivates and triggers a function within the microcontroller. The microcontroller can determine what tile was selected via a switching matrix. The user can only select one piece at a time, but they can select any piece from any team at any given time. When the piece is selected, within an instant the LED interface displays the possible movements. If two pieces are selected at the same time, then a significant error will occur, and the microcontroller will display nothing on the LED interface. If the player makes and illegal move, then the tile will light up red and will require the user to fix the mistake.

5.2.2.2 Chess Player Vs Computer Mode (PVC)

With the mode select switch in the PVC mode, the computer will run the PVC software. This mode makes use of the chess engine. The PVC software works by keeping track of the positions of each of the pieces because it knows the starting position of each piece and it can detect each of the movements that the user makes. Since this mode needs the starting position there is no flexibility for the user to start the game any way they like or in any scenario they like. The starting position must be the initial starting position that define the rules of chess. Emulation mode makes use of the reed switch physical interface located in all 64 of the tiles on the chess board. Every time the user moves a piece, whether that be for themselves or on behalf of the computer, the computer knows that an event has taken place because the reed switch will toggle off when the piece lifts off from the current tile and then toggles on the switch on the new tile. This user physical

interface is the most interactive but also the most restricted. The user will be interactive with this interface every time a piece is lifted off a tile and placed onto a tile, but the game restricts the user to taking turns and will recognize if the user performs an illegal move, unlike in the PVP mode. The same rules will apply as in PVP mode and the tiles will light up red in mistakes and green to follow a path.

5.2.2.3 Checkers Player Vs Player Mode (PVP)

In this game mode all functions except for the LED display will not be used. The illuminated chess tiles will be illuminated and be a random color switching once a second. This provides a rainbow-like effect on the chessboard and the pattern allows checkers to be played. We call this game mode rainbow checkers.

5.2.2.3 Checkers Player Vs Computer Mode

This mode does not exist, instead we use this switch configuration as a development diagnostic tool. When this mode is engaged, only the tiles with chess pieces sitting on top of them will light up green. Everything else will be switched off. This mode is very helpful when troubleshooting issues with the game modes because we can see whether the issue is a switching issue or another issue.

5.3 Piece Identification Subsystem

Piece identification is one of the more challenging subsystems in this project. This technology in a chess board is very uncommon. Most “smart chess” boards have a locator within each tile and can determine if a tile is occupied or empty. This works because a computer-based emulator keeps track of the initial starting position of each piece and can keep track of where each piece has been moved to because only one piece moves at a time. This is a technology that we are implementing in another subsystem, but this is not the basis for the piece identification subsystem. This subsystem will allow each piece to be unique. It will allow any configuration to be placed initially on the board, so a training scenario can be taught. The research required for the system was extensive due to the complexity of functions it needs to perform. The research is highlighted in section 3.4 where a more detailed explanation of the system is located along with a comparison of other systems not used for this design. The goal of this subsystem is to bring flexibility to the training mode of this chess board.

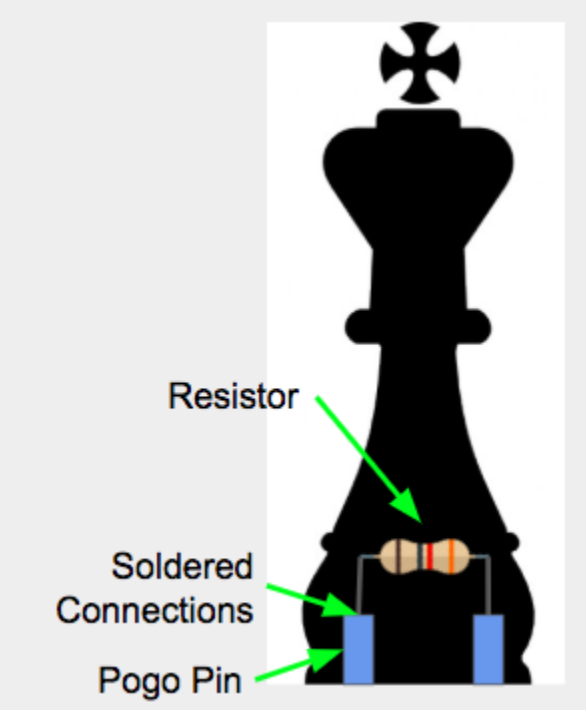
While in training mode, a player may select any piece on the board by lifting up a piece from the board which deactivates a reed switch under the board. The switch will send a signal to the microcontroller and the ADC will sample the sum of the remaining resistors on the board. The microcontroller will compare that sample with a sample it took before the event. It can determine the value of that resistor

based on the difference. The microcontroller can then determine which tile the piece was lifted off based from the reed switch matrix. The microcontroller know knows the location and type of piece that was on the tile before the event. It will light up the LEDs in the paths or tiles that the piece can move to. These functions are the basics for the piece identification subsystem.

5.3.1 Unique Resistors in the Chess Pieces

There are 16 chess pieces on each team in one standard game of chess. There are 6 unique pieces on one single team, 8 Pawns, 2 Knights, 2 Bishops, 2 Rooks, a King and a Queen for a total of 16 pieces. So, in total there can be up to 12 unique pieces on a chess board at any given time. Each unique piece will have a different value resistor installed in it with terminals on the underside of the piece and on top of the board. This will allow a signal to flow through the resistor when properly installed on the board on any of the 64 tiles. The contacts we will use on the underside of the pieces will be pogo pins. They will contact copper contacts embedded on the surface of the tile. The magnets in the pieces will serve a dual function. Firstly, they will be magnetic so that there is a firm holding force onto the board. The magnets will also toggle a magnetic reed switch underneath the tiles in the board, these switches are also for tracking pieces in emulation mode.

Figure 52: Cross section of chess piece



The 12 values chosen for the resistors need to be large, so we can neglect contact resistance and other small resistance factors. The key point is to have a consistent resistance for one piece across every tile on the board. The resistance values also need to be chosen keeping in mind that there needs to be a semi-significant change in the summation of all the resistors in a parallel configuration. Every pair of contacts in every tile on the chess board will be connected in parallel. So, the sampling can be taken during any situation. The values chosen are highlighted in table 5.3.1 below.

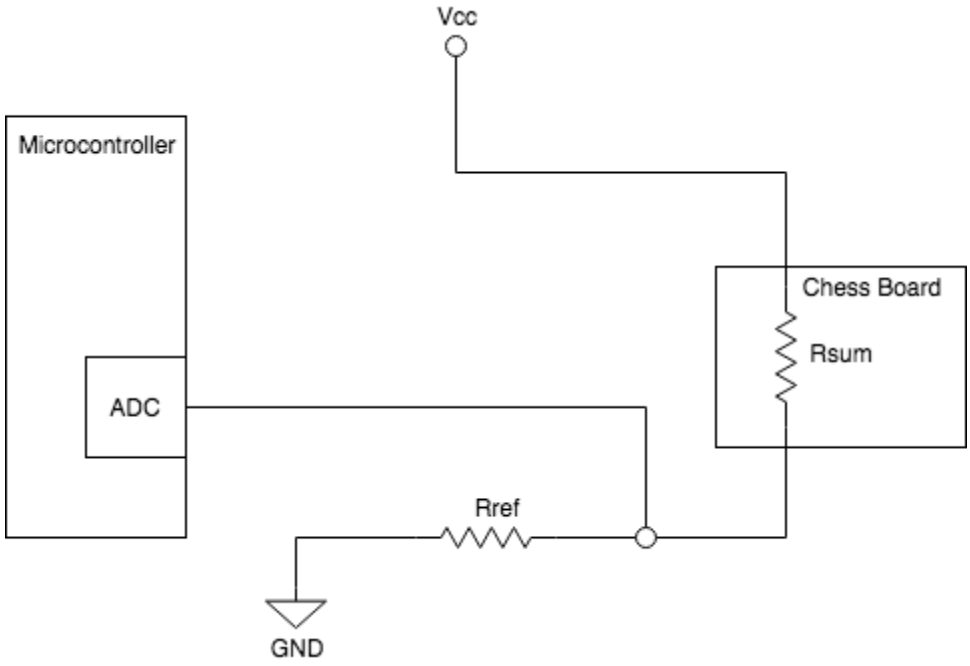
Table 18 Piece Identification Resistance Values

Chess Piece	Number of Pieces	Resistance value chosen
White King	1	200 Ω
White Queen	1	220 Ω
White Bishop	2	243 Ω
White Knight	2	274 Ω
White Rook	2	324 Ω
White Pawn	8	402 Ω
Black King	1	499 Ω
Black Queen	1	604 Ω
Black Bishop	2	750 Ω
Black Knight	2	1000 Ω
Black Rook	2	1500 Ω
Black Pawn	8	3010 Ω
Total	32	20 Ω

A fixed reference resistor pulled down to ground will be where the ADC samples the voltage. The reference resistor will be chosen at the median of summation of all the chess piece resistances in parallel. This is because at the median the changes in resistance when we remove a piece from the board will be at its greatest. This gives the ADC the best chance to be able to compare the sample before and after an event. The reference resistor value will be about 50KΩ. The figure below shows a basic block diagram of the piece identification circuit. Rsum

before an event of lifting a piece off the board will be the parallel summation of all the chess piece resistors on the board. After the event the sample will be the same value minus that of the resistor removed during the event.

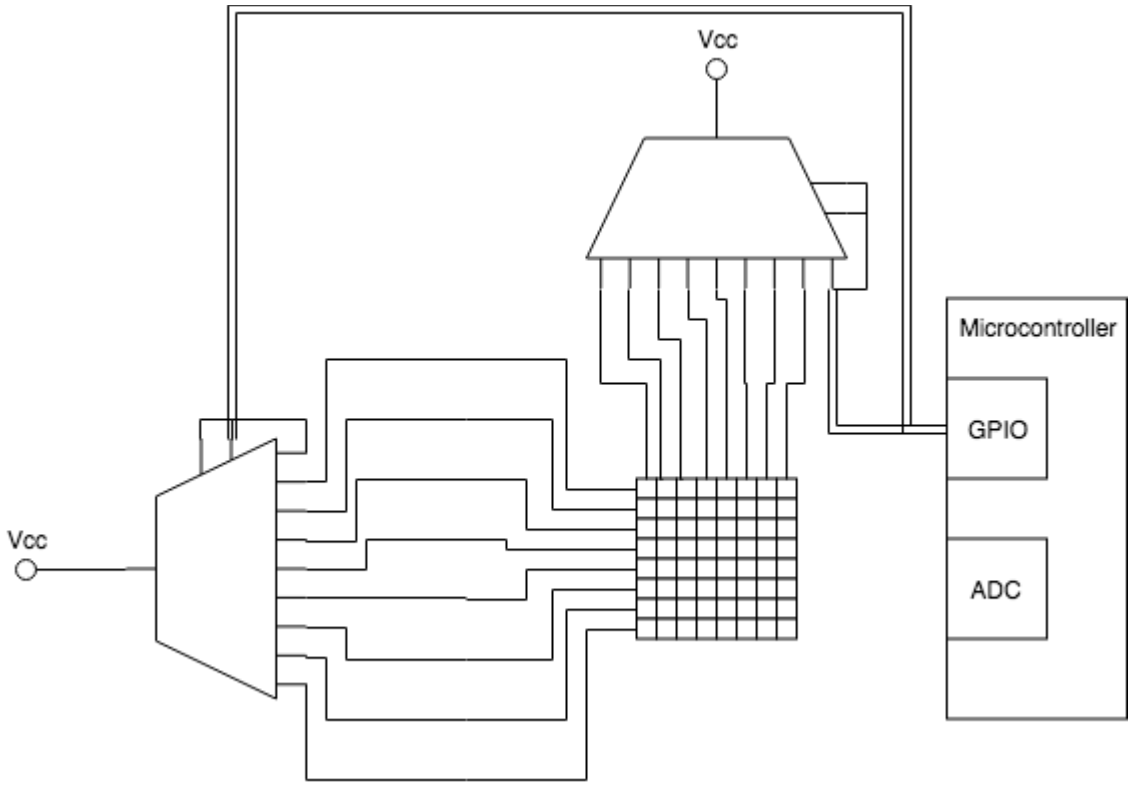
Figure 52 Basic Block Diagram of Piece Identification (Simplified for Clarity)



5.3.2 Board Array Design

This section covers the design for the piece identification subsystem across all 64 tiles on a chessboard. The logic behind this design stems from the fact that only one piece will ever be selected at one time, per the game rules and design simplicity. There will be a selection matrix and a sampling matrix. The selection matrix will be an array of reed switches that will allow one and only one switch to be selected at a given time. This design will consist of 64 switches and two 8 channel demultiplexers. One for the rows and one for the columns. When a switch is pressed each multiplexer can determine which row and which column was triggered. This can be narrowed down to a single tile. The block diagram is shown in the figure below.

