Blitz Board

By

Nick Bingenheimer

Owen Shin

James Tang

Final Report for ECE 445, Senior Design, Spring 2023

TA: Hanyin Shao

03 May 2023

Project No. 66

Abstract

Our proposed project, Blitz Board, aims to develop a way to play chess against a computer at a physical interface. We implemented a dual-axis motor system with three motors to move the metallic chess pieces and an electromagnet to grab them. With reed switches embedded under the chessboard, the system can know the location of every chess piece. There is also a display to show the time, and the user is able to interact with the clock through a button.

Contents

1. Introduction

This report will cover the many aspects of the blitz board, from the challenge this project attempts to overcome, to design considerations and engineering decisions made, as well as the problems occurring in the design and the setbacks faced along the way.

1.1 Problem

When one plays chess against a remote player or bot on chess.com, there is no physical component- the board is replaced with a display and mouse. A good solution is a two-axis motor system to move pieces from beneath the board on behalf of a remote or virtual opponent, but this method is slow, especially for “takes”, where a piece must be taken off the playing field. One existing product on the market, Square Off, does this by taking a piece off the board and then moving another into its place. This requires up to twice this distance traveled compared to taking a piece off the board as the second movement. This lack of speed from the computer side restricts the variety of game modes that can be played on the board. Additionally, most players punctuate their moves by pressing the switch on a chess clock, which is not featured as part of the existing solutions.

1.2 Solution

Our solution to this problem is to expand upon the capabilities of the two-axis motor system by implementing faster movements on the computer side. More efficient pathfinding for a given move will allow for faster games of chess, such as blitz chess. We wished to also create space for a digital chess clock that both the player and the computer will be able to control via electromechanics. This would further replicate the experience of an in-person game of chess with physical pieces and a chess clock with a mechanical switch. Additionally, we wished to expand upon the ability of the board to communicate with the player via lights within the board that can communicate illegal moves, the last move made, or even suggestions for the player if time permitted.

