GROUP 4

Final Document

Interactive Automated Chess Game

Brett Rankin, Paul Conboy, Samantha Lickteig, & Stephen Bryant

12/3/2012

Table of Contents

Executive Summary 1
1 Section 1 Definition 2
1.1 Goals and Objectives 2
1.2 Project Description 2
1.3 Features and Functions 2
1.4 Design Considerations and Assumptions 2
1.5 Design Requirements and Specifications 3
1.6 Physical Chessboard and Pieces 4
2 Section 2 Research 6
2.1 Physical Chessboard 6
2.2 Microcontroller 6
2.2.1 Requirements 6
2.2.2 Researched Microcontrollers 7
2.3 Software 9
2.3.1 Chess Module 9
2.3.2 Engine 9
2.3.3 Chess Communications Protocols 9
2.3.4 Board Representation 10
2.3.5 Move Generation 10
2.3.6 Chess AI 11
2.3.7 Search 11
2.3.8 Evaluation 11
2.3.9 Machine Learning 14
2.3.10 Evaluating Existing Chess Engines 14
2.3.11 I/O Module 17
2.3.12 CEM-1203 Buzzer 17
2.4 User Interface 17
2.4.1 HMI 17
2.4.2 LCD Controller 20
2.5 Reading the Board State 22
2.5.1 Visual 22
2.5.2 Hall Effect Sensors 23
2.6 LED Setup and Controllers 25

2.6.1 LED Setup 25
2.6.2 LED Controller 26
2.7 Magnets (Pieces) 27
2.8 Mechanical Assembly 28
2.8.1 Similar Applications Studied 28
2.8.2 Available Parts Studied 29
2.8.3 Wire Management on a Machine with Moving Assemblies 31
2.9 Motors and Motor Control 31
2.9.1 Motors and Actuators 31
2.9.2 Position Feedback 34
2.9.3 Motor Control 35
2.9.4 Stepper Motors 36
2.9.5 Servomotors 36
2.10 Gripper/Claw 37
2.11 Audio Design 38
2.11.1 Audio System 38
2.11.2 Magnetic Buzzer 38
2.11.3 Wav File Player 38
2.11.4 Comparison of Audio Devices 39
2.12 Power Supply 40
3 Section 3 Design 45
3.1 Physical Chessboard 45
3.2 Microcontroller 47
3.2.1 Requirements 47
3.2.2 Design 47
3.2.3 Programming 48
3.3 Software 48
3.3.1 Requirements 48
3.3.2 Software Architecture 48
3.3.3 Main Module 50
3.3.4 Player’s Move 52
3.3.5 AI Move 53
3.3.6 Chess Module 54
3.3.7 I/O Module 57

3.3.8 Error Codes 60
3.4 User Interface 60
3.4.1 LCD Interfacing with the AI 67
3.4.2 Human Input Panel Interfacing with the AI 67
3.5 Hall Effect Sensors 68
3.5.1 Hall Effect Sensor Interfacing with the AI 71
3.6 LED Setup and Controllers 71
3.6.1 LED Control 77
3.7 Magnets (Pieces) 77
3.8 Mechanical Assembly 78
3.9 Motor Control 81
3.9.1 Gear Selection 84
3.9.2 Motor Control 84
3.9.3 Control Circuits for Motion Control 86
3.10 Picker/Claw 87
3.11 Audio Design 89
3.11.1 Audio Interfacing 89
3.12 Power Supply 91
4 Section 4 Prototype and Testing 91
4.1 Build and Implementation Strategy 91
4.1.1 Theory of Operation 95
4.2 Acceptance Testing 111
4.2.1 I/O Checkout 113
4.2.2 Mechanical Assembly 114
4.2.3 Human Machine Interface 115
4.2.4 Motor and Axes Control 118
4.2.5 LED Lighting Matrix 119
4.2.6 Hall Effect Sensor Grid 125
4.2.7 Picker/Claw 126
4.2.8 Chess Module 126
4.2.9 Main Module for AI 127
5 Section 5 Budget and Time Management 129
5.1 Pricing Breakdown 129
5.2 Sponsorship and Guidance 130

5.3 Time Allocation 131
6 Section 6 Appendix 135
6.1 Acronyms 135
6.2 References 136
6.3 Diagrams and Other Supplemental Material 136

Executive Summary

The purpose of this endeavor is for senior undergraduate level Electrical and Computer Engineering students to create a project that will not only give them real world experience into the process of building a project from its inception until its inevitable completion, but to do so in a manner that is challenging yet ultimately accomplishable; the Interactive Automated Chess Set is exactly that.

The Interactive Automated Chess Set combines not only Electrical and Computer Engineering skills, but those of Mechanical Engineering as well. Instead of the standard pieces being moved by each respective player physically all piece movement will be entered into one of the two HMI terminals, complete with LCD screens, and a claw suspended above the chessboard will be responsible for all piece movement. Under each square on the grid a LCD light and a Hall Effect sensor, to create the grid and keep track of each piece respectively, will reside; the information read in from the Hall Effect sensors will be checked against the computer's AI which will also be keeping track of all piece movement as well so that any error that may occur, or interference from an external source, will not permanently upset gameplay. The overall goal of the Interactive Automated Chess Set is to allow for three different modes of play, whichever the user prefers, as well as a unique spin on one of the most ancient strategy games still in existence today.

As a group we looked extensively at other examples of automated chess sets, like the one found on the lets make robots website http://letsmakerobots.com/node/26979, to see if the project was viable. When it was inevitably determined that that would become our project we set to evaluating each example we had viewed. This proved to be an invaluable informational treasure trove in the fundamental design of the project including an unforeseen difficulty regarding the use of two servomotors simultaneously when moving the belt that would control the claw's horizontal movement, requiring the claw to not only be suspended above the pieces but with enough clearance to lift the tallest piece and move it over another piece of the same height without interfering with any of the other pieces, and finally the 2:1 ratio rule.

If everything stated above is combined with the myriad of possible features that can be included in the Interactive Automated Chess Set, like audio speakers that can say any message that can be recorded from a basic "check" warning to encouraging a player to do better or LED lights that do a lighting show comparable to a disco whenever a piece is taken out of play, you are left with a project that has infinite potential that is only limited by the group's imagination and time.

1 Section 1 Definition

1.1 Goals and Objectives

The idea is to create a portable interactive chessboard where gameplay will be fully automated using a crane suspended above the chessboard that is responsible for moving all of the pieces no matter if the game is person versus computer or person versus person; any piece that is removed from play will be placed on a specified grid to the side for each player's respective color. It will be powered by a computer power supply.