Figure 53 Block Diagram of Selection Matrix

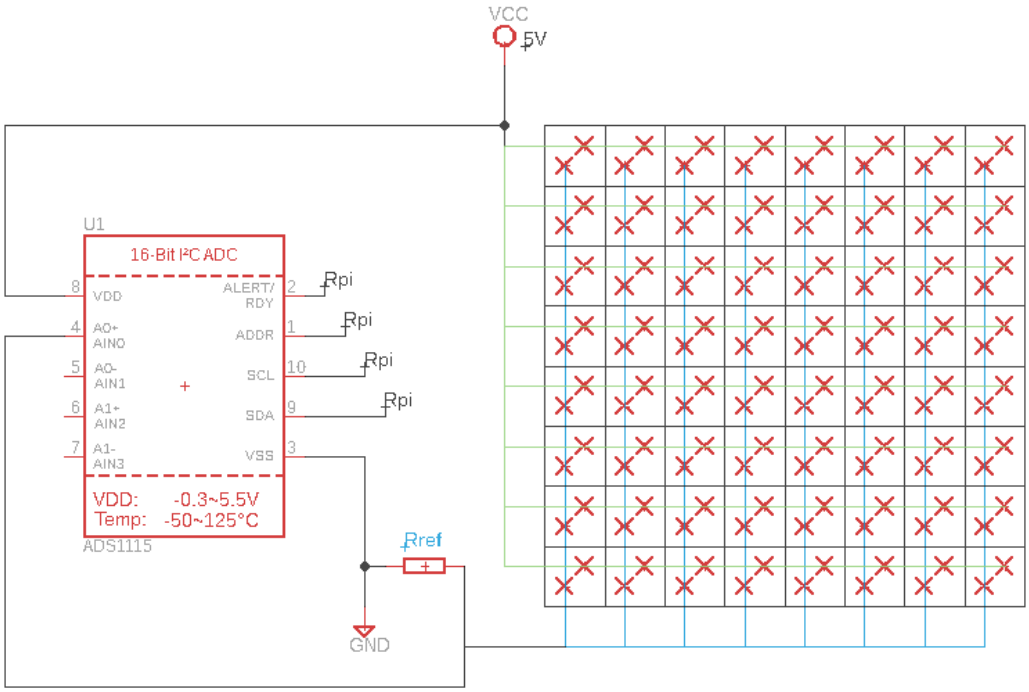


The Sample Matrix's purpose is to allow combine the contacts for every tile on the board so one ADC can sample them all in one run. The microcontroller can determine which tile was selected and compare the samples before and after the event to determine the change in value. The sample matrix is a very simple concept where all the terminals under every tile are connected in parallel between a voltage source and a reference resistor.

5.3.3 Design Summary and Schematics

The piece identification design in its basic form is a simple resistance test circuit, demultiplexers select the tiles and the ADC does the testing. The value the ADC sample is directly based off the sum of the resistors in parallel. Sampling is done before and after the event and compared. The circuit shown in the figure below is the closed-circuit block diagram for the entire system.

Figure 54 Piece Identifier Electrical Schematic Block Diagram



5.4 Power Management and Distribution

Calculations are required to properly diagnosis the exact amount of battery capacity needed to power the system to an acceptable standard. Since the chessboard should be as convenient as possible for consumer interest; the chessboard should house a rechargeable battery with enough capacity to maintain several games. Since the design scope will house several integrated circuits with their own power consumption and current draw it will be important to also consider power protection devices to prevent anything from overheating and corrupting. Most of the power consumption should occur at the LED array since lightening these components will need to be distributed to 64 individual RGB LEDs.

5.4.1 Rechargeable Lithium Batteries

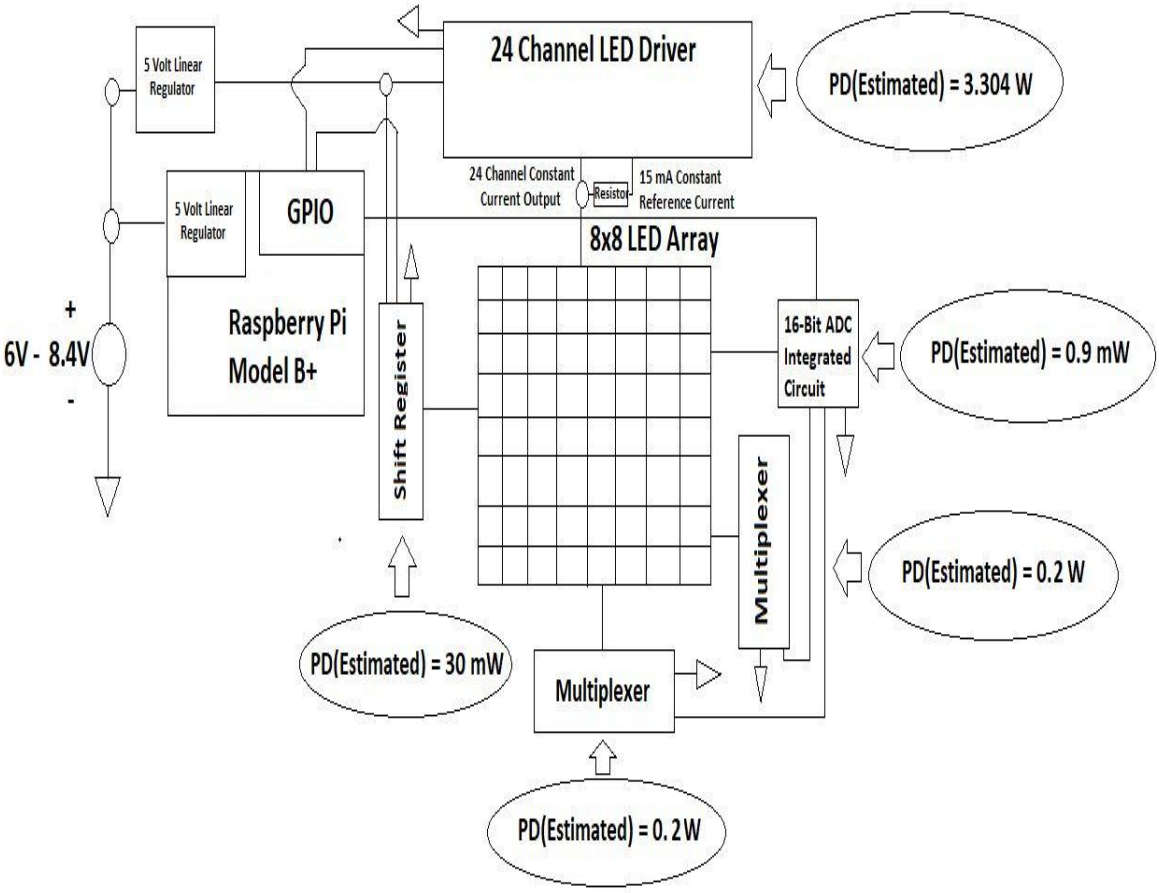
Most of the testing for the design shall be done with a USB connected power supply but to make the product more marketable the chessboard will need to be portable. The perfect power supply for this is a pack of lithium batteries with enough capacity and voltage to power the product. Since a voltage regulator will be used to maintain a steady voltage input of 5 volts; the batteries can be hooked up in series and in

parallel in a variety of ways if it doesn't pass the 35-voltage input limit. Ideally, the battery supply should be enough to power all the devices for at least five games but with only 12000 mAH of capacity available in the batteries some calculations should be considered first.

5.4.2 Power Consumption Schematic

After a thorough examination of the power consumption of each integrated circuit that will be controlled by the Raspberry Pi; it was seen that the LED driver had highest power consumption compared to the other devices. This is most likely since it's powering 64 RGB LEDs which will have a high-power draw. Since the power draw for the LED driver is so high compared to the other integrated circuits, they can safely be considered negligible. This is since the other devices are "low-powered" circuits designed to not have a heavy burden on the system. Since the power consumption for the circuits has a heavy emphasis on the LED Driver, an in-depth analysis of the power dissipation of the devices should be considered.

Figure 55 Power Consumption Block Diagram



5.4.2.1 LED Driver Power Dissipation

For the aim of accurately calculating how much battery storage will be needed for a consumer-friendly run time; the datasheet will be thoroughly examined to see exactly what variables will be significant in the power dissipation calculation. After much inspection, the main power consumption occurred at the LED level. This was expected, and it was important to considered factors such as LED count, output current, duty cycle, and LED voltage output. Power savings could be achieved with optimizing the constant current output and the duty cycle. The current could be adjusted depending on the external resistor value while the duty cycle will be dependent on the software. Current values can be easily chosen using the table below. Once all the variables are known, the power dissipation equation from the datasheet can be used to calculate total wattage.