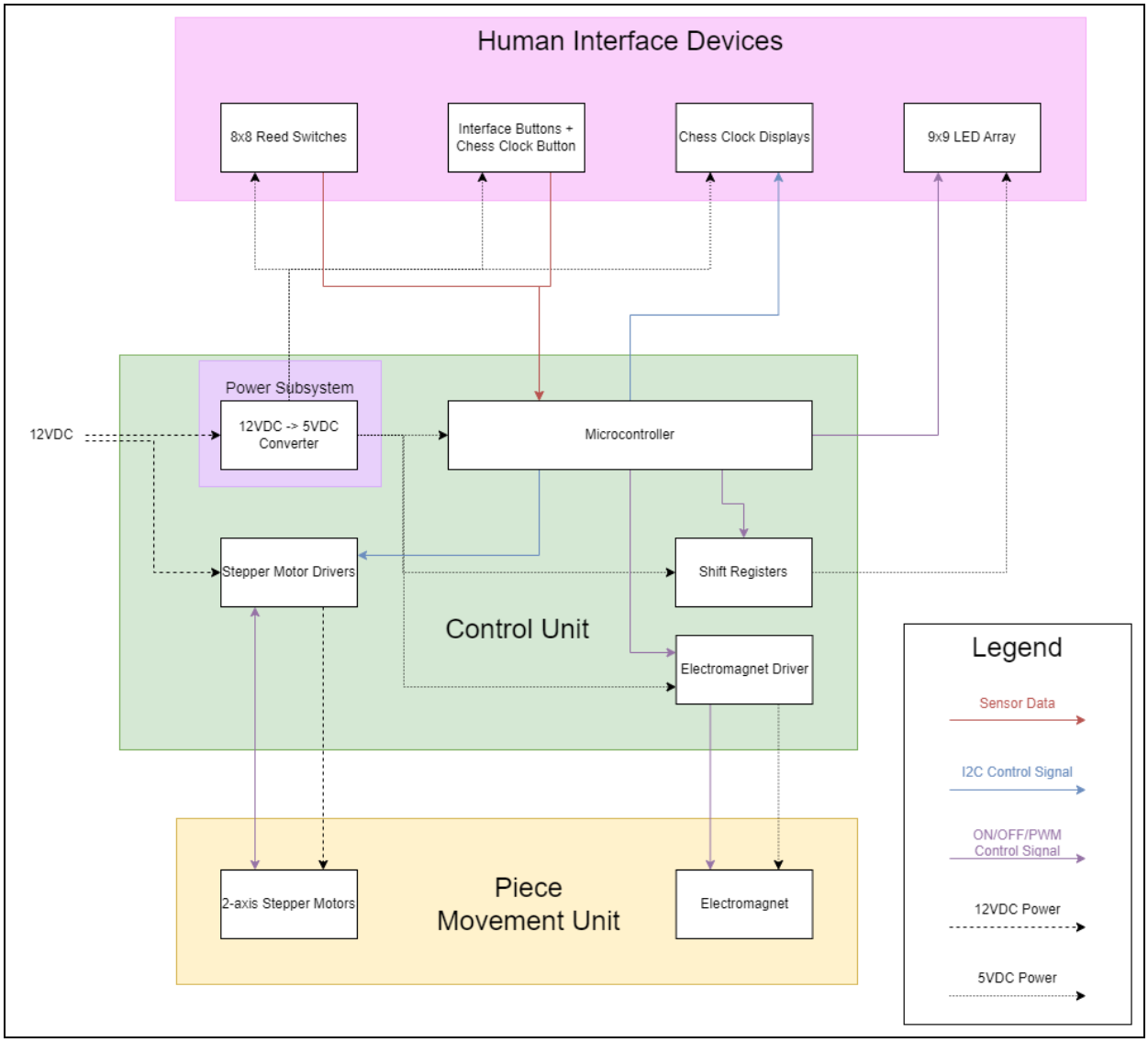

Figure 1: Block Diagram of Original Design

2 Design

The General design of the Blitz Board is similar to a 3D printer in some ways. The board is constructed of a large wooden base with tall metal standoffs. Secured by bolts, two layers of plexiglass act as the tabletop of the board. Underneath this board top, three stepper motors placed in a belt system act to move aluminum rails in both horizontal directions. An electromagnet is mounted at the intersection of this dual-axis motor system which can be switched on and off to grab pieces, while the motor moves this electromagnet around the board. Sensors placed in between the two layers of the top of the board will be able to sense the location of pieces and relay this data back to a microcontroller on a custom PCB.

2.1 Design Procedure

The design procedure we followed was mostly informal. As a group, we would meet to discuss ways to accomplish the different tasks we needed to complete, and how each of these tasks would contribute to the final product and high-level requirements. As per our team's contract, the design work at the lower level was split up by expertise and subsystem.

2.2 Human Interface Unit

The user interface allows users to interact with the board in several ways. It is connected to the control unit, and it consists of a led display, chess clock components, and several buttons. We hoped to have some sort of On/Off switch, a button to operate the chess clock on the user side, HEX displays to denote time on the clock, and LEDs to provide extra information to the user during gameplay.

Another option we considered for the chess clock was to implement a lever switch, similar to a real chess clock. It would be operated by a solenoid plunger pulsed by a magnetic field and the human player. This would preserve an elevated authenticity of playing chess at a physical location and would be a critical function to the option of playing Blitz Chess. However, the lever switch was abandoned as it strayed too far from our high-level requirements and the group decided we lacked the time to implement it properly.

2.2.1 Reed Switches

The reed switches underneath the board floor report the position of chess pieces to the microcontroller. They detect the presence of the magnetic field generated by the magnets placed in the base of the pieces, and their output voltages are directly proportional to the strength of the magnetic field. These will not affect the use of the electromagnet for moving pieces as the signals they provide to the control unit won’t be processed while the electromagnet is moving, and they will be placed out of the physical way of the Piece Movement Unit.

The original design included hall effect sensors to be placed on the board top that could sense the position of pieces on the board. However, they were abandoned at the behest of the ECE Machine Shop as it was deemed that we lacked the space to put them in and was not possible. Additionally, we discovered our own challenge with using the hall effect sensor as the output of the hall effect sensor was a fully analog signal, and would require too many analog-digital converters to implement into a multiplexer. With the reed switch, the case of each sensor switch was considerably smaller and easier to place into the board, and a digital signal could be carried by each. This allowed us to directly multiplex the data from the reed switches.

2.3 Control Unit

2.3.1 Microcontroller

The main task of the microcontroller, the MSP430F6736 from Texas Instruments, is to detect and respond to the player's moves. The intent was to run the Stockfish chess engine on the microcontroller and process sensor input and user input in order to output computer moves for the motors and electromagnet to carry out. This microcontroller would handle all other peripherals in the system, such as the chess clock and multiplexed LEDs.

The MSP430F6736 was chosen for its cost-effectiveness while fulfilling the requirements to run Stockfish and manage the relevant peripherals. This microcontroller had 9 ports, which exceeded the 6 ports required for this system. It also had 128 kB of flash memory, which was required to store Stockfish and the rest of the firmware to manage the system. Processing speed and ADCs were not relevant to this project due to the computational demand of Stockfish and the nature of the sensors used; microcontrollers with greater processing power or high-quality ADCs were more expensive.

2.3.2 Motor Drivers

Since the focus of this project was on developing a chess board, and just on driving stepper motors, we decided that it was best to buy off-the-shelf drive circuits that were capable of driving our motors. After some research, the group settled on the EasyDriver4.5 [1]. It was designed by Brian Schmalz, and had all of its data available as open-source, and was specifically designed with our model of stepper motor in mind. Each EasyDriver was only capable of running two motors at a time, so two were purchased so that all three motors could be driven with similar programming and control signals. Unfortunately, we found out late into the project that one of the drivers was a faulty piece and did not function properly, and had to procure a replacement, the L298N motor driver, in a short time.

We decided to control two of the motors with the EasyDriver and one of the motors with the L298N motor driver. However, the controlling mechanisms of the two drivers are different. The EasyDriver controls the motor by directly generating a step wave and setting the number of cycles of the step wave. We are also able to select the step size directly. On the other hand, the L298N motor driver only allows us to place the step size and step per revolution. In order to coordinate the two different mechanisms, we found an Arduino library called AH EasyDriver, which transforms the controlling method of the EasyDriver into something similar to the L298N motor driver.

2.3.3 Multiplexing

The multiplexing circuit within the control unit was designed to allow the microcontroller to read from a large number of sensors and control a large number of LEDs. This circuit used two SN74AHCT594 shift register ICs connected in series to sequentially supply a positive voltage to one of 16 columns of reed switches and LEDs. The cathodes of the LEDs were connected to BJTs which were controlled by a port of the microcontroller. The cathodes of the reed switches were connected to a voltage divider such that their state could be read by the microcontroller as a digital signal.