1.2 Project Description

The actual board was constructed out of Duraplex (a similar material to Plexiglas) which is frosted so that the light produced by the LEDs underneath the board will be dispersed over the entire square. Attached on opposite ends of the board are two displays for user input and a crane is suspended above the board, attached to an A frame that slides underneath the board, and was constructed such that it is able to move over both the X and Y axis using gears and motors as needed; the crane is used to move all of the pieces.

1.3 Features and Functions

The LED lights inside of the board serve to light up specific squares for various purposes: showing where a user can place a specified piece, where a specific piece is going, and for a general ambience effect.

Each user input has four command buttons as well as an LCD screen; the four buttons are FROM (what piece is being moved), TO (where the selected piece is being moved to), GO (once the FROM and TO are specified, GO finalizes the selection and the claw moves the specified piece; cannot be undone once GO is pressed), and CLEAR (to clear the player's selection so it can be changed; can only utilized before GO is pressed). In the event the machine fails to properly relocate a piece, the LEDs and two HMI displays can prompt the user to move the piece to the correct destination, and then clear the fault to resume game play.

1.4 Design Considerations and Assumptions

The automated mechanical system presented the most risk and challenge. We fortunately were able to find a Mechanical Engineer mentor for guidance on this critical portion of the project and as such we have had far more success than we would have otherwise. One of our biggest challenges was being able to reliably pick up pieces without disturbing any other pieces and then moving the piece to its destination, again without disturbing the other pieces. The picker had to be designed with the tradeoff of being able to re-align a piece that is not optimally centered in the cell, or the optimal parking tolerance of the picker, all verses not

disturbing the other pieces on the board. When a chess piece must be removed from play it is relocated to the discard zone on either side of the chess grid.

Another challenge was finding pieces that our claw would be able to pick up and after viewing a number of photos of chess pieces we discovered that, fortunately, there is no one standard size for chess pieces. We needed pieces that could be reliably picked up and relocated. The Knight in particular required a scaled down horse head that won't get stuck in the picker or be too slim for the claw to grab. All pieces will need a ridged surface that the picker can consistently clamp and lift.

From a practical stand point there is a need to limit I/O and wiring to both the under board LEDs and Hall Effect sensors. If every point was hard wired we would need two-hundred and fifty six wires plus the DC common buss. Two-hundred and fifty seven wires would quickly become a massive wire harness. The solution we chose was to build a multiplexed grid under the board. Each row will have its own RGB buss and each column will have its own column select buss. A similar arrangement was chosen for the Hall Effect matrix. Each of the two HMIs have a small 20x4 line LCD display. With the displays, the multi-color RGB LEDs, and the Hall Effect sensors all in use, error detection should be possible. Our worst case failure mode is if the automated picker fails to move a piece from its source to its destination. With our design we can detect the fault and communicate it to the users and the users could then reposition the piece at the correct location, and the game could continue to be played.

Our versatile concept lends itself well to additional features either being added or removed from the final product in the future as time and resources allow. For example an automated reset feature could be added, where all of the pieces in the discard zone, or relocated pieces on the playing surface can be automatically moved back to the start configuration. This feature simply becomes software and testing add on. The LEDs could give the player hints on possible user moves, again it comes down to programing and testing.

1.5 Design Requirements and Specifications

After much deliberation, research, and trial and error we as a group came up with the following list of specific parts and pieces that were required in order to get the Interactive Automated Chess Set off the ground; that compiled list is as follows:

- 1 Microcontroller for monitoring and controlling under board LED lighting, motion control commands, and feedback for X-Y axis and picker controls, interfacing with the user HMIs, and for running the AI for the computer player
- 192 LED lights for the squares under the chessboard (64 each for the red, green, and blue lights respectively)

- 32 chess pieces with a magnet attached to the bottom (came standard on the pieces we chose)
- 3 motor controllers (X,Y, Z Motion Axes)
- 3 Stepper motors for X, Y, & Z axis motion control.
- 1 LED controller
- 1 Gantry Assembly
- 1 Picker/Claw Assembly
    - Each tip covered with memory foam
    - Approximate 88-90% accuracy with each grab and placement of a single piece
- Playing surface is covered with a defused plastic material (Duraplex) for an ideal LED lighting look.
- Power supply
    - 120 VAC 60 Hz input purchased power supply with no battery operated functionality. Output voltages are 3.3V, 5V, and 12V. We insured the power supplies were mounted inside a suitable enclosure to comply with NFPA 70E Electrical Safety Standards.
- 2 User Interface Panels (HMIs) switch arrays with a 20x4 line LCD Display
- 1 Chessboard with 2 discard grids
    - Playing grid adheres to standard dimensions (12”x12”) with each square 2/3” x 2/3” with 64 total squares (in an 8x8 configuration) while each discard grid will be 1 1/2” x 12” with 16 total squares (2 x 8 configuration).
- Logic of chess AI
- Total weight: Less than 100 lbs.
- Custom designed PCBs are mounted so they are visible to the public, but still properly protected within the shell of the chessboard
    - Custom PCBs to include I/O and Motor Control

1.6 Physical Chessboard and Pieces

The chessboard has taken the classic 8x8 form as shown below in Figures 1.6-1 and Figure 1.6-2 with the two discard sections (the “graveyard”) where the pieces that are no longer in play will reside until the game is finished though there is no grid pattern on the physical board to indicate this as shown in Figure 1.6-2 versus the one shown in Figure 1.6-1. HMI 1 and HMI 2 refer to the two human interfaces that are attached to the board for the user’s to input their piece selection and movement. Each grid square has its own unique alphanumeric designation using A through H and 1 through 8.

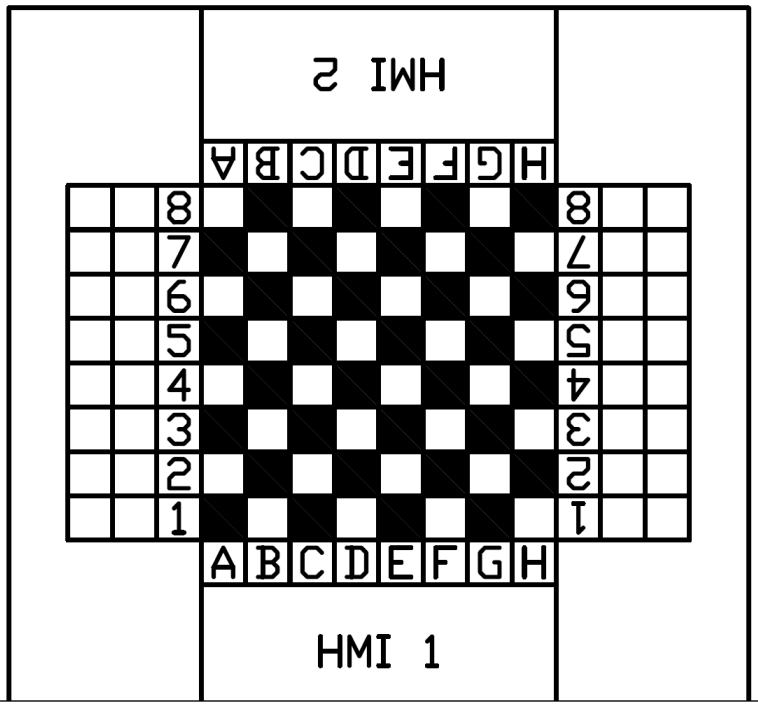

Figure 1.6-1: The chessboard that will be used for the Interactive Automated Chess Game. This picture was drawn using AutoCAD educational software.