Table 19 Constant Current vs Reference Resistor

Constant Output Current	Reference Resistor
30 mA	1640 Ω
25 mA	1968 Ω
20 mA	2460 Ω
15 mA	3280 Ω
10 mA	4920 Ω
5 mA	9840 Ω
2 mA	24600 Ω

Figure 56 Power Dissipation Equation

$$P_D = (V_{CC} \times I_{CC}) + (V_{OUT} \times I_{OLC} \times N \times d_{PWM})$$

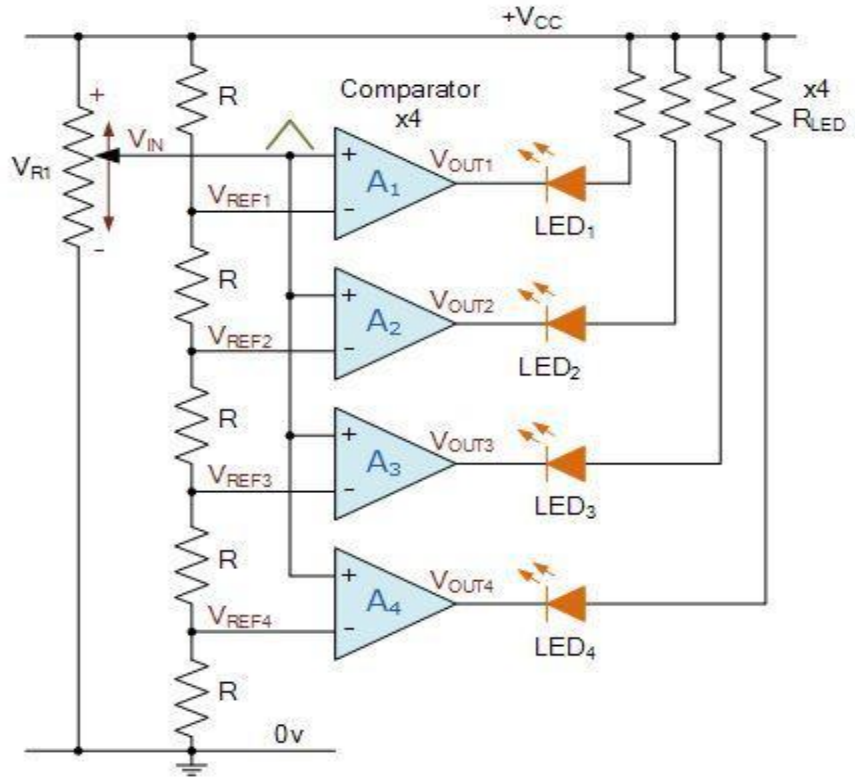
where

- V_{CC} = device supply voltage
- I_{CC} = device supply current
- V_{OUT} = OUTn voltage when driving LED current
- I_{OLC} = LED current adjusted by R_{IREF} resistor
- N = number of OUTn driving LED at the same time
- d_{PWM} = duty ratio defined by GS value

5.4.3 Power Protection Design

Lithium batteries are very effective batteries but require safety features since they are dangerous given certain electrical conditions. Such conditions include allowing the voltage to dissipate below its safety voltage and charging it above its maximum safety voltage. There's also a question of too much current draw from the battery. Since these conditions can be potentially dangerous, most rechargeable lithium batteries come with safety circuits built in them to handle such conditions. While there are protective circuits in place for safety, these conditions should be accounted for since they can damage the battery and make it unusable. One potential idea to avoid discharging the batteries too much includes a circuit schematic which turns on a red LED if the batteries reaches six volts together (meaning three volts per battery). This can be done by using a comparator and hooking up a red LED to indicate that the voltage level has reached closed to the minimum voltage allowed before the protective circuit is toggled. To make sure that the circuit isn't overcharged, a similar circuit to the previous one can be built but instead the comparator is used to make sure that the battery isn't overcharged past eight volts (meaning 4 volts per battery) and uses a green LED to indicate those conditions.

Figure 57 Voltage Level Detector Circuit Example



Another important component that will require a safety feature circuit is the LED driver. Since LED drivers require a constant current, the voltages may sometimes spike causing a temperature increase which could cause a thermal runaway. If this condition is reached the LEDs will fail. To avoid these circumstances, fuses may be utilized to protect against overcurrent. These fuses should be selected according to voltage rating, current rating, temperature de-rating, and interrupting rating. However, since the scope of the design won't require a heavy load on the LED array it's not necessary to add protective features to the LED driver.

5.5 LED Matrix Subsystem

This section highlights the design of LED Matrix. The LED matrix as described in prior sections above, is an array of RGB LEDs that sit under every tile on the chess board and provide an illuminated output that the user can understand. This is one of the only communication methods that the Smart Chess Board must communicate with the user. The LED is one of the most complicated systems within this design and it's the most visible as well. Because of this, the system needs to work with a much higher percent of success than the other systems because it is seen by the user visibly.

The basic function of this subsystem is to keep it as inexpensive as possible. We will be configuring our 8 by 8 RGB display with 3 shift registers and 8 GPIO pins switched on by 8 BJT's. The shift registers will provide the color and brightness data and the BJT's will provide the refresh rate. The theory behind this design is that we will connect the rows and columns together and switch the rows one at a time. Rows will be connected via the LED common anode. The rows will toggle source to the LEDs and the column will pull the LEDs to ground. As we switch through the rows each one lights up a specific pattern then changes to the next. We switch through the rows so fast that the human eye cannot see the change in switching. It will look light all the rows are lit at the same time. The time the rows switch through a whole cycle is the refresh rate. The refresh rate will be set by the microcontroller, but it will need to be high enough to

5.5.1 LED Controller

The LED controller we chose for this project is the TPIC6B595. This shift register has 8 channels individually and is daisy chained together with the other shift registers to get 24 channels altogether. We are using RGB LEDs which are basically three separate LEDs put into one package. So, with 24 channels we can support 8 RGB LEDs. There are 8 tiles in a column on a chess board, so every 3 channels will correspond to a red, green, and blue LED under that tile. The shift register will be controlling the columns so when daisy chained, it will serve the purpose to control the 24 different channels.

5.5.2 LED Source

Since the LED controller is the shift registers that are daisy chained together, we have enough GPIO pins on the microcontroller to switch through the rows one at a time for the refresh rate of 60 Hz. We decided on using regular 2N2222 BJT's to serve our purpose. The outputs of the microcontroller will connect to the base of a transistor. This design lets us utilize the ease of programming while giving us a high current source.

5.5.2.1 Power

The transistors collector will connect to a five-volt source and the emitter to the rows of the LED matrix. When the shift registers outputs a high value then the transistor will allow current to travel from the source to the LEDs. The transistor must be able to supply current to all the LEDs in that row at the same time. There are 8 RGB LEDs so there are 24 LEDs total. If each run at about 20 mA, then we would need a transistor that can source about 500mA at least. The 2N2222 bipolar junction transistor (BJT) is the transistor of choice because it can supply 800mA and still provide a fast-enough switching speed.

5.5.2.2 Refresh Rate

The LED matrix refresh rate is crucial for providing a clean visible output. The refresh rate will be created via the shift register connected to the rows. Once row will turn on at a time sequentially e.g. the first row will switch on and then the second will switch on at the same time the first switches off and the onto the third. The seamless transition from row to row while changing the LED controllers' colors at the same time provides challenges. A lot needs to happen in a short amount of time. This system needs very fast switching components to keep up with demands. This hardware will require precise programming as well in very little time. There will need to be tradeoffs for which method is the best and fastest for programming the driver and source.

5.5.3 Schematic

The schematic for the subsystem is simple. The design itself will be very time consuming to construct and prototype, but the team will implement time saving practices. The design schematic is shown in the figure below. The schematic highlights the way the LEDs are connected and how they will be attached to the LED driver and other components around them.

Figure 58 LED Controller and Shift Register

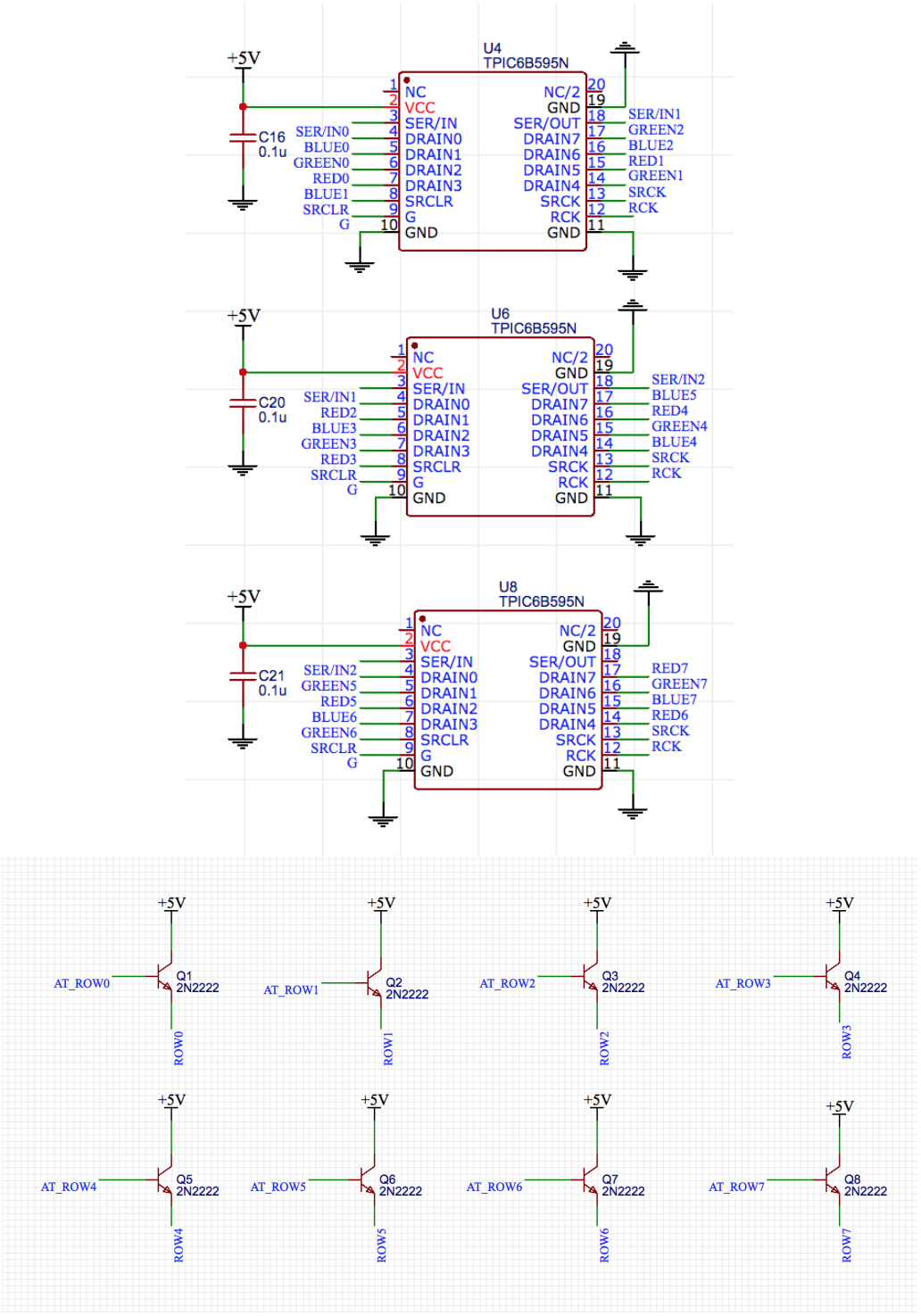


Figure 59 LED Matrix

5.6 Detection Matrix Subsystem

This section describes the research and design of the detection matrix used by the smart chess board to determine when the user lifts a chess piece off the board.

5.6.1 Chess Piece Detection

When using switches for detection, there are two major problems that can occur during the detection, ghosting and masking. Ghosting refers to when a switch seems to have been pressed when the switch was never pressed. Masking refers to when a switch is not being detected. For example, the detection of a switch being unpressed might go unnoticed, causing the switch to stay pressed for an unknown amount of time [16].