The choice in shift register IC was arbitrary as many similar options with 8 or 16 registers were available. The SN74AHCT594 was chosen because it was cost-effective and well-documented.

2.4 Piece Movement Unit

The Piece Movement unit consists mostly of stepper motors and electromagnets. Both of these components will receive data through the microcontroller and will operate according to the instructions given by the control unit. A motor driver component will be used to interface with the microcontroller and stepper motors, and will ensure that with the control signals given by the microcontroller, the motor will move the correct amount of steps at a certain speed.

2.4.1 Stepper Motors

The stepper motors we used came with the creation the ECE Machine Shop gave us. This board was used in a similar, yet unsuccessful project from a previous semester. The stepper motors are simple 2-phase, variable reluctance machines. They have a 1.8° step angle, which means that when the motor is told to move one step in any direction, the motor will spin 1.8° and remain stationary until told to do otherwise. These motors communicate in steps, so careful math must be done in order to translate the desired rotation angle into steps. Furthermore, an additional step of calculations must be done in order to understand how far you wish the motor to move, based on the diameter of the gear-belt system used. This will directly translate to an angle, and that angle will directly translate to steps. Additionally, the direction of rotation about the motor axis is important as the motor is able to take steps in any direction it is instructed to, those directions being clockwise or counterclockwise.

2.4.2 Electromagnet

The electromagnet will remain at a position at the intersection of the two-axis motor set-up. A simple transistor will be used to turn the electromagnet on and off. The transistor must be able to withstand a large enough amount of current such that the electromagnet can create a large enough magnetic field to carry a piece along the surface of the board. Additionally, the transistor will receive its gate signal directly from the microcontroller.

Originally, the electromagnet that came with the board from the ECE Machine Shop was a small 5VDC electromagnet. This electromagnet proved to be too weak and could provide enough field to reach through the board top, especially when the Reed Switches were embedded into shallow channels in the board. This prompted us to request a larger 12VDC electromagnet to be mounted in place of the original one. In exchange for more power drawn from the increased current rating of the new magnet, we were able to move pieces much more reliably. However, switching to a larger electromagnet also limited the range of motion of the stepper motors, and just made the board a little bit tighter.

3. Design Verification

The Design Verification section will highlight experiments designed by team 66, that verify the working order of the Blitz Board by subsystem. The Subsystem Requirements Table can be found in Appendix A and includes a thorough explanation of how to test them. Due to the limited time and the lack of a working microcontroller, the group was not able to test all subsystems of the Blitz Board.

3.1 Human Interface Unit

For most components of the human interface unit, functionality could be expressed simply as whether the microcontroller was registering an input or controlling a peripheral as intended. All inputs were digital and could be easily correlated with events on the chessboard. Verification of the reed switches would have involved the microcontroller receiving a 0 from all reed switches when no magnets are present, and receiving a 1 from any reed switch which is near a magnet of sufficient strength. The chess clock and LEDs could be verified visually as a graphical output matching the intended result.

3.2 Control Unit

Verification of the control unit would have focused on the microcontroller and multiplexing circuit. While other peripherals were connected to the control unit, these could be verified without the control unit and tested a second time for functionality when controlled by the microcontroller.

3.2.1 Microcontroller

The main way to verify the microcontroller was to connect it to a computer through a MSP430 JTAG debugger and observe whether the computer can upload firmware to it. Functionality can also be shown through the ability to step through a program or read registers from the computer. The microcontroller was capable of all of these tasks.

3.2.2 Multiplexing

The multiplexing circuit could be verified through successful control of which register outputs a high signal. This would be done by the microcontroller modulating the clock and input data pins of the shift registers. A successful test would involve one shift register outputting a high signal while the rest are low, beginning with the first register and finishing with the sixteenth. It would also be important to test that a high voltage would be maintained under the maximum load of a register, i.e. all LEDs in a column on and all reed switches in a column closed.

3.2.3 Motor Drivers

We hard-coded several cases to imitate the moves of pawns, bishops, and knights. The verification of the motor drivers is done as follows. One of the team members sits in front of the chess board and decides on multiple chess moves he wants to perform. For example, bishop to b6 or knight to d7. The other team member then enters the corresponding command into the Arduino. As a result, we were able to verify that the motor drivers could indeed reliably control the stepper motors to move in the desired direction and distance.

3.3 Piece Moving Unit

The main function of this unit is to move pieces across the board. When the board is crowded with pieces, it will be difficult to move pieces along the spaces, and thus we planned to design the software to move pieces in the lanes in between chess spaces. In the end, it was clear that the pieces had to be moved precisely so as to not interfere with other pieces on the board. Additionally, the electromagnet needed to be controlled carefully so that the pieces could be placed just as carefully. Thus, most of the verification tests of this unit included ensuring that the pieces were able to move precisely and in a controlled manner.