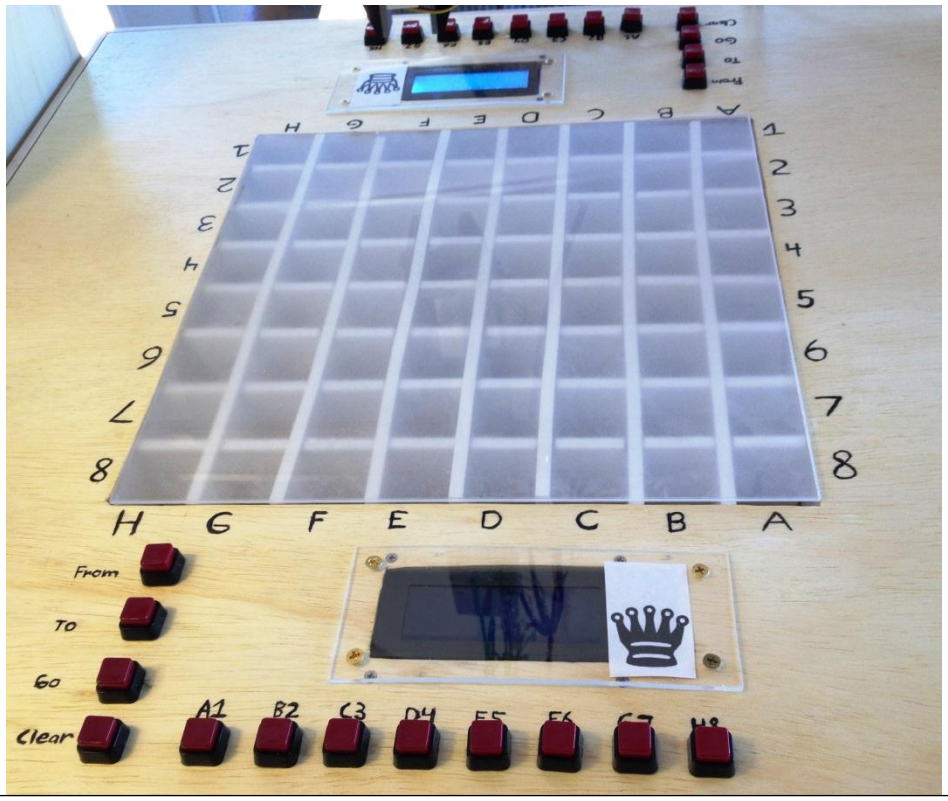

Figure 1.6-2: The chessboard that is used for the Interactive Automated Chess Game as shown from the perspective of HMI 2.

2 Section 2 Research

2.1 Physical Chessboard

Normally this would be a simple option of what off the shelf chessboard would we like to purchase? Our design requirements dictate two critical features. The first is that LED light must be able to project onto the playing surface from underneath the board. Second the Hall Effect sensors must be able to sense the magnets embedded into the chess pieces through the playing surface. Considering both of the above requirements, we are basically unable to find a simple board to purchase for a reasonable price and we were forced to design a custom board. We needed to select some form of semitransparent plastic material for our playing surface. This material had to allow the light wave to penetrate through, and ideally defuse the light evenly across the cell. One idea was that we could select two distinct colors of plastic material and cut them into 1.5 by 1.5 inch squares then integrate them with the checker board pattern. Another idea was if we could build a vertical blinder grid under the board so that the LEDs can be used to illuminate the two basic color pattern so that only one piece of plastic would be required. The sliding A frame structure also needed to pass under the board without damaging any of the wiring under the board. The whole chessboard assembly must bridge over the A frame structure base. One of the plastics options considered was the Lumicore® line from Professional Plastics INC as well as the Duraplex® line we found at Lowes. The Lumicore® line offers highly attractive patterns embedded into the material as well as nice translucent white sheets but the cost and availability of the Duraplex® made it the better choice. The board itself is able to enclose the lighting effects from bleeding over to adjacent cells or from projecting down from under the playing surface while the space under the board is housing the Y axis slide rails, Y axis base plate, the Y axis stepper motor, and connect to the wire management system. Under the slide rails space should be utilized for the electronics.

2.2 Microcontroller

2.2.1 Requirements

The biggest factor in choosing a microcontroller was getting a MC that could reliably hold a representation of a chess game in memory. To represent the chessboard alone in memory, we need a double word for each piece (8 bytes * 32 pieces = 128B) of memory; this doesn't include pieces in the graveyard. More memory will be taken up by our search algorithm. We have to recursively call evaluation methods during application with every level deeper we go increasing our memory usage exponentially. We also have to keep in mind the amount of I/O pins we need. In the case where we wouldn't have enough pins on our microcontroller, we need to make sure that the microcontrollers are capable of

communicating with each other. Another problem we have run into is the logistics of experimenting with a surface mounted microcontroller. In order to begin our research process for a microcontroller, we created a table with tentative pin counts for our various planed subsystems (Table 2.2.1-1 below).

The Hall Effect sensors of our system require 7 pins to operate, including a multiplexer and demultiplexer. Our LED matrix control requires 1 pin for each color. The controller we are looking at uses serial communication. Our stepper motors require 3 pins each, and we have 3 motors. To sense when our crane system is at the “origin” point, we will need 2 pins. Our user input pads should take 8 pins for the number pad (which will be reused) and 4 pins for control buttons, including FROM, TO, GO, and CLEAR. The LCD displays require 6 pins each and the clamp (part of the claw mechanism) utilizes one pin to “activate” the closing apparatus.