5.6.1.1 Ghosting

When multiple switches are active at once, detecting which switches are active and which switches are open can become tricky. For the smart chess board, an 8 x 8 matrix of switch will be needed. Each square on the chess board needs a switch to detect when a chess piece is placed on that square. During the game, there are numerous scenarios where there are multiple chess pieces placed on the same row or column. Thus, there are some switches that are open and some that are closed. In an 8 x 8 matrix, this creates many routes for the current to travel through when trying to accurately detect which switches are closed. This is known as the ghosting effect. The ghosting issue would affect the smart chess board by detecting that a chess piece is on a square and close the switch, when in reality the chess piece is not on that square and the switch should remain open.

5.6.1.2 Masking

In the same scenario with multiple chess pieces on the same row or column, when a chess piece is lifted, the microcontroller may not be able to detect the action. This is known as the masking effect. Depending on the board configuration of the chess pieces, when a chess piece is lifted, the current may be able to travel in a way that does not allow for the detection of a chess piece being lifted and the switch underneath opening.

5.6.2 Diodes

The easiest solution to solving both the problem of ghosting and masking is to add diodes in series with the switches. With the diodes in series, the diode becomes reversed biased when a current flow through it. When the diode is reverse biased, the diode prevents the current from traveling backwards thus there is only one path for the current to travel and each switch can be detected [16]. However, adding a diode in series with the switch adds a delay in how long it will take for the microprocessor to sample the matrix to determine which switches are closed. This delay will be multiplied by sixty-four because each square has a switch and each switch needs a diode. Therefore, a diode with a fast switching time is necessary.

There are various diodes that can be used in series with switches. While there are several diode types that are available, PN junction diode, a Schottky diode, and the PIN diode are the based diode types for the application.

The PN junction diode has three operating modes, forward, reverse, and zero bias. If the PN junction diode is in zero bias, the diode is said to be in a state of equilibrium because there are an equal number of electrons and holes. With an equal number of electron and holes, there is no current traveling through the diode. In a reverse bias configuration, the depletion region increases as electrons and holes distance themselves from each other. The increase in the depletion region stops current from traveling through the diode. In a forward bias configuration, the depletion region decreases as electrons and holes move closer together. If the depletion region becomes too small, the diode will act as short and the effects of ghosting and masking will still be prevalent. Due to the ability to stop current flow, PN junction diodes are primarily used in rectifier applications such as converting a sinusoidal input into a full wave output so that the negative power can also be used [18].

The Schottky diode has many desired and improved characteristics over a PN junction diode. Schottky diodes are smaller, more efficient, and have a smaller voltage requirement over a PN junction diode. These characteristics are due to the design of the Schottky diode. The Schottky diode is a n-type only device, thus it only allows current to flow in one direction. While Schottky diodes can also be used in rectifier and high frequency applications, the Schottky diode can also be used in

switching applications due to the current only being able to travel in one direction and the small voltage drop. However, Schottky diodes are generally more expensive than other types of diodes [18].

The PIN diode is like the PN junction diode except a PIN diode has an intrinsic layer between the P and N type layers. The electrons and holes can move much faster between the layers which gives the PIN diode its fast switching characteristics. The PIN diode can perform all the same functions as other diodes. However, its fast switching ability makes it a prime candidate for small signal diode switching applications. Due to how fast a player can move a chess piece if the player knows where they want to move, a PIN diode would be beneficial because there would be very little delay for the microcontroller to read the switch opening and closing [18].

After researching the different types of diodes that can be used to solve the ghosting and masking problems associated with switch matrices, a PIN diode will be used in series with the switches. The PIN diode was chosen due to its fast switching time. This minimizes the delay between the sampling times when the microcontroller is checking to see which switch has opened from a lifted chess piece. With a reduced delay, the microcontroller will be able to send a signal to the LED driver to light up the required LEDs faster than if another type of diode was used.

5.7 Possible Features to Incorporate

Due to time and design constraints of the project, there are many features that the group wanted to implement but were unable too. If there is enough time, these features will be reconsidered. The possible features are detailed in the following sections.

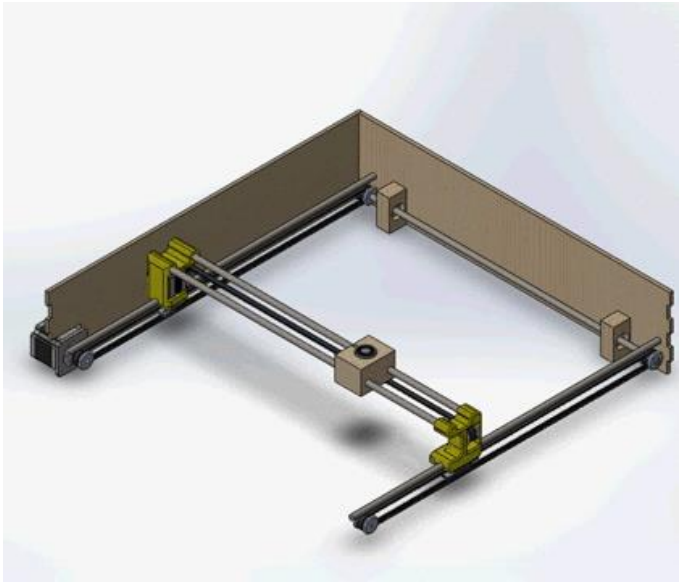
5.7.1 Voice Activation

One possible feature under consideration is to implement a voice for the chess engine. After the chess engine makes a move, the chess engine will announce where the move needs to be made so that the User can move the piece. For example, if the chess engine wants to move the knight on b8, the voice command would be “knight on b8 to c6.” One issue with this feature is that the voice feature would only be useful for advanced players. Beginner chess players will not be familiar with referring to chess pieces and locations by their square positions. Also, to incorporate this feature, a speaker will have to be added to the design of the board. Voice pattern software will also need to be included for the chess engine to speak. Python would be used for the voice pattern software because there are numerous python libraries for voice functions.

5.7.2 Magnetic Moving Chess Pieces

Another interesting feature to include is the ability for the computer chess pieces to move on their own. One method of accomplish this feature is to use stepper motors underneath the chess board. The stepper motor would have a magnet attached to it and use the magnet to attract with the magnet inside the chess piece. The stepper motor would need a mechanical device that can move on the X and Y axes. The stepper motor would get the information on where to move the chess piece from the chess engine. However, the stepper motor would need a method of determining where the squares are and be able to differentiate between squares such as between E4 and E5. Also, moving the knight in between pieces would be difficult and would require extremely accurate motors. Also, because the project is self-funded, buying stepper motors would greatly increase the final cost of the device. A mechanical system to move the motor along the two axes would also be expensive. The group does not consist of mechanical engineers, or anyone with mechanical knowledge to develop that type of system. The system would be like the design shown in the following figure.

Figure 60 Prototype Design of Stepper Motor with X and Y Axes



5.7.3 Smartphone Application

Another feature that was considered was a smartphone app to allow a User to play against another player in a different location. The smartphone app would allow a player to make a move on the app and see the opponent's moves as well. The chess board would need WIFI to connect to the smartphone app and gather the data for the move from app. Then using LEDs, the chess board would light up

where the opponent made a move. If the stepper motor is implemented, the motor would be able to move the piece to the correct location. However, a smartphone app is beyond the software development capabilities of the group. The smartphone app would require a graphic user interface as well as be able to utilize wireless communications to send a player's move to the chess board.

6.0 System Prototyping and Demonstration

The main purpose of system prototyping is to create a model that is as close as possible to what the system will potentially be like to get an idea of what the final product will entail. This is an essential part of any design product or project, as it takes in the ideas needed to implement and puts them into play. It will give us an idea of what may or may not need to be changed before the final product can be official. It can help to consider the unforeseen issues, design constraints, costs, and benefits.

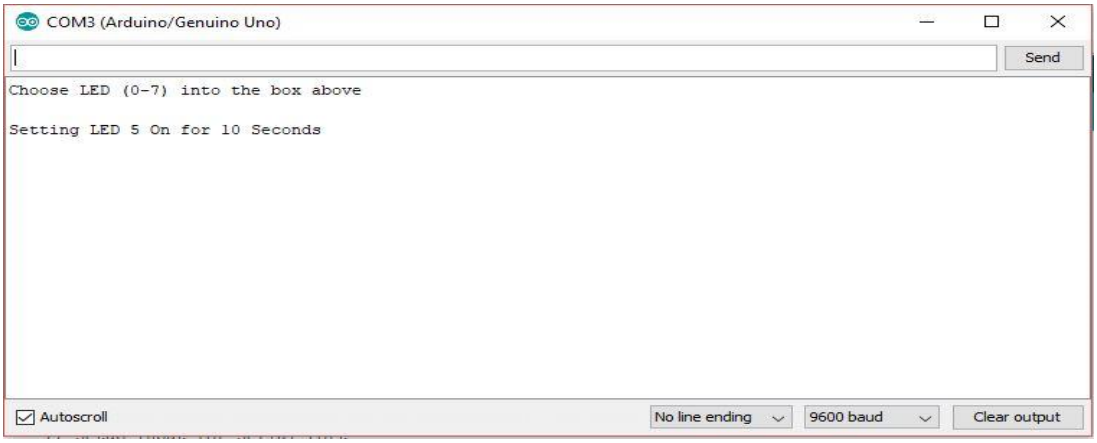
6.1 Hardware Testing

This section details how each major hardware component was tested to verify that each component will work with each other properly and work under the required specifications.

6.1.1 Shift Register

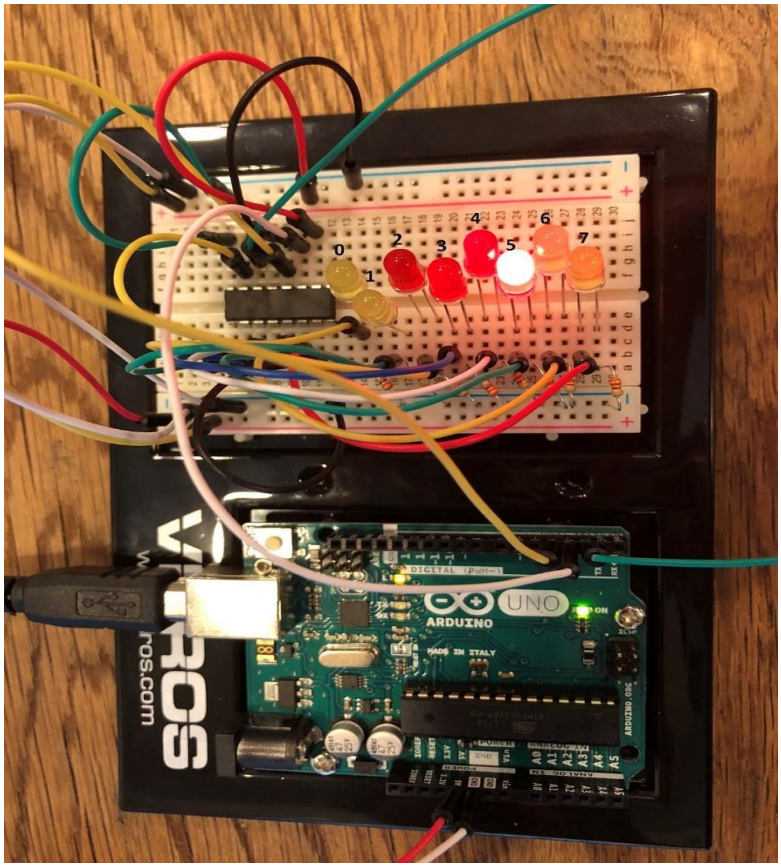
The chessboard is going to host 64 RGB LEDS which is going to be controlled by an LED driver. The LED driver can only control 8 RGB LEDs at a time. This can be fixed by using a shift register to select the row in which the LEDs will be controlled by. By using a shift register to run through eight individual rows, the microcontroller can create the illusion of all eight rows being on simultaneously if the clock speed is fast enough. To test the logic behind a shift, register eight LEDs were lined up in a row and hooked up to an Arduino Uno to see if the microcontroller would properly communicate with the integrated circuit. Once hooked up correctly according to the datasheet, software can be compiled to test the performance of the shift register on the LEDs.