3.3.1 Stepper Motors

The stepper motors were the biggest part of this design, and their control directly affected how the pieces moved on the board. One of the high-level requirements of the Blitz Board was to be able to move faster than other boards on the market. After studying some film from the Square-Off company of Auto-Chess Boards, we estimated that the most realistic goal for our Blitz Board would be to have the motors move pieces at a speed greater than 1 space per second. Since our board was designed with some extra space in mind, our spaces were designated to be 2 inches on each side and thus would require that the motors be able to move pieces at a rate of 2 inch/s. In the following figures, some data on the effects of different step modes and step-signal frequencies on the speed of the stepper motors across the board. We accomplished this by setting a 10-inch runway for the motors to move, and we recorded the time elapsed for the motor to move the magnet over the 10in. From this, we could deduce the linear speed of the motors in inches per second. The default setting of the EasyDriver was to drive the stepper motors in Full-step operation, where the motors rotate a full 1.8° per step. The group found that this produced a very slow, and very loud “thrumming” that was absolutely not acceptable. We believe that in the Full-Step operation, the motors were not able to produce enough torque to move the electromagnet, and this failure was producing the loud noise we heard from the board. We tried different step operations and settled on the Eighth-Step operation to greatly reduce the noise and reach the desired speed. As seen in the plots below, the Quarter-Step operation reached the desired speed but suffered less control of speed through the step signal frequency. Additionally, we struggled to keep hold of pieces when the electromagnet moved faster than 2 inch/s.