Module	I/O Pins	Analog?	Part #
Hall effect Sensors	3 8-1 mux + 4 16-1 demux	No	~
LED Matrix Controller	1x3 (RGB)	No	MAX7219
3xStepper Motors	3x3	No	~
Home sensing	2	No	
2xUser Input Pads (reuse row and column selectors) (go is the power button)	2x12 (4 for FROM/TO/GO/CLEAR)	No	~
2xTwo Line LCD displays	2x6	No	Hitachi HD44780
Clamp Control (Solenoid, Closed sensor, Home Calibration)	1 + 3-4(sensor) +2 pins (home sensing for calibration)	yes	~

Table 2.2.1-1: Tentative pin counts for our various subsystems

2.2.2 Researched Microcontrollers

We began our research with the MSP430 since it is very accessible with the Launchpad starter kit from TI. It has ADC, a modest amount of flash memory and SRAM, and one timer. However, with its low I/O pin count and lack of I2C/UART ports it doesn't lend itself to being easily networked with other microcontrollers. We also looked at a couple PIC controllers from Microchip, and found a wide variety. Specifically, the PIC16F877A was a promising candidate. It contained a large amount of flash memory, I2C, UART, and 3 timers. However, it has

significantly less SRAM and EEProm than the controllers we later found from Atmel.

Atmel AVR microcontrollers have a lot of benefits, the first being that they are used as the basis of the ever-popular Arduino microcontroller boards and there is therefore a large user support community. There are many AVR libraries and support websites. We began our search with the ATMega328, a very popular microcontroller. It has an I2C port and can therefore be used in an I2C network. It has a decent amount of SRAM and flash memory, so it could hold a module of the application. The I/O pin count is low, but that is why we would use it in a network. The Atmel Xmega line of microcontrollers has much more power. With multiple I2C and UART ports, we need not worry about communication over a small network. The large amount of flash memory and SRAM means we would have more wiggle room to experiment with semi-advanced chess AI. It has more I/O pins than our estimated pin count, so we will have room to expand if needed. Our team has not worked with a surface mount technology like this, but that barrier can be overcome.

The Stellaris LM4F is TI's powerful surface mount microcontroller. It also contains multiple I2C and UART ports. It has more memory than the Atmel Xmega as well as a faster clock speed and the option of having up to 105 GPIOs. Overall, it is a very solid microcontroller that would surely be useful for our project. The evaluation board runs around $150; all of the microcontrollers researched above can be found summarized below in Table 2.2-1.

Specification	Atmel Xmega 128A1U	ATMega328	MSP430G2211	PIC16F877A	Stellaris LM4F Series
Pins	100	28	14	40	64/100/144
I/O	78	23	10	33	49/69/105
ADC	16 channel	8 channel	Slope	6 channel	24 channel
Mounting	Surface	Through Hole	Through Hole	Through hole	Surface
PWM	Yes	Yes	No	Yes	Yes
Flash Memory	128KB	32KB	2KB	14KB	256KB
EEProm	2KB	1KB	0	256B	2KB
SRAM	8KB	2KB	2KB	368B	32KB
Clock Speed	32 MHz	20 MHz	16 MHz	20 MHz	80 MHz
I2C	4	1	0	1	6
UART	8	1	0	1	8
Timers	8	3	1	3	12

Table 2.2-1: Different microcontroller specifications setup for easy comparison

2.3 Software

Our software system consists of two major modules: the chess module and the I/O module. The questions we must answer in our research are: How to implement a chess module that fits within the constraints of our microcontroller, whether or not to offload our AI processing to an external computer or server, and how to interface with our various software peripherals.

2.3.1 Chess Module

There is a large chess programming community on the internet, offering tutorials and advice for people writing the most basic engines and AI, to the most advanced engines and AI. One such website is http://chessprogramming.wikispaces.com. We have used this website as our main resource for programming the chess AI. It describes the theory and implementation of creating chess engines and chess AI from scratch. It also references chess engines of all levels of complexity: beginner, intermediate, and advanced this means that we do not necessarily have to create our own chess engine.

2.3.2 Engine

The software research process began with the chess engine. This is the system that maintains a representation of the chess pieces in memory as well as checks user input moves for validity. There are many methods and techniques discussed in the following sections; our goal is to create a set of data structures that can represent and validate our chess game with the amount of memory we have. Since our project is focused on the electromechanical aspects of the chess game, we also wanted to research the simplest way of implementing a chess engine. We needed to look at protocols to communicate with chess AI engines for the possibility of remote connection, but we also needed to research having the entire AI engine onboard in our microcontroller. We also needed to look into creating an interface to send commands to the chess motor controllers. And we hoped that everything has been programmed on one microcontroller so the interface will be part of the software package.

2.3.3 Chess Communications Protocols

If we are to offload the chess AI to a remote server or computer, we must have a way of communicating with the computer. There are many open communications protocols for chess engines to communicate with other programs including graphical user interfaces. The first protocol is called the Chess Engine Communication Protocol, or more simply the xboard or WinBoard protocol. This protocol is old and has been mostly replaced by the Universal Chess Interface protocol.

The second protocol is called the Universal Chess Interface (UCI). The entire system is designed to be a model view of the controller system, with the general user interface (GUI) (or in our case, physical chessboard) being the “view” module. However, it also keeps a model of the internal game state and delegates many engine control parameters to the view module, which is essentially not a model view controller (MVC).

2.3.4 Board Representation

The chessboard can be represented in a piece-centric or square-centric fashion. A piece-centric representation keeps lists, arrays, or sets of all pieces on the board while the information contained is the square they occupy. Since there are 64 squares, this amounts to a list of 32 double words, which is 128 bytes in memory. A square-centric board representation would involve storing a representation of the physical board in memory, with the associated information identifying each piece.

2.3.5 Move Generation

Move generation involves checking for legality. A legal move is defined as a move that does not leave the king in check. In order to check for legality, we must make sure that the move is not absolutely pinned by the king in its move direction. En passant requires horizontal pin tests of both pawns, which disappear from the same rank. Also, while castling, the rook cannot be pinned horizontally. Other than that a single function must be used to check the legality of moves for each piece type’s particular movement capabilities.

Generating sets of moves is highly dependent on the board representation. Move generators can generate pseudo-legal moves (might leave the king in check) and leave it to the make move function to check for validity, or they can generate legal moves. For debugging move generators, a function known as a Perft function can be created. This function generates moves for the current position and child positions to a certain depth. Counting the moves against a table of values will test accuracy. There exists a table (as shown in Table 2.3.5-1 below) of counts for the initial position of the game.