Figure 61 Serial Command Prompt to Activate LED



Once the hardware and software were correctly implemented the LEDs were numbered from zero to seven (logic was based of the shift registers output pins). As can be seen in the figure below, LED five is activated when the serial prompt receives the number five by the user. As currently set up, LED five will receive a "HIGH" signal from the digital pin on the shift register and turn on. Since this is a simple test circuit to see how to communicate with shift registers; the LED is only designed to be on for ten seconds and then turn off. There were slight bugs that may become apparently in the final design. While the circuit did perform correctly in essence, initializing the code to the microcontroller caused all eight LEDs to turn on for about one second and then turn on. When the serial command prompt was opened the issue contained and turned on all eight LEDs once again and proceeded to turn off a short moment after. This isn't a big problem right now, but unforeseen consequences could arise from this simple mistake in software. Currently, it seems to be the case that when the program is initialized there is a small moment in time where the inputs of the shift register cause all eight outputs to activate on "HIGH". This issue will be investigated and fixed before the start of the final design to avoid any bigger issues.

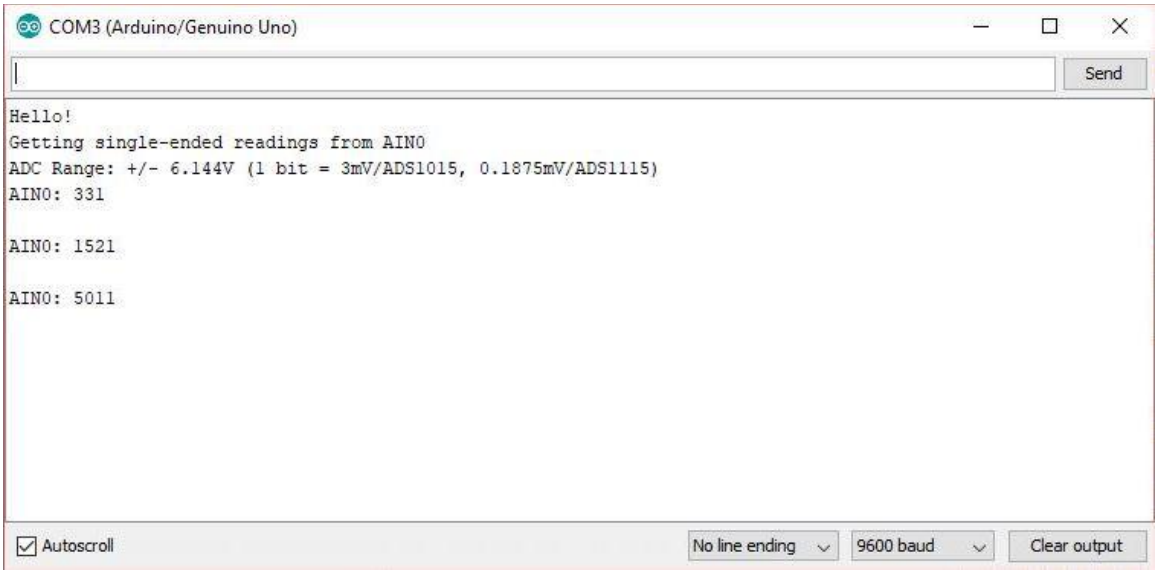
Figure 62 Hardware Wiring of Microcontroller with Shift Register



6.1.2 Analog-to-Digital Converter

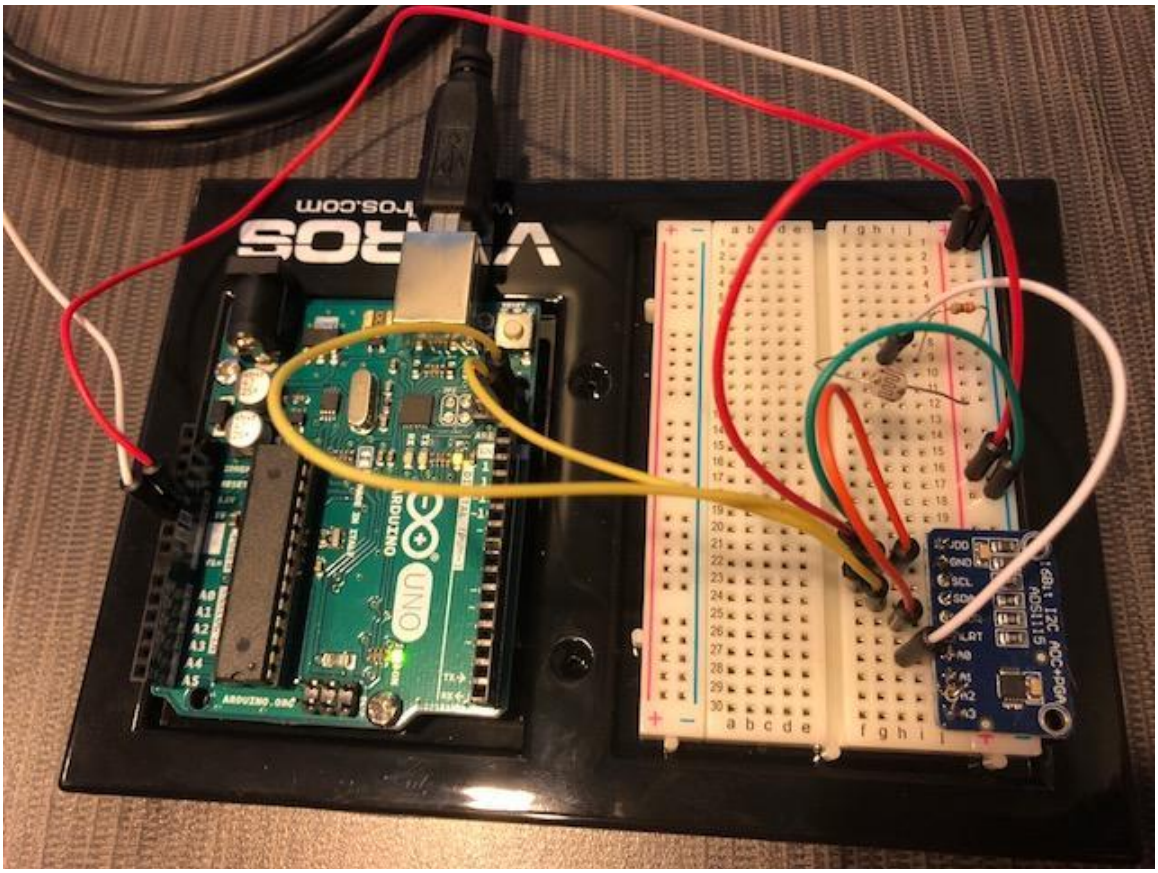
To provide some sense of direction for when measuring the chessboard reference voltages for piece identification, the ADC integrated circuit was set up. Unlike the other circuits, this device didn't follow SPI protocol but instead used I2C protocol for communicating. This communication protocol was easier to set up since there's plenty of open sourcing that handles the communicating side of things. It's also two pins to communicate compared to the three pins that the other integrated circuits required. Once the hardware was correctly wired, setting up an interface to see the analog signal that was being captured by the photoresistor was easy. Using the serial monitor of the Arduino IDE, three different scenarios were captured. The first scenario showed the analog input when the photoresistor was covered by a hand, the second was normal room light, and the third was with a flashlight shining on it.

Figure 63 Three Analog Inputs for Multiple Lights



Since the value only went up to 5011 in the testing, it was questionable if the ADC really had 16 bits of resolution. After further research, it was seen that the circuit only had 15 bits of resolution when using single ended inputs compared to the differential inputs which required the use of two inputs. That's not a problem for the end design of the project since 16 bits of resolution was overkill for the application at hand. Still though, the value was low for a device that's supposed to give 15 bits of resolution. It was later noted that the reason behind the resolution size of the testing was because of the limited light of the flashlight. Given a much stronger light source like the sun; the value that serial monitor returned was sufficiently large to indicate 15 bits of resolution.

Figure 64 Hardware Wiring of Microcontroller with ADS1115 A-D-C IC



6.1.3 Reed Switches

The reed switches were bought from Amazon for around two dollars. For testing purposes, only a pack of ten reed switches were bought. To test the reed switches, a multimeter was used to measure the continuity of the switch. The leads of the multimeter were connected to the terminals of the reed switch. Then a neodymium magnet was placed close to the reed switch to simulate the opening and closing of the switch when a chess piece is lifted off the board. A beep from the multimeter confirms that the switch is closed, and current can pass through the reed switch.

To test that the current was able to travel through the reed switch, a LED was placed in series with the magnetic switch and connected to a 1-volt DC input. When the magnet is not in proximity of the reed switch, there is no current traveling to the LED. When the magnet is placed next to the reed switch, the reed switch closes and allows the current to travel through. This can be seen as the LED lights up.

6.1.4 Magnets

The neodymium magnets were also tested to make sure that they would activate the reed switches from underneath the chess board. The current design of the chess board uses an acrylic top covering. The magnetic switch was placed underneath a piece of acrylic and attached to the multimeter to measure the continuity as the magnet approaches the reed switch. When the magnet is directly on top of the acrylic surface, the reed switch closes. The proximity needed for the magnet to trigger the switch is ideal because as soon as a chess piece is lifted from the board, the reed switch immediately opens. Thus, the current will stop flowing and the ADC will be able to instantaneously detect what piece was lifted and the shift register can highlight the appropriate square with very little lag noticed.