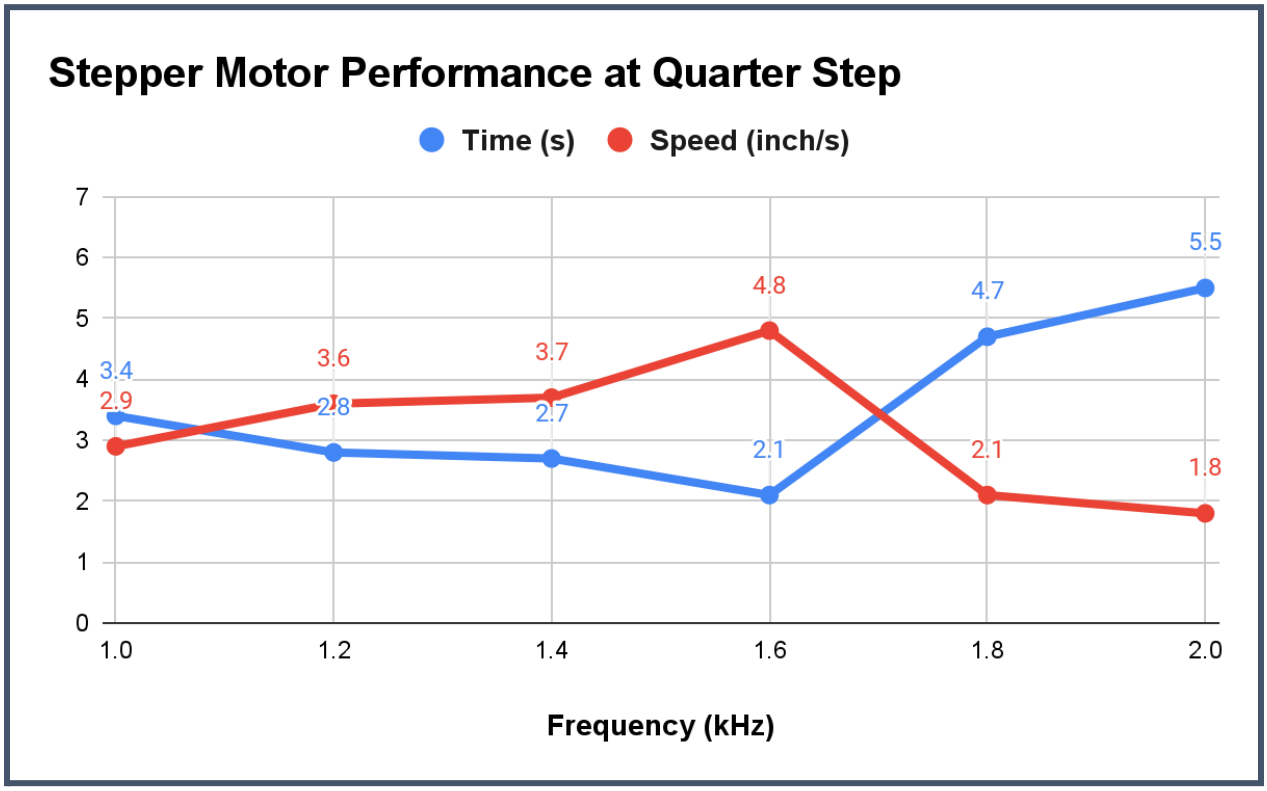

Figure 2: Stepper motor performance at quarter step

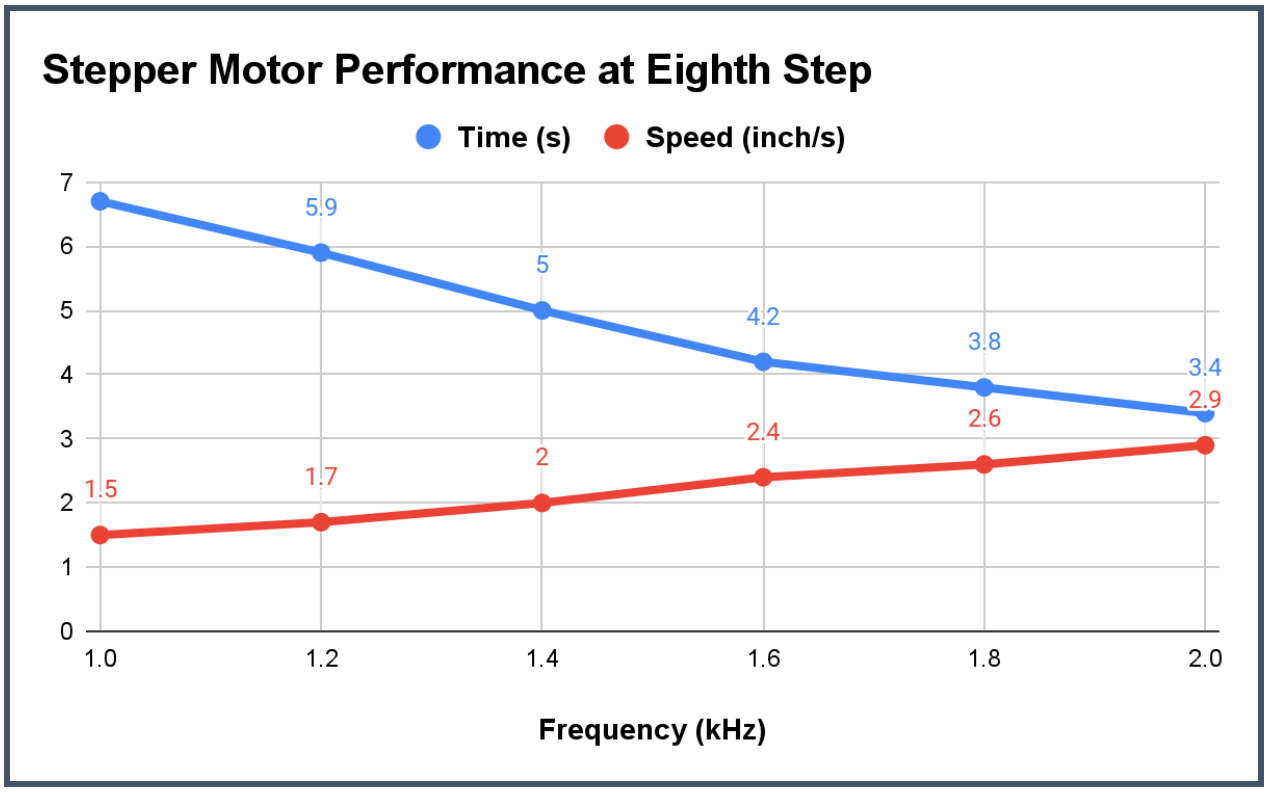

Figure 3: Stepper motor performance at eighth step

3.3.2 Electromagnet

The Electromagnet was originally meant to be controlled by a suitable MOSFET that could sustain the 12VDC when the electromagnet was off, and carry the 600mA that the electromagnet needed when it was on. Unfortunately, without the ability to program the microcontroller, this idea was abandoned for the purposes of the demonstration and was replaced with a button. However, the biggest verifications that needed to be tested were that the strength of the field through the board top was enough to move pieces, but also that it would not interfere with other pieces on the board. We were able to verify that the electromagnet could indeed reliably move pieces across the board, but due to a complication with the modification of the board top by the ECE Machine Shop to satisfy the implementation of the Reed Switches, a small air gap was created in some places of the board, and this would occasionally cause the chess piece to be left behind by the electromagnet.

4. Costs

The total costs of the project are composed of two parts: the parts cost and the labor cost. The labor cost is composed of the development cost and the machine shop cost.

Table 1: Total Costs

Part	Actual Cost ($)
Parts Cost	$92.61
Labor cost	$43687.50
Total	**$43780.11**

4.1 Parts

Table 2 shows the bill of material (BOM) of our chess board. The total cost of all the parts we ordered is $92.61.

Table 2: Parts Costs

Part	Manufacturer	Retail Cost ($)	Bulk Purchase Cost per Unit ($)	Actual Summed Cost ($)
ROB-12779 Easydriver	SparkFun Electronics	$16.95	$15.53	$31.06
Cylindrical Magnet	Radial Magnets, Inc.	$2.99	$1.89	$1.89
Qwiic Alphanumeric Display	SparkFun Electronics	$9.95	$8.96	$17.92
MSP430F6736 (Microcontroller)	Texas Instruments	$12.97	$7.84	$7.84
SN74AHCT594 (Shift Register)	Texas Instruments	$1.24	$0.80	$2.40
MIC94051YM4-TR (P-Channel MOSFET)	Microchip Technology	$0.67	$0.58	$2.32
MMBT3904-TP (NPN BJT)	Micro Commercial Co.	$v0.13	$0.08	$2.52
Reed Switch	Standex Electronics Inc.	$0.55	$0.38	$26.66
Total				**$92.61**

4.2 Labor

Our ideal salary is estimated at $25 per hour. We spend 15 hours per week and 15 weeks across the semester working on the project. As a result, the total labor cost of the 3 team members is $42187. As for the machine shop, the estimated labor cost is $1500.

$25 per hour x 15 hours per week x 15 weeks x 2.5 = $42187

Table 3: Labor Costs

Labor Type	Actual Cost ($)
Development cost	$ 42187.50
Machine Shop Cost	$1500.00
Total	**$43687.50**

5. Conclusion

As this project draws to a close, there are many lessons the group will take with them as they continue their ever-lasting engineering career and education. This section will highlight some of our biggest accomplishments in this project, and what we learned to do right. Additionally, the conclusion will convey some of the challenges faced and shortcomings of our ambitious project, as well as the ethical considerations of the Blitz Board. Finally, some recommendations of stretch goals and other features we would have liked to implement if more time was found are given at the end of the report.