Depth	Nodes	Captures	Checks	Checkmates
1	20	0	0	0
2	400	0	0	0
3	8902	34	12	0
4	197281	1576	469	8
5	4685609	82719	27351	347
6	119060324	2812008	809099	10828

Table 2.3.5-1: Table of results from Perft funtion

2.3.6 Chess AI

We could offload the chess AI from the board to a computer and receive commands to a dumb microcontroller over a network. This would involve purchasing a module such as the WiFly GSX for wireless communication. This would allow us to use a prepackaged chess AI library on a server which has many advantages: it allows us to use professional level AI that is very competitive as well as allowing us to focus entirely on low level programming for our microcontroller. This would require us to learn how to create a network module for the microcontroller however.

Conversely we could keep the chess AI directly on the board; this has many advantages and disadvantages to consider. The chess AI must be programmed by our team entirely. AI can be very difficult to write in a low level language such as C and it could be even more difficult to write and debug on a microcontroller. There is also the potential of adding more memory constraints, since chess AI involves searching algorithms that take an exponential amount of memory as the lookahead increases. This means we would have to settle for a rudimentary AI that looks one or maybe two turns ahead. Our main programmer, Nick, is very interested in AI and is up to the challenge of creating a good AI in this manner.

2.3.7 Search

Searching involves looking forward at the possible move sequences of both players and evaluating the results. There are two different types of search: Type A and Type B search. Type A search is known as brute force, which involves looking at every possible move sequence. Type B search involves using a heuristic to only search important branches. Type A search is almost completely unmanageable in an application like ours, since the number of possibilities would quickly grow to a number we can't compute leaving Type B as the more logical choice. Most chess applications use an alpha-beta search to search the game tree in a depth-first manner. It is very efficient compared to the standard minimax and negamax algorithms.

2.3.8 Evaluation

Evaluation involves determining the relative value of a chess position i.e. the state of the board. We can begin the evaluation process by determining the point value of the pieces. One such scale for evaluating pieces and moves is called the centipawn scale. It is used in many chess AI systems and basically involves the relation 100 centipawns = 1 pawn. The standard valuation, called the 1/3/3/5/9 scale, is given in Table 2.3.8-1 below.

Piece	Queen	Rook	Knight	Bishop	Pawn
Valuation (centipawns)	900	500	300	300	100

Table 2.3.8-1: The standard 1/3/3/5/9 scale

There are many other valuations which include the time values of the pawns, notably including Jacon Saratt's valuation and Emanuel Lasker's valuation. Jacon Saratt's valuation is interesting because it takes into account the endgame value of pawns as shown below in Table 2.3.8-2. Emanuel Lasker is a chess world champion. His scale is further refined to depend on the proximity of pieces to the king in early game. For example, a bishop that begins close to the king is worth more than the one that begins closer to the queen; his scale is shown below in Table 2.3.8-3.

Piece	Queen	Rook	Knight	Bishop
Early game valuation (centipawns)	1,187	487	462	487
Endgame valuation (centipawns)	1,187	487	462	487

Table 2.3.8-2: Jacon Saratt's valuation

Piece	Queen	King rook	Rook pawn	Knight pawn
Valuation (centipawns)	1,100	700	50	125
Piece	Central pawn	Knight	Queen bishop	King bishop

Table 2.3.8-3: Emanuel Lasker's valuation

We could continue to research scores of valuation models: some from chess guides in the 1800s, some written by chess grandmasters, and some analytically derived from computer analysis. The value of pieces is obviously dependent on the situation. It's dependent on the time in the game as well as the location of the pieces. In order to optimize the decisions made by the AI, a proper scoring system is required. This is especially important in a system such as ours that won't have enough memory to support a large game tree. It should also be noted that a system as rigid as these cannot describe the entire situation of the game.

For example, there are openings which sacrifice a pawn for later advantage. However, most computerized systems make up for such limitations with sheer processing power over the human brain.

Once these piece values are defined, we can define the “material” value as the sum of piece values of each side. However, certain advantages and disadvantages can be created between pieces. An evaluation system should take into account the following factors (Table 2.3.8-4). In terms of our searching algorithms, the “material balance” term is incredibly important. It is equal to the difference between the material values of both sides. It is recommended that the material evaluation scores be stored in hash tables. Beyond scoring material value, mobility value must be taken into account as well. Mobility measures the amount of choices a player has while in a certain position. Higher mobility is preferable. It is usually calculated on a piece-by-piece basis by summing up the amount of legal moves the piece can make. A general equation for evaluating the move side is the following:

Eval = (materialScore + mobilityScore) * who2Move

Who2Move is +1 for the person being evaluated and -1 for the opponent.

Combination	Bonus/Penalty	Reason
Bishop pair	Bonus	Control two different color squares
Rook pair	Penalty	Redundant
Knight pair	Penalty	Two knights are less successful against the rook than any other pair of pieces
Rook pawns	Penalty	
Central pawns	Bonus	
Trade pieces	Bonus	Encourage winning side to trade pieces
No pawns	Penalty	Difficult to win endgame

Table 2.3.8-4: Material bonuses and penalties

2.3.9 Machine Learning

Machine learning can be incorporated into a chess AI as well. This would allow the program to change its behavior as more games are played. The three learning paradigms that can be used are supervised, unsupervised, and reinforcement learning. In our chess program, we could provide examples of good sequences of moves and examples of bad sequences of moves. We could give the program specific answers with which it can decide its own performance.

With unsupervised learning, which may not be very applicable in our case, the computer would be given sequences of moves and "cluster" them together into its own sets of classes of moves. With reinforcement learning, we could give certain rewards for taking pieces and advancing the game further. The agent would then move to maximize rewards. The agent would actually have to play many games to increase its "knowledge" of how to play. This kind of learning could be implemented alongside the traditional AI techniques we are using. If the computer does not yet know what to do in a situation, it can evaluate what it believes to be the best solution using a game tree search.

2.3.10 Evaluating Existing Chess Engines

Since nobody wants to reinvent the wheel, we considered porting many already-built chess software engines to our microcontroller program. This could present some issues since most implementations are meant to be run on a computer and are thus not very lightweight. Either way, looking at an already fully implemented system greatly simplifies the act of programming a new one.

2.3.10.1 TSCP

TSCP stands for "Tom Kerrigan's Simple Chess Program". It stands at 2,258 lines with mostly comments. It uses about 64K of RAM. Our Xmega has about 8K of RAM, so it can't be directly ported. It was written with clarity of reading in mind, so it may be optimizable. It is copyrighted and therefore we must obtain explicit permission from the owner if we are to use it. Otherwise, we would have to rewrite it. TSCP is multi file, and very complex. It has a board file, which is in charge of generating moves, representing the board, checking for attacks on squares, and testing whether the king is in check. It is worthy to note that it generates castling, en passant, and promotion moves for each piece. It does not use queen-only promotion. It has a module for reading openers from playbooks, which we could not port because there is no operating system to store text files in.