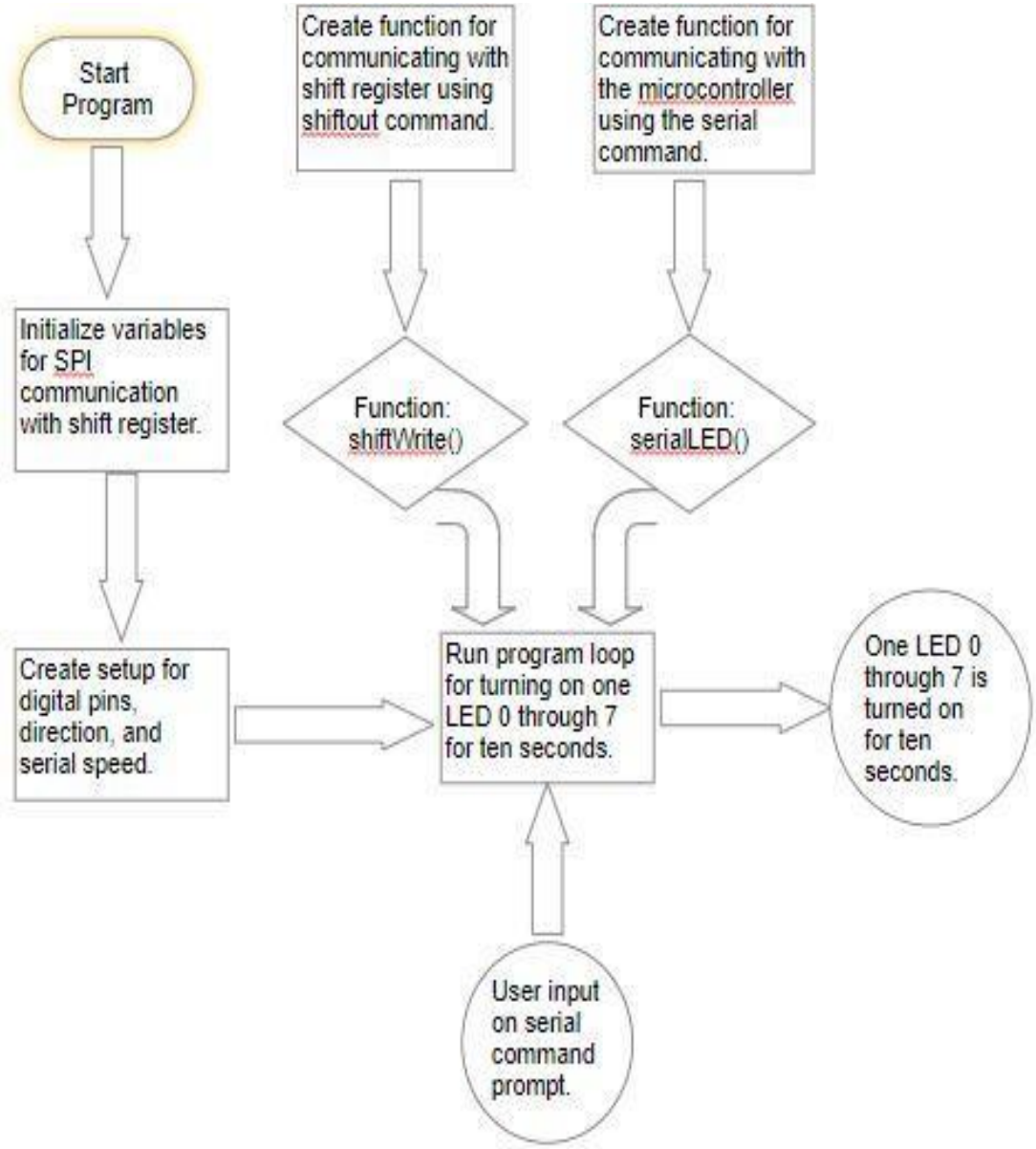
6.2 Software Testing

Making the correct connections with the hardware won't be enough for the design. Making sure the correct software is uploaded onto the microcontroller is just as important as making sure the correct wires are hooked up to the correct pins. Once the hardware is doubled checked, the software will be the next part. The first step for most of the coding required will be correctly setting up the SPI communication with the microcontroller and the integrated circuits it will be synchronizing with. Everything after that will be simple digital logic that's coded to meet our desired objective.

6.2.1 Shift Register LED Controller Software

Before starting on any software pseudo code should always be written down to ensure that the correct logic is implemented. The first step to any code is to initialize global and data variables. The Arduino Uno also typically has a setup loop that's used for initializing pins, allocating the correct direction of those pins, setting serial speed for communication, etc. The main program loop is usually featured after that and calls upon functions which are declared later in the code. For this code, there are two main functions that will be needed. The first once is the shiftWrite() function and is necessary because the shift register will be communicated with through this function. The second function is the serialLED() function that declares which LED shall be turned on depending on the number inputted by the end user. Once all this software is correctly implemented, the user can successfully tell the microcontroller which LED needs to be turned on.

Figure 65 Software for Controlling LEDs



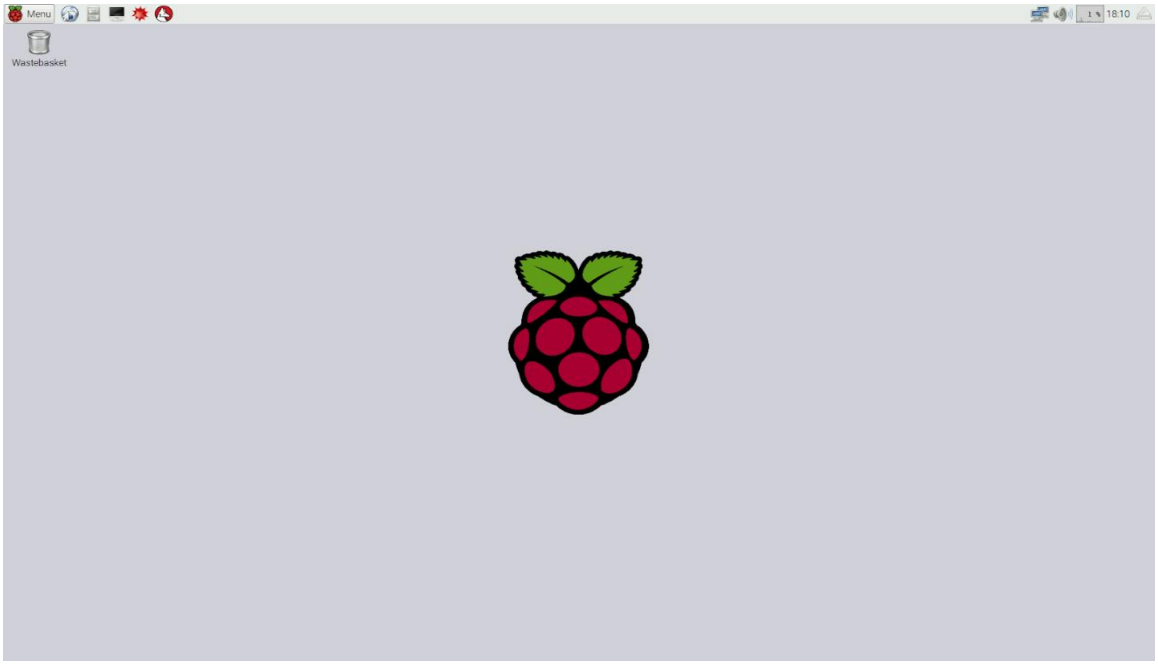
6.2.2 Analog-to-Digital Converter Software

Since there no software engineers on the project, a significant amount of the basic coding for communication protocols will copied from open source libraries. Using libraries from popular websites such as Adafruit, most of the code necessary for communicating an analog-to-digital converter circuit with the Arduino Uno was fairly simple. Very little code manipulation was required to get the software working with the hardware on hand.

6.2.3 Raspbian Operating System

The chess emulator will require an operating system to run which is why a micro-Linux computer was so important for the design of the project. The first thing required to go about this obtaining a SD card. Once that's in possession, the software for the operating system is open source but will download as an image format. Downloading software to decode this information will be required in order to have the operating system in a useable format. Once that's completed the operating system can be downloaded directly onto the SD Card and after a little bit of formatting the SD Card is ready for use on the Raspberry Pi.

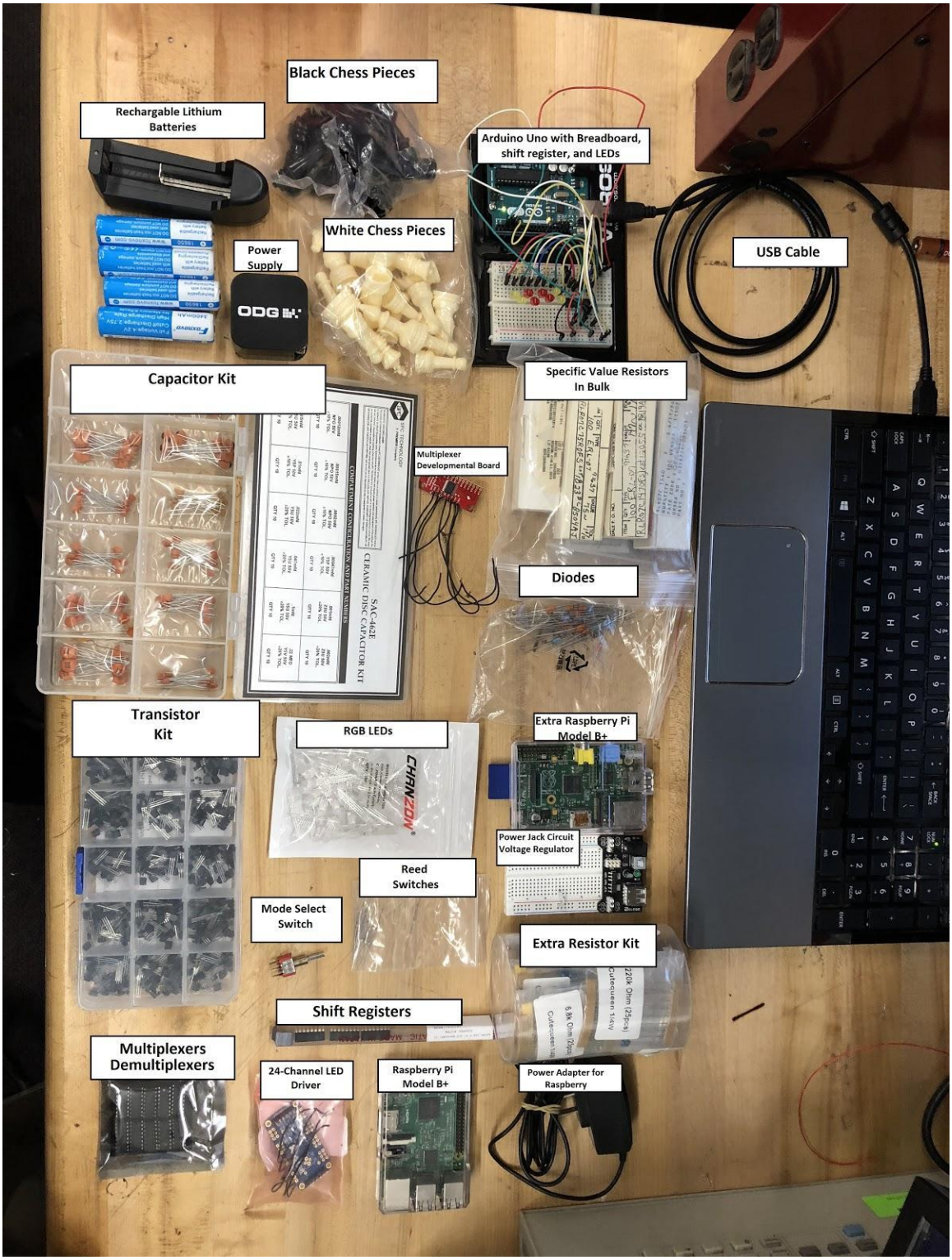
Figure 66 Raspbian Operating System



The Raspbian operating system to test the chess engine software as well as the C code required for the hardware to interact with each other. The Raspbian operating system is extremely lightweight, yet powerful and can run both the chess engine and the hardware C code at the same time. The Raspbian operating system used for testing including a graphical user interface. However, the final product will use the operating system version without a graphical user interface. All interaction with the operating system will be done through the Linux terminal. One downside of the Raspbian operating system is the loading time needed to boot up the operating system. There is also no way to store data in flash to prevent cold starts after a shutdown has occurred. Thus, the computer must bootup whenever the device is turned on, unlike a microcontroller which is instantaneous.