5.1 Accomplishments

Even though we did not get the microcontroller to function properly, we had complete physical construction of the chess board and all other hardware components in place. Once the microcontroller is ready, the hardware can be connected to the PCB. Furthermore, we hard-coded some chess moves for pawns, bishops, and knights, and the stepper motors were able to move the chess piece to the designated location precisely at an adequate speed. Also, the electromagnet can generate a strong enough magnetic field to move the chess piece. It can be turned on to pull chess pieces as the movement begins and turned off when the movement is complete.

5.2 Uncertainties

One of the major uncertainties was that the microcontroller wasn't functioning properly at first. It prevented us from implementing other features such as the chess engine, chess clock, and the multiplexing of reed switch signals. The other uncertainty is the physical limitation. The electromagnet won't be strong enough to move the chess piece if the board top is thick enough to embed the LEDs. If the board top is thin enough for the electromagnet to move the chess piece, the LEDs won't be able to fit. As a result, we have to discard the LEDs.

5.3 Ethical considerations

Our project has few ethical aspects to it. It is merely a board meant for playing a game of chess. However, games can often be taken too far, or the boards they are played on incite annoyance and anger. While we consider our project to be safe to play with and use, it may not be best for mental health and safety. For users that struggle with interaction, loneliness, depression, or other mental health issues, this board may not be healthy to play on as it lacks human interaction and cannot provide emotional support like a true human companion and chess mate can provide.

Additionally, it is our responsibility to provide users with a board that works and will not cause issues to upset the user, and we are aware of and adhere to point 3 of the IEEE Code of Ethics: " to avoid real or perceived conflicts of interest whenever possible…" [2]. This would ultimately hurt the development of and sale of this board were it to hit the free market. Ultimately though, the goal is to provide the average chess player with a chance to challenge themselves against the processing power of computers in a physical location in their home that is both safe and fun to use when you lack the ability to play with another.

5.4 Future work

Avenues for further development of the project include improvements to the speed, precision, and reliability of the motors and electromagnets moving pieces. Fully enclosing the board and making it lighter would improve safety and desirability as a consumer product. Further development of the control hardware and software holds the potential of automatic setup for a game, choice in opponent difficulty, online games on the Blitz Board, a user interface via smartphone, and voice control of piece movement for accessibility.

References

[1] B. Schmalz, “Easy Driver Stepper Motor Driver,” *SchmalzHaus.com Brian Schmalz Homepage*. [Online]. Available: https://www.schmalzhaus.com/EasyDriver/. [Accessed: 03-May-2023].

[2] “IEEE code of Ethics,” *IEEE*. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 09-Feb-2023].

Appendix A Requirement and Verification Table

Table 4: Converter Requirements and Verifications

Requirements	Verification Methods

• The power unit is able to provide 12V to the motor and 5V to everything else
• The ripple of the converter's output voltage stays within 5% of the rated voltage
  ○ 12V: 11.4V~12.6V
  ○ 5V: 4.75V~5.25V
	• Probe the voltage output of the power unit and converter with a multimeter or an oscilloscope to make sure that it is within the rated range

Table 5: Microcontroller Requirements and Verifications

Requirements	Verification Methods
• The microcontroller can be programmed and debugged via a JTAG connection	• When connected via JTAG to a computer running TI CCS IDE, the IDE can upload a basic blink program, step, and throws no errors regarding the connection
• The microcontroller is able to identify that a chess piece has been moved from one square to another	• When a piece is moved between squares, interpretation of sensor data should allow the microcontroller to register that a piece is absent from one square and present in another
• The microcontroller is able to translate the response of the Stockfish chess engine into actions that are executed by motor and electromagnets	• An example move of a queen taking a rook laterally, longitudinally, or diagonally from the other side of the board can be translated into servo motor and electromagnet control • The example move can be executed with the pieces in the intended ending positions.
• The microcontroller is able to control the chess clock accurately	• The counting down of the chess clock matches that of a smartphone clock • The switch being actuated by the user or computer following a valid move changes which time is counting down

Table 6: Electromagnet System Requirements and Verifications

Requirements	Verification Methods
• The motors are able to move precisely and quickly as specified by the controller, and return to the center of the board when not in use.	• Orient the two-axis motor system such that the electromagnet begins at a specified starting point, such as the center of the board • Give the motors a set of instructions, (such as move up two spaces, to the left three spaces, etc.) • Ensure that the set of instructions should have the motor end up back at the starting point, and measure the variation from the start point and end point location using distance as a metric. • If all steps are equal by magnitude of angle, then the electromagnet should return to the exact starting point with little error. Steps should be taken to reduce this error
• The electromagnet shall produce enough magnetic field to reliably move pieces across the board despite variation in mass of the piece or position.	• The electromagnet will begin to move a piece at a rated speed. This speed will be determined by the speed needed to be faster than other chess boards • If the piece fails to move completely though the end of the move, the supplied current through the electromagnet will be increased. • This can be repeated until a piece is able to make the move.
• The electromagnet shall not interfere with other pieces on the path when moving a chess piece from one place to another	• Test whether the electromagnet will attract nearby pieces when moving a chess piece to its destination • Determine the minimal distance between the electromagnet and chess pieces
• The motors shall run within specified, rectangular bounds of 18” by 22” with respect to the center of the board	• The number of steps taken will be processed by the control unit carefully, and the electromagnet and motor unit will always return to the same spot • With the piece movement unit always returning to the same spot, bounds can be

	set for how far the unit moves in any one direction. ● The number of steps in a given direction can be counted by the processing unit, and can be checked with an upper bound limit before executing a move to ensure motors do not drive too far off the rails.