The evaluation module is very complex. It evaluates the worth of pieces based on their location on the board or the time period of the game. Unfortunately, it does this by storing 8x8 arrays of integers in memory. This is the kind of thing

that we have to seriously limit in our code. For search, it uses a recursive minimax search function known as negamax.

2.3.10.2 Micro-Max

Micro-Max is a great candidate for porting in our project. It comes in at 133 lines of portable C code. It has actually already been ported to AVRs such as our Xmega. The pieces on the board are encoded using 8 bits. The board is represented as a 1 dimensional array, rank after rank. The 4 lowest order bits indicate the rank, and the 4 highest order bits indicate the file. The ranks of the board have 16 squares. This is used to efficiently recognize moves that follow off the board. If the fourth bit of a rank/file location is a “1”, the move is invalid.

The move generation system implements En Passant, castling, and pawn promotion. However, it does not implement minor promotions. Therefore, pawns can only be promoted to queens. The search tree is based on Alpha-Beta Minimax and evaluation is based on the typical 1/3/3/5/9 pawn setup. The focus of this implementation is conciseness, not perfect play. The program uses a concept called iterative deepening for the game tree. Basically it involves doing progressively deeper searches by choosing the most likely path through shallow searches. The idea for this is that it will eliminate doing search paths that begin with ridiculously stupid sequences, such as a king going on the offensive. The recursive function has a maximum depth that can be set by the caller. Thus, we can set our own maximum depth to further reduce the memory usage of the game.

The interface to the program is rather simple. It reads in four characters which represent the move numbers. If we choose to implement this engine on the AVR, we will have to interface it to our I/O module in another way. A lot of our porting work has already been done for us. A programmer named Andre Adrian ported the MicroMax to the Atmel ATMega and AVR GCC. He sacrificed the hash table and added a Winboard driver. It defines a standard interface that can be called by our main module. The interface is discussed in Table 2.3.10.2-1and Table 2.3.10.2-2 below.

Function	Description
Side	Side to move
Move	Move input to or output from engine
PromPiece	Requested piece on promotion move
TimeLeft	Ms left to next time control
MovesLeft	Number of moves to play within TimeLeft
MaxDepth	Depth limit of search tree
Post	Debugging flag

Table 2.3.10.2-1: Global variables of the chess interface

Function	Description
InitEngine	Program start-up initialization
InitGame	Initialization to start a new game
Think	Think up move from current position and stores it in Move variable
DoMove	Perform the move in Move and toggle Side
ReadMove	Convert input move into engine format
PrintMove	Print Move on standard output
Legal	Check move for legality
ClearBoard	Make board empty
PutPiece	Put a piece on the board

Table 2.3.10.2-1: Functions of the chess interface

2.3.10.3 Faile

Faile is another open source chess engine. It is much stronger than the previous two engines. It can play as a master AI on moderate hardware. The source is small, clear, and clean. However, due to the complexity, it takes up much more program memory and RAM than the other two engines and is thus not very useful.

2.3.10.4 Comparison

We identified the most important factors to weigh when comparing engines as shown below in Table 2.3.10.4-1. The first is customizability and Portability. In order to use an engine on our project it has to be written in a clear manner that we can tackle in terms of porting to a microcontroller. Since having a chess engine is critical to the project, this is the first thing we must ensure. Computational resources are the next most important factor. Our microcontroller only has a small amount of memory. In all probability, we will be running a game tree that is very shallow. As such, we need to make sure that we don't attempt to port a program that has a large overhead.

Time cost is another important factor as well. This is closely related to customizability. If the application comes without sufficient documentation or the code is not clear enough, it will incur a great time cost to understand it. Since there are more important features to look to on this project, we must consider the time cost. Red tape is something we don't want to deal with in our project. If the code, such as TSCP, is not allowed for usage in the project, we must rewrite it from scratch. This will incur father time overhead. Lastly, the level of play can be used as a tie breaking factor. It really does not matter the level of play that our AI has as long as it works.

Engine	Customizability and Portability	Computational Resources	Time Cost	Red Tape	Level of Play
TSCP	Low	Medium	High	Copyrighted	Medium
Micro-Max	High	Low	Low	None	Low
Faile	Low	High	High	Open Source	High
Self-Written	High	Low	Medium	None	Low

Table 2.3.10.4-1: A comparison of chess engines

2.3.11 I/O Module

We researched ways of communicating with our I/O devices such as the LCD screens and stepper motors. This involved a lot of searching for AVR specific tutorials.

2.3.12 CEM-1203 Buzzer

Driving the CEM-1203 buzzer is very simple. We began by searching for online articles as we always have, and we found an article at the URL http://elasticsheep.com/2009/10/driving-a-buzzer/. It takes a Hz square wave to drive it. The highest frequency response is around 2000 Hz, but it can reliably be driven between 200 and 1800 Hz with at least 70 dB. The square wave should be controlled by bit banging at ½ duty cycle. In their example, they create a loop with a delay – toggle – delay – toggle sequence. Toggling with an output loop doesn't free up the processor very much, so we would have to find another way. Luckily, we will be able to drive the pulse width modulation with a hardware timer from our microcontroller.

2.4 User Interface

2.4.1 HMI

A Human Machine Interface (HMI) is necessary for a project such as this to allow for easy communication between the user and the chessboard. The three basic types of HMI are the pushbutton replacer, the data handler, and the overseer ("Anaheim Automation") and each type has its own particular niche with which it excels. The pushbutton allows for the centralization of all the functions each individual button has effectively taking the place of many extra LEDs, on/off buttons, and switches through the use of an LCD screen ("Anaheim Automation"). The data handler is used with applications that require constant feedback and monitoring of a system, usually through graphs or other visual illustrations, and requires a large, generally high resolution, screen ("Anaheim Automation"). The overseer is usually used with sophisticated and complex systems that would need to run programs like Windows or applications like

SCADA (Supervisory Control and Data Acquisition) or MES (Manufacturing Execution Systems) (“Anaheim Automation”). Since the Interactive Automated Chess Set only requires simple inputs from the user and would be used in place of extra buttons and/or LEDs the pushbutton HMI seems to be the best fit for the project.

After narrowing down the type of HMI from the three basic ones listed above, the next consideration must be the physical properties that the HMI exhibits. Since the Interactive Automated Chess Set should not have to work in extreme environments, compensating for water, heat, vibrations, or other extreme conditions should not be necessary. The only major physical consideration that needs to be addressed for this particular project is the size of the HMI so that it can be mounted safely to the board without being cumbersome or interfering with the game play.