6.3 Testing Components

Figure 67 Parts for Testing



Most of the components required for the design of the project is readily available and ready for testing. Using block diagrams and schematics, it was possible to start creating subsystems that represent the final design. While the actual hardware and software implementation of these tests may not be exactly the same for the final physical presentation; it's important to understand the concepts learned during this testing for when debugging the final product. The following table will evaluate the importance of each individual component to the scope of the project.

Table 20 Components Available

Component	Function	Importance
Rechargeable Lithium Battery	Rechargeable power supply	Four pack battery which will be connected two in parallel and two in series for 6 - 8 volts input.
Capacitor Kit	Smoothing output voltages	Capacitors will be used when necessary to smooth out voltages.
Transistor Kit	Amplifying or switching signals	In the case that the LEDs require more current draw, transistors will be used to power them.
Multiplexer/ Demultiplexer	Selectors based on inputs	Will be utilized for inputting an array of data for piece detection.
Chess Pieces	Units for game	Consumer piece for playing the game. Will also inhibit a resistor with two magnetic plates attached.
Multiplexer Developmental Board	Ready to use 3-to-8 Multiplexer	Ready to use solid state multiplexer for testing.
RGB LEDs	Color control LEDs	Aesthetic color schemes and user interface for possible moves.
Reed Switches	Allow current if magnetic detection	Used to identify if there is a magnetic piece on the board and will allow a current to flow if so.
Mode Select Switch	Used to toggle the output	Will be used to toggle output given two inputs for testing.

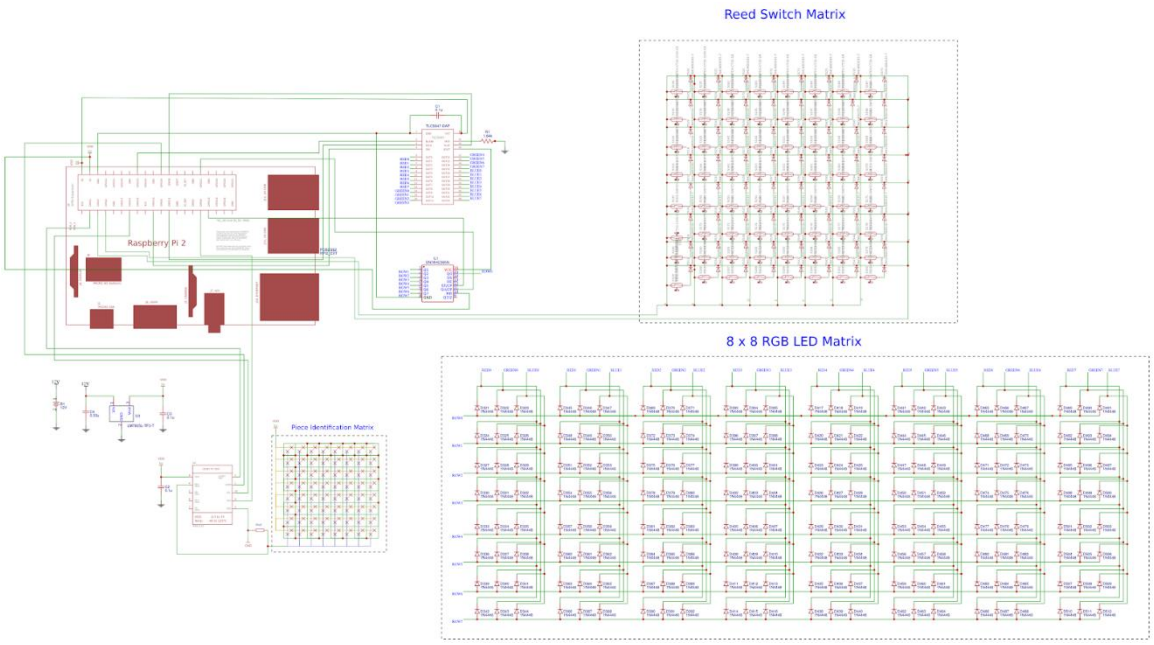
Shift Registers	Used to translate data to serial data	Will be used to select between rows of the LED driver and must appear fast enough to create the illusion of each row on simultaneously.
24-Channel LED Driver	PWM control over multiple RGB LEDs	Used to power the RGB LEDs and has 24 channels to power 8 separate ones at once.
Raspberry Pi Model B	Micro-Linux Computer	Main processing unit of the project. Will control all peripherals and also handle the chess emulator.
Arduino Uno	Microcontroller	Mostly used for testing the integrated circuits.
Resistors	Used to reduce current	Pull up resistors will be used when needed and according to datasheet specifications.
Diodes	Used to control current direction.	Will mostly be used for power protection for the integrated circuits.
Power Jack Voltage Regulator Circuit	Regulates voltage to 5 volts when connected to power jack	Will be used in conjunction with the batteries to regulate the voltage to a sustainable 5 volts
Power Adapter	Provides 5 volts of power	Main use is to power the Raspberry Pi for testing.
USB Cable	Provides 5 volts of power	Main use is to power the Arduino Uno for testing

Having all the components on hand this early into the project scope puts us well ahead of schedule. All the test circuits that were run to see if the integrated circuits behaved just as they were expected to run is also a huge help and will reduce a significant amount of debugging time when the final product is constructed. The only thing missing from the table and picture is the Analog-to-Digital Converter which came last in the shipment but was already test and implemented with the Arduino Uno. There are no more parts that we believe we may still need except for the PCB board which will be ordered early into senior design two.

6.4 Prototype Circuit Design

The schematic for the overall prototype circuit design connecting the system can be seen below. The major components described from the block diagram are fully implemented and portrayed. This model will serve as the general system design for implementation when testing out the full design.

Figure 68 Prototype Circuit Design



7.0 Administrative Content

In this chapter, administrative content including timeline milestones and budget analysis will be researched to ensure that the project is both realistic and completed in a timely manner.

7.1 Milestones

In order to not fall behind as a group, the milestones list allows for the group to have a rough outline of what is needed to be finished in a timely fashion. The outline gives a guide for when each milestone should be completed by for Senior Design 1 and Senior Design 2. The timing for each milestone is subject to change based upon group progress and other constraints. To assist with deadlines, schedules will be made, and the workload will be divided up so that each group member is responsible for their part of the design process

Table 21 Senior Design 1 Milestones

SUMMER 2018		
Description	**Time**	**Dates**
Project Idea/Division	1 week	May 21 - May 27
Documentation	1 week	May 28 - June 7
Initial Project Documentation		8-Jun
Research on past projects	2 weeks	June 11 - June 24
Individual writing	2 weeks	June 25 - July 5
Initial Draft (60 pages)		6-Jul
Design & Development prototyping	2 weeks	July 9 - July 19

100 Page Submission		20-Jul
Finish documentation	5 days	July 21 - July 25
Documentation Review/Purchase Components	4 days	July 26 - July 29
Final Documentation Submission		30-Jul

Table 22 Senior Design 2 Milestones

FALL 2018		
Description	**Time**	**Dates**
Component Testing	1 week	August 20 - August 26
Build Prototype	8 weeks	August 27 - October 14
Test Prototype	3 weeks	October 15 - November 4
Finalize Project	2 weeks	Nov 5 - Nov 18
Final Documentation and Presentation	2 weeks	Nov 19 - Dec 1

7.2 Budget Analysis

A rough estimate of the budget is provided to show best and worst-case scenarios for pricing on the parts. These prices are subject to change if certain materials or parts are chosen or changed once implementing the final product. All the prices were gathered from online research and vendor sites. All parts considered are listed in the table below and give a rough estimate of the price range possibilities

found from different vendors. Price range possibilities may be vast in difference due to a multitude of reason. Some reasons include the range of options for a part, the precision of the part, or just the fact that it could be a high reputable vendor making the part. As of right now, all the expenses shall be divided evenly amongst the members of the group to ensure fairness since there is currently no sponsor at this time. The team will keep a detailed log of expenses that detail the item the quantity and the price. Extra parts and no used items can either be split by the group and thrown away or kept by a team member and the cost of those parts deducted to the final split cost

Table 23 Parts List

Item	Supplier	Cost/Item	Number Of items	Total Cost	Purchaser
Common Anode LEDs (pack 100)	Edgelec	8.99	1	8.99	Eric
16bit ADC dev board	Hiletgo	6.99	1	6.99	Jean
Chess Piece Set	Amazon	11.08	1	11.08	Eric
Mux/Demux (pack 10)	TI	7.99	1	7.99	Eric
24 Channel LED driver Dev Board	Adafruit	18.97	1	18.97	Eric
USB FTDI Adapter	Amazon	5.99	1	5.99	Eric
Lauan Plywood	Lowes	6.32	2	12.64	Brandon
Test Magnets	Amazon	12.3	1	12.3	Saeed
Piece Magnets	Amazon	10.2	1	10.2	Saeed
Little Magnets	Amazon	9.99	4	39.96	Saeed
Test Reed Switches	Amazon	8.88	1	8.88	Saeed
Reed Switches	DigiKey	0.51	120	61.2	Saeed
PCB	JCBPCB	21.7	1	21.7	Brandon
Battery Holder	Amazon	6.99	1	6.99	Eric
FTDI USB Adapter	Amazon	9.99	1	9.99	Eric
Male Header Connector	Amazon	7.99	1	7.99	Eric
Female Header Connector	Amazon	6.82	1	6.82	Eric
1/4 inch acrylic sheet	Amazon	12.99	1	12.99	Eric
ATMEGA 2560 dev board	Arduino	21.5	1	21.5	Jean

Assorted Resistors	Arrow	7.44	1	7.44	Eric
Capacitor and Resistor	Arrow	1.98	1	1.98	Eric
Copper and Pogo Pins	Amazon	28.75	1	28.75	Jean
Primary PCB parts	Arrow	51.19	1	51.19	Brandon
PCB Boards x5	JLCPCB	31.3	1	31.3	Eric
			TOTAL COST	**413.83**	

7.3 Team Member Task Division Table

Below is a table describing owner and sub owner of the basic tasks and subsystems defined in this project. The primary team member is the team member who owns the portion of the project. The second member is the support, also working on it to complete deadlines but under the supervision of the owner.

Table 24 Task Division

Task	Primary	Secondary
Power/ Microcontroller Hardware	Jean	Saeed
Chess piece identification matrix hardware	Eric	Brandon
LED Matrix Hardware	Brandon	Eric
Switch Matrix hardware	Saeed	Eric
Mode Select	Jean	Brandon
Chess Engine	Saeed	Jean
Chess piece identification Matrix software	Eric	Saeed
LED Matrix Software/ Firmware	Brandon	Eric
Switch Matrix Software	Saeed	Eric
Overall software/firmware	Jean	Saeed

8.0 Conclusion

The research and design highlighted in this paper has a purpose of defining how the final smart chess board will come together. The basis of the paper is to describe what technologies will be used in developing the project and then elaborate on the future design that the team will build after the paper is written. Minimal testing and prototyping is done during the writing of this paper. Each part was chosen based on a set of criteria determine by the group as well as the goals of the project. Each group member was assigned specific areas to research and present to the team. Each group member was responsible for researching multiple solutions to the design areas assigned to them. Each group member has to meet deadlines or risk not completing the project on time.