Table 7: Chess Clock and Interface Buttons Requirements and Verifications

Requirements	Verification Methods
● The LEDs in the table top should be able to denote the most previous position of a piece as well as its current position after either party makes a move.	● Pieces should not get left behind by the electromagnet, the piece should stop at the location it is being sent to ● The electromagnet should not interfere with pieces that aren't being moved, thus it should only move one piece at a time for normal moves ● During a “take”, the electromagnet will use the piece performing the move to push the “taken” piece out of the way. This is the only time that movement of a piece with the electromagnet should affect another piece ● After the “taken” piece is pushed away and the piece performing the take is placed, the electromagnet should be able to move the piece off the board.
● The hall effect sensors should be able to reliably detect whether or not the magnets used in the chess pieces are present above them	● With all sensors connected to the rest of the system and no magnets on the board, the system will register a voltage that doesn't change by more than 10% ● For each sensor, one part of a magnet in a chess piece being above one part of the sensor registers a range of voltages that doesn't overlap with that of when there are no magnets present
● The HEX displays will reliably keep time	● Set the HEX displays with specified amount of time (~3 minutes) ● Ensure the HEX displays now read 3 minutes each. ● Allow time to pass on either clock

	• Using a separate stopwatch or clock, record the time passed and then stop the clock using the lever switch • Ensure that the clock stops at the press of the switch • Further, ensure the time passed is represented accurately by the HEX displays. If 1 minute has passed, then the time left on the clock should have decreased by one minute. • This can be verified by an number of different play times, down to the seconds
• The intended numbers are displayed and readable	• The microcontroller can control which LED segments are lit up, and the lit segments can be distinguished from non-lit segments by a person indoors
• The LEDs can communicate when an illegal move has been made by lighting up the entire board with red lights and stopping flipping the clock back to the player until the move is corrected.	• To test this, we simply take a player piece and move it to a spot it is not able to move • When this move is made ensure the lights are activated. • Additionally, before correcting the move, ensure the board keeps the clock in the players control and does not run the computer's clock during this move • Once the move is corrected, verified, and the clock button is pushed to signal it is the computer's turn, deactivate all lights, and resume play.
• The solenoid can actuate the switch with a 5V pulse that lasts less than 500 ms	• Out of 20 5V DC pulses sent to the solenoid, all times the switch was actuated by the solenoid

Appendix B Arduino Code

Combined_Motor_Driveer.ino

```
// Include the Arduino Stepper and AH_EasyDriver Library
#include <Stepper.h>
#include <AH_EasyDriver.h>
//#include <AccelStepper.h>

//Declare pin functions on RedBoard
//#define EM 13

// Declare variables
char user_input;
const int stepsPerRevolution = 400;
const int step = 200;
const int step2 = 1375;
const int RES = 80; // RES -> RESOLUTION per full revolve

// Create Stepper library instance for L298N
Stepper stepper1(stepsPerRevolution, 8, 9, 10, 11);
// Create AH_EasyDriver library instance for Easy Driver
// AH_EasyDriver(int RES, int DIR, int STEP, int MS1, int MS2, int SLP, int ENABLE, int RST)
AH_EasyDriver stepper2(RES, 3, 2, 4, 5, 6, 7, 12);

void setup()
{

  //pinMode(EM, OUTPUT);
  //resetEDPins(); //Set step, direction, microstep and enable pins to default states

  // Setup L298N
  stepper1.setSpeed(52.5); // Set the speed at 52.5 rpm
  // Setup Easy Driver
  stepper2.sleepOFF(); // Set Sleep mode OFF
  stepper2.resetDriver();
  stepper2.enableDriver();
  stepper2.setMicrostepping(3); // MODE 3 -> 1/8 microstep
  //stepper2.setSpeedRPM(52.5); // RPM , rotations per minute
  stepper2.setSpeedHz(1400); // Hz, steps per second
  // Initialize the serial port
  Serial.begin(9600);
  Serial.println("Begin motor control");
```

Combined_Motor_Driveer.ino

```
  Serial.begin(9600);
  Serial.println("Begin motor control");
  //Print function list for user selection
  Serial.println("Enter the moving piece:");
  Serial.println();
}

// Main loop
void loop() {
  while (Serial.available()) {
    user_input = Serial.read(); // Read user input and trigger appropriate function
    if (user_input == '1')
    {
      RookForwardStep();
    }
    else if (user_input == '2')
    {
      RookBackwardStep();
    }
    else if (user_input == '3')
    {
      RookRightStep();
    }
    else if (user_input == '4')
    {
      RookLeftStep();
    }
    else if (user_input == '5')
    {
      BishopForwardRightStep();
    }
    else if (user_input == '6')
    {
      BishopBackwardLeftStep();
    }
    else if (user_input == '7')
    {
      BishopForwardLeftStep();
    }
```