The next major concern is the software used to interface the HMI into the Interactive Automated Chess Set. The three main categories to consider are proprietary software, hardware independent software, and open software (“Anaheim Automation”). Since the proprietary software comes with the HMI device it is usually the easiest to integrate but is also the least flexible; hardware independent software is usually designed to work on multiple different HMIs and allows for more flexibility in the application; open software allows for the ultimate customization ability but puts the most strain on the programmer as it is the most difficult over all to program (“Anaheim Automation”). If a HMI can be found with suitable proprietary software that would be the first choice but in the event that that is not the case, then hardware independent software would be the next logical choice with open software left as a last resort.

When searching for a physical HMI to use one of the first companies found was Schneider Electric with their Magelis STO. The Magelis STO is a small touch screen panel backlit by LEDs with a removable power connector, USB capabilities for application downloads, and comes with the same software used for the larger Magelis products making this a very versatile product (“A small HMI with a big attitude” 1-4). The particular features that would make this product a reasonable choice for the Interactive Automated Chess Set start with the actual screen itself, “Two monochrome versions are available with three-color backlighting for high visibility and status indication. The display features 16 levels of grey, numerous character fonts, and the Vijeo Designer extensive object library, which all help to provide best in class, high-quality graphic animations”(“A small HMI with a big attitude” 1-4). Another positive feature of the Magelis STO is that the panel comes with spring clips to allow for easy mounting (“A small HMI with a big attitude” 1-4). One final feature that seems to make this a decent choice for use with the Interactive Automated Chess Set is the choice of how it can be connected to a system either through RJ45 serial, Ethernet, or Zelio Serial ports; it also comes with “Two USB ports: one mini B port for downloading

and one standard A port for USB memory stick and USB peripherals" ("A small HMI with a big attitude" 1-4).

These features would seem to make the Magelis STO an obvious choice but there are some points that detract from this particular HMI. The first is that, since this model is the smallest ones available through the company with the screen at 3.4" with a 200 x 80 pixel resolution ("A small HMI with a big attitude" 1-4), it does not contain any physical buttons. Since the original design calls for at least twelve buttons, as shown below in figure 2.4.1-1, this would mean that any of the planned physical buttons would have to be programed into the panel and that the screen couldn't be use much for display purposes without it becoming overcrowded. The Magelis line of HMIs do contain bigger panels with buttons but they only have a maximum width of six button across; this would lead to the panel becoming larger than necessary and possibly leading to interference with the physical layout and gameplay that is desired for the Interactive Automated Chess Set. Another prohibitive feature is the price at $425 for each panel. Since two panels are required for person versus person game play this would lead to the cost of the panels alone to become at least seventy five percent of the proposed budget.

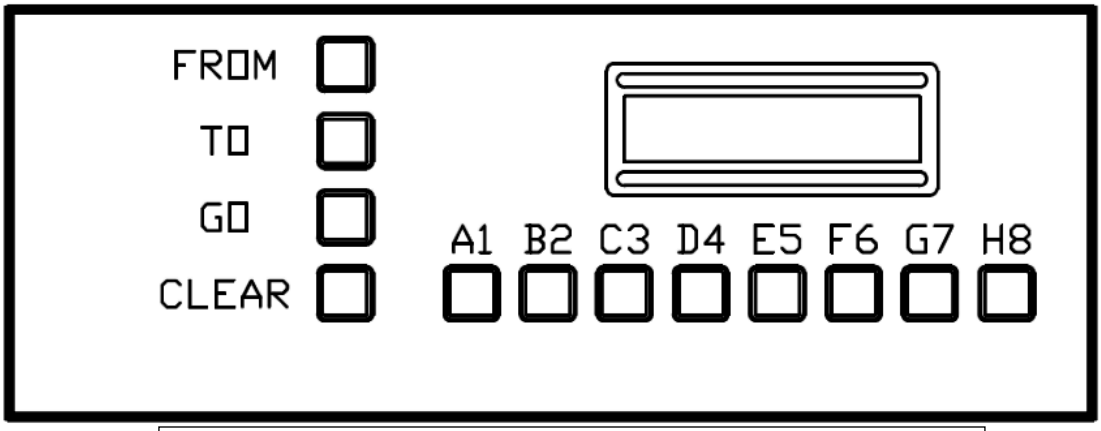

Figure 2.4.1-1: The original concept design for the HMI for the Interactive Automated Chess Set. This picture was drawn using AutoCAD educational software.

The keypads would be simple to mount into our HMI, but the layout does not maximize simplicity and intuitiveness for the user. This is item really is something we can and should design on our own. We simply need a switch matrix and some type of user display. When a similar chessboard was studied it had one switch for every row and one switch for every column. Then it had three function switches and a selector switch but this similar design only had one HMI; this design deficiency could create user confusion issues. Why not provide just eight zone switches and reuse the same switches for both row and column selection? The user will more than likely only use one switch at a time like the panel shown above in Figure 2.4-1. One of the most fundamental design considerations is how

will the end user react to interface; HMIs must be user intuitive. If the user fails to respond properly to a specific condition what will happen? For our chess game it is not really a big problem. The display will just wait and prompt the user to input some data, but if this was a piece of equipment with safety ramifications it's a major deal.
Another possible choice for our LCD display is a Serial LCD Module 20x4 Blue with White Backlight for Arduino products. Since it is designed to work with Arduino products, and can be purchased online for $30.16 (shipping included) each, it would be an ideal choice for our project. The only downside to this choice is that it does not come with buttons as an integrated part of the device but it does have the pins available to attach some of our own as needed.

2.4.2 LCD Controller

Most 16x2 LCDs have two registers. There is a command register that stores commands such as initialization, clear screen, or set cursor position. The data register stores the data to be displayed by the LCD. This data is usually stored as an ASCII value as shown below in Table 2.4.2-1 with the pin inputs shown in Table 2.4.2-2 below.

Hex Code	Command
1	Clear screen
2	Return cursor to home
4	Decrement cursor position
6	Increment cursor position
E	Display ON, cursor ON
80	Cursor to beginning of line 1
C0	Cursor to beginning of line 2
38	Use 2 lines and 5x7 matrix

Table 2.4.2-1: Commonly used LCD command codes

In order to use our LCD we must first check data pin 7, which is our busy flag. It means that the LCD cannot accept data at this time. In order to send data to the LCD, we must set R/W to low and RS = 1, which specifies that we are to latch the data in the data register as opposed to the instruction register.