Now the group has narrowed down how each problem will be solved and what parts will be used. The parts have been ordered, the necessary components tested and connected to the peripherals. The next step is to design the PCB and start building the first fully functioning prototype. This design processes will be ongoing. This paper goes into details about the design, but this will most likely change during prototyping phase. Despite a reduced semester length and busy schedules, each group member has been diligent in completing their assigned tasks.

This paper represents stable research, development and design of the Smart Chess Board. The future of chess and how people will learn it is going to change based on what is done here. This project is the next step into what could become a line or smart gaming systems based on old generation games. Chess is a great starter because it’s complicated to learn. The goal of this project is to further develop smart chess board systems by adding this team's unique design and functionality criteria, providing a stepping stone for future engineers to continue off the design and engineer something revolutionary to the chess playing world.

Throughout the course of Senior Design II, many skills were developed. As the project was discussed and designed between the group, each group member has learned how to work in cohesion with one another. As design challenges pop up, various precautions and designs must be considered to solve each challenge. Occasionally, this meant redesigning a schematic from the beginning. The switch matrix had to be redesigned several times as new research was found, and problems were encountered during testing.

Now that the parts for each section and schematic of the project have been bought and tested, the goal is to build each separate system and connect them together to develop a working prototype. This working prototype must be able to properly demonstrate the main functions of the final design to a panel of judges. Failure to

do so means that our design is flawed, and certain design will have to be redesigned which would severely put our group behind schedule.

The hardest challenge coming forward is the design of the physical chess board. There are multiple design constraints associated with the physical chess board. LEDs must not shine through multiple chess squares and each chess square must have metal contacts for each chess piece to connect to. Developing the physical chess board also requires the group to learn how a CNC milling machine works. Since the project is not sponsored, careful attention to the parts used and destroyed must be taken into consideration. Buying multiple parts increases the total cost of the final project.

Overall, the group feels confident in their design and is ready to begin the assembly of the smart chess board in Senior Design II. Several parts have already been tested and several schematics have already been built and tested for functionality. All that remains is to build the final schematics and connect each subsystem together.

9.0 Appendices

9.1 Bibliography

[1] “History of Chess: The Basics.” *Chess.com*, Chess.com, 28 Jan. 2009, www.chess.com/article/view/the-history-of-chess.

[2] “Endgame Tablebase.” *Wikipedia*, Wikimedia Foundation, 5 July 2018, en.wikipedia.org/wiki/Endgame_tablebase.

[3] “Minimax.” *Wikipedia*, Wikimedia Foundation, 3 July 2018, en.wikipedia.org/wiki/Minimax.

[4] “Learn to Play Chess.” *Chess Corner*, www.chesscorner.com/tutorial/learn.htm.

[5] “World Chess Handbook.” *World Chess Federation - FIDE*, www.fide.com/handbook?option=com_handbook.

[6] "*IEEE Std 1118.1-1990*", IEEE Standard for Microcontroller System Serial Control Bus

[7] “Packaging Terminology.” *Texas Instruments*, www.ti.com/support-packaging/packaging-resources/packaging-terminology.html.

[8] “USB-IF Developers Area.” *Universal Serial Bus*, www.usb.org/developers.

[9] “IEEE” *Design and Implementation of Autonomous Vehicle Valet Parking System - IEEE Conference Publication*, Wiley-IEEE Press, ieeexplore.ieee.org/stamp/stamp.jsp?tp=.

[10] “Windows 10 Internet of Things.” *Meet the Evangelists*, developer.microsoft.com/en-us/windows/iot.

[11] Canonical. “Ubuntu Core.” *White Label App Store for IOT | Ubuntu for the Internet of Things | Ubuntu*, www.ubuntu.com/core.

[12] “Kali Linux on Raspberry Pi.” *You Are Being Redirected...*, docs.kali.org/kali-on-arm.

[13] “Raspbian.” *Raspberry Pi*, www.raspberrypi.org/documentation/raspbian/.

[14] Extra, DGT. “DGT Smart Board - Digital Game Technology.” *Digital Clocks - Digital Game Technology*,

www.digitalgametechnology.com/index.php/products/electronic-boards/smart-board/554-dgt-smart-board.

[15] “UCI Protocol.” *UCI Protocol*, wbec-ridderkerk.nl/html/UCIProtocol.html.

[16] Joshua. “Ghosting and Masking.” *Weirdscience TV*, weirdscience.us/index.php/2017/03/01/ghosting-and-masking/.

[17] “NASA TECHNICAL STANDARD: SOLDERED ELECTRICAL CONNECTIONS.” *NASA*, Dec. 1997, www.nasa.gov/.

[18] Agarwal, Tarun. “Different Types of Diodes and Their Uses.” *Overview of Various Types of Diodes and Their Uses*, www.elprocus.com/types-of-diodes-and-applications/.

[19] “Micro-Max.” *The Shanghai Connection*, home.hccnet.nl/h.g.muller/max-src2.html.

[20] Regimbald, Adrien. “Home.” *Homepage of Faile*, faile.sourceforge.net/index.php.

[21] “GNU Chess 6.1.2.” *[A GNU Head]*, www.gnu.org/software/chess/manual/gnuchess.html.

[22] “Sunfish.” *Chessprogramming*, chessprogramming.wikispaces.com/Sunfish.

[23] “24-Channel, 12-Bit PWM LED Driver with Internal Oscillator (Rev. B).” *Texas Instruments*, Jan. 2015, www.ti.com/lit/ds/symlink/tlc5947.pdf.

[24] “SN54HC595, SN74HC595 8-Bit Shift Registers With 3-State Output Registers Datasheet (Rev. I).” *Texas Instruments*, Sept. 2015, www.ti.com/lit/ds/symlink/sn74hc595.pdf.

[25] “TPIC6B595 Power Logic 8-Bit Shift Register (Rev. B).” *Texas Instruments*, June 2015, www.ti.com/lit/ds/symlink/tpic6b595.pdf.

[26] “Ultra-Small, Low-Power, I2C, 16-Bit ADC With Int Ref, PGA, and Prog Comparator Datasheet (Rev. D).” *Texas Instruments*, Jan. 2018, www.ti.com/lit/ds/symlink/ads1115.pdf.

[27] “uA7800 Series (Rev. J).” *Texas Instruments*, May 2003, www.sparkfun.com/datasheets/Components/LM7805.pdf.

9.2 Permissions

Authorization from Adafruit

Re: [[EDUCATOR INQUIRY]] Inbox x

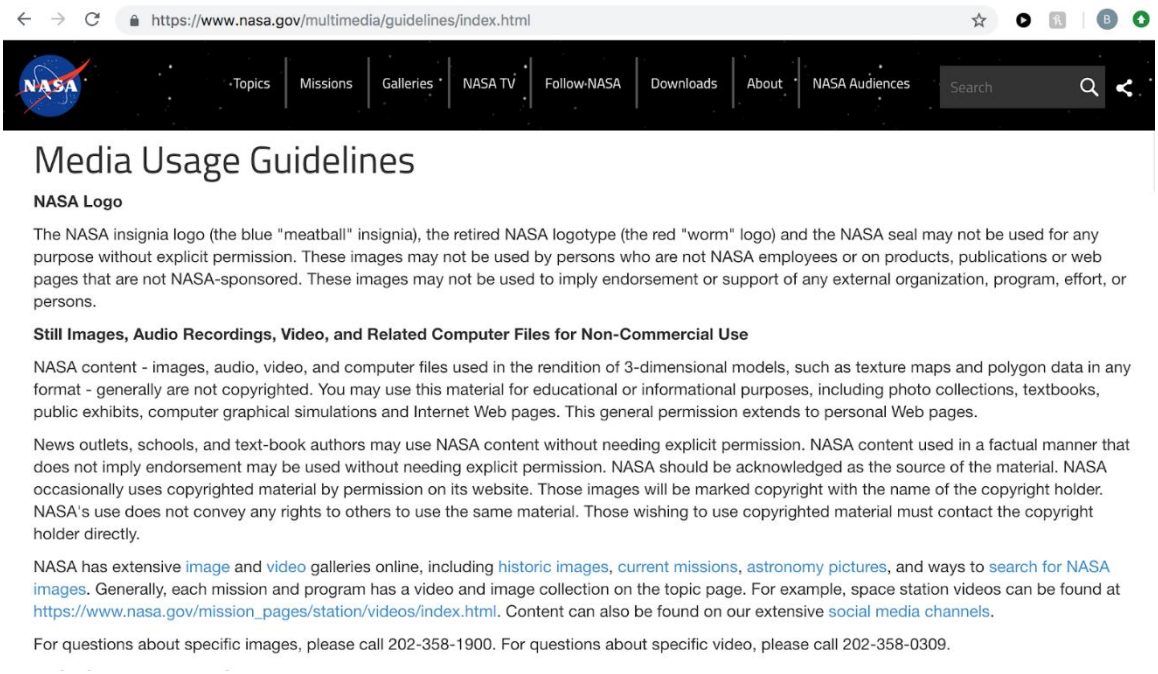
Adafruit Industries <support@adafruit.com> Jul 23 (6 days ago)
to me

totally OK, thanks for the note.

On Mon, Jul 23, 2018 at 11:35 AM, Brandon Dupoux <support@adafruit.com> wrote:

> contactname : Brandon Dupoux
> email address : brandon.dupoux@ieee.org
> Message : Good morning Adafruit,
>
> I have an inquiry on allowing for the use of a schematic capture in our report. I am a student from the University of Central Florida and currently in a group for our senior design project. We are currently using the Adafruit TLC5947 development board in order to test the T.I. component. I would like to ask for permission to use your schematic capture diagram picture under the "TLC5947 Schematic & Print" section of the link:
>
> https://learn.adafruit.com/tlc5947-tlc59711-pwm-led-driver-breakout/downloads-and-links
>
> We will be including this picture to show how we went about testing this part in our report.
>
> Thank you,
> Brandon Dupoux
> Client IP: 2600:387:9:3:0:0:0:bb

Authorization from NASA



Authorization by Texas Instruments

Texas Instruments is pleased to provide the information on these pages of the World Wide Web. We encourage you to read and use this information in developing new products.

TI grants permission to download, print copies, store downloaded files on a computer and reference this information in your documents only for your personal and noncommercial use. But remember, TI retains its copyright in all of this information. This means that you may not further display, reproduce, or distribute this information without permission from Texas Instruments. This also means you may not, without our permission, "mirror" this information on your own server, or modify or reuse this information on another system.

TI further grants permission to nonprofit, educational institutions (specifically K12, universities and community colleges) to download, reproduce, display and distribute the information on these pages solely for use in the classroom. This permission is conditioned on not modifying the information, retaining all copyright notices and including on all reproduced information the following credit line: "Courtesy of Texas Instruments". Please send us a note describing your use of this information under the permission granted in this paragraph. Send the note and describe the use according to the request for permission explained below.