Combined_Motor_Driveer.ino

```
    else if (user_input == '7')
    {
      BishopForwardLeftStep();
    }
    else if (user_input == '8')
    {
      BishopBackwardRightStep();
    }
    else if (user_input == '9')
    {
      KnightForwardForwardRightStep();
    }
    else if (user_input == '0')
    {
      KnightBackwardBackwardLeftStep();
    }
    /*else if (user_input == '9')
    {
      PieceTakingStep();
    }*/
    else
    {
      Serial.println("Waiting for next move");
      Serial.println();
    }
  }
}

/*
// Rook Moving Forward with Electromagnet function
void MagnetStep()
{
  Serial.println("Rook moves 2 blocks forward");
  digitalWrite(EM, LOW); // Turn on the electromagnet
  stepper1.step(-step*2); // Positive -> Backward
  digitalWrite(EM, HIGH); // Turn off the electromagnet
  Serial.println("Enter next move");
  Serial.println();
}
```

Combined_Motor_Driveer.ino

```
*/

// Rook Moving Forward function
void RookForwardStep()
{
  Serial.println("Rook moves 2 blocks forward");
  //digitalWrite(EM, LOW); // Turn on the electromagnet
  stepper1.step(-step*1); // Positive -> Backward
  //digitalWrite(EM, HIGH); // Turn off the electromagnet
  Serial.println("Enter next move");
  Serial.println();
}

// Rook Moving Backward function
void RookBackwardStep()
{
  Serial.println("Rook moves 2 blocks backward");
  stepper1.step(step*1); // Positive -> Backward
  Serial.println("Enter next move");
  Serial.println();
}

// Rook Moving Right function
void RookRightStep()
{
  Serial.println("Rook moves 2 blocks right");
  stepper2.move(step2*1); // Positive -> Right
  Serial.println("Enter next move");
  Serial.println();
}

// Rook Moving Left function
void RookLeftStep()
{
  Serial.println("Rook moves 2 blocks left");
  stepper2.move(-step2*1); // Positive -> Right
  Serial.println("Enter next move");
  Serial.println();
}
```

Combined_Motor_Driveer.ino

```
}

// Bishop moving forward-right function
void BishopForwardRightStep()
{
  Serial.println("Bishop moves 2 blocks forward-right diagonally");
  stepper1.step(-step*2); // Positive -> Backward
  stepper2.move(step2*2); // Positive -> Right
  Serial.println("Enter next move");
  Serial.println();
}

// Bishop moving backward-left function
void BishopBackwardLeftStep()
{
  Serial.println("Bishop moves 2 blocks backward-left diagonally");
  stepper1.step(step*2); // Positive -> Backward
  stepper2.move(-step2*2); // Positive -> Right
  Serial.println("Enter next move");
  Serial.println();
}

// Bishop moving forward-left function
void BishopForwardLeftStep()
{
  Serial.println("Bishop moves 2 blocks forward-left diagonally");
  stepper1.step(-step*2); // Positive -> Backward
  stepper2.move(-step2*2); // Positive -> Right
  Serial.println("Enter next move");
  Serial.println();
}

// Bishop moving backward-right function
void BishopBackwardRightStep()
{
  Serial.println("Bishop moves 2 blocks backwardward-right diagonally");
  stepper1.step(step*2); // Positive -> Backward
  stepper2.move(step2*2); // Positive -> Right
  Serial.println("Enter next move");
```

Combined_Motor_Driveer.ino

```
// Bishop moving backward-right function
void BishopBackwardRightStep()
{
  Serial.println("Bishop moves 2 blocks backwardward-right diagonally");
  stepper1.step(step*2); // Positive -> Backward
  stepper2.move(step2*2); // Positive -> Right
  Serial.println("Enter next move");
  Serial.println();
}

// Knight moving forward-forward-right function
void KnightForwardForwardRightStep()
{
  Serial.println("Knight moves 2 blocks forward and 1 block right");
  stepper1.step(-step*2); // Positive -> Backward
  stepper2.move(step2*1); // Positive -> Right
  Serial.println("Enter next move");
  Serial.println();
}

// Knight moving backward-backward-left function
void KnightBackwardBackwardLeftStep()
{
  Serial.println("Knight moves 2 blocks backward and 1 block left");
  stepper1.step(step*2); // Positive -> Backward
  stepper2.move(-step2*1); // Positive -> Right
  Serial.println("Enter next move");
  Serial.println();
}

// Piece taking function
void PieceTakingStep()
{
  Serial.println("Knight takes a piece then brings the piece to parking space");
  stepper1.step(-step*2); // Positive -> Backward
  stepper2.move(step2*1); // Positive -> Right
  delay(500);
  stepper2.move(-step2*4); // Positive -> Right
  Serial.println("Enter next move");
```

```
212  // Piece taking function
213  void PieceTakingStep()
214  {
215    Serial.println("Knight takes a piece then brings the piece to parking space");
216    stepper1.step(-step*2); // Positive -> Backward
217    stepper2.move(step2*1); // Positive -> Right
218    delay(500);
219    stepper2.move(-step2*4); // Positive -> Right
220    Serial.println("Enter next move");
221    Serial.println();
222  }
223
224  /*
225  //Reset LED switch pins to default states
226  void resetEDPins()
227  {
228    digitalWrite(EM, HIGH);
229  }
230  */
```