We are planning on using the Hitachi HD44780 device. It meets all of the specifications that our general tutorial used. However, it has two modes of operation: an 8-bit mode and a 4 bit mode. In the 4-bit mode, we only use registers DB7-DB4 for data and we transfer the data in two packets. This is good for reducing our pin usage if it comes down to it, but it also adds a bit of complexity to the software side. In this case it is important to note that we must pulse the enable bit with a positive edge for each nibble of data we send. According to a timing diagram from the Hitachi data sheet, we can read the busy

flag by simply setting the R/W bit too high and pulsing the enable bit. DB7 will become the “busy” flag in this case. Please refer to Table 2.4.2-3 and Table 2.4.2-4 below for the signals for instructions in 8 bit operation for a Hitachi HD44780 LCD device.

Upon finishing research of the specific device, we realized many functions would have to be implemented to do something as simple as initialize the device and write a word. We went looking for a library that could do these things for us. We found a tutorial by Extreme Electronics, whose URL is http://extremeelectronics.co.in/avr-tutorials/using-lcd-module-with-avrs/ which discusses a library they have created and its use in AVRs. In AVR studio, we can add it directly as source files. It contains a function to initialize the LCD with parameters for a blinking or underlined cursor. It has a function simply called LCDWriteString for writing a string to the LCD screen. To write numbers, we simply call LCDWriteInt. For changing the position of the cursor, there is a function called LCDGotoXY. For clearing the display, there is a function called LCDClear. Depending on which I/O ports we use for the LCD connections, we can change the header file.

Pin	Code	Description
1	VSS	Ground
2	VCC	Main Power
3	VEE	Power supply to control contrast
4	RS	Register select
5	R/W	Read/write
6	EN	Enable
7	DB0	Data pin 0
8	DB1	Data pin 1
9	DB2	Data pin 2
10	DB3	Data pin 3
11	DB4	Data pin 4
12	DB5	Data pin 5
13	DB6	Data pin 6
14	DB7	Data pin 7
15	LED+	Backlight power
16	LED-	Backlight ground

Table 2.4.2-2: Pin inputs to LCD device

Instruction	R/W	RS	EN	DB7-DB0
Function set (8-bit operation, 1 line	0	0	Posedge	0b001100XX
Display on/off control (turns on display and cursor)	0	0	Posedge	0b00001110
Set cursor to auto increment during	0	0	Posedge	0b00000110
Write data	0	1	Posedge	ASCII code
Write command	0	0	Posedge	Command code
Read busy flag	1	X	Posedge	

Table 2.4.2-3: Signals for instructions in 8 bit operation, 8 digit x 1 line display mode for Hitachi HD44780 LCD device

Instruction	R/W	RS	EN	DB7-DB4
Function set (8-bit operation, 1 line display)	0	0	Posedge	0b0010
	0	0	Posedge	0b00XX
Display on/off control (turns on display and cursor)	0	0	Posedge	0b0000
	0	0	Posedge	0b1110
Set cursor to auto increment during write	0	0	Posedge	0b0000
	0	0	Posedge	0b0110
Write data	0	1	Posedge	ASCII code (1^{st} 4 bits)
	0	1	Posedge	ASCII code (2^{nd} 4 bits)
Write command	0	0	Posedge	Command code (1^{st} 4 bits)
	0	0	Posedge	Command code (2^{nd} 4 bits)
Read busy flag	1	X	Posedge	X
	1	X	Posedge	X

Table 2.4.2-3: Signals for instructions in 8 bit operation, 8 digit x 1 line display mode for Hitachi HD44780 LCD device

2.5 Reading the Board State

2.5.1 Visual

We could use a visual system for reading the state of the game board. This would involve mounting some sort of camera over the board with view of the board. Besides the difficulty of actually implementing vision algorithms on a small piece of software, we might have trouble mounting a camera in a location that

can see every space on the board and is out of the way of the motor controller, which hovers above the board.

2.5.2 Hall Effect Sensors

The purpose of the Hall Effect sensors is to sense which specific chessboard squares are occupied with a chess piece and which are not. These placement sensing devices give the computer algorithm, which keeps track of the location of each individual chess piece, a physical comparison to the software's tracking program for evaluation and error handling purposes. Research into Hall Effect sensors led to several different types of sensors that could potentially be used successfully with the Interactive Automated Chess Set; one such sensor type is the latching Hall Effect sensor. This sensor turns from on to off by utilizing magnets that switch between a north pole and south pole near the physical Hall Effect sensor. Another popular type of Hall Effect sensor is a simple switch Hall Effect sensor; the simple switch Hall Effect sensor will be in an off state in the absence of a magnetic field and an on state in the presence of a magnetic field up to a certain magnetic strength. Due to each of the chess pieces containing a magnet in the bottom, the switch Hall Effect sensor would be able to send a digital signal to a multiplexer which would then be sorted and sent to the microcontroller; this makes the simple switch Hall Effect sensor ideal for the Interactive Automated Chess Set.

After choosing the type of Hall Effect sensor that is to be use with the Interactive Automated Chess Set the sensor grid had to be designed. After considering several different designs it was decided that the final grid should be based on simplicity; this also means that the grid should take up the fewest number of pins from the microcontroller as possible. Keeping these goals in mind it was decided that an overarching concept would be developed to set up the structure of the grid as a control/feedback system with the microcontroller governing a demultiplexer that feeds into the Hall Effect sensor grid. That demultiplexer would then lead into a multiplexer that feeds back to the microcontroller (*See Figure 2.5.2-1 below*). After this structure was finalized with the group, the rest of the design decisions were based around this system which left the decision of how many pins the overall design would need to connect to the microcontroller a much easier one to make especially since it was focused so that the project could stay within reasonable parameters such that the design did not become overly cumbersome, or too simplistic, to function properly.

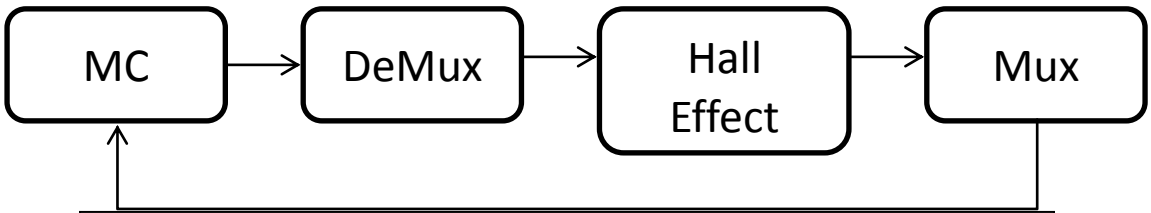

Figure 2.5.2-1: The flow chart for the final set up for the Hall Effect sensor grid; MC refers to the microcontroller, DeMux refers to the demultiplexer, and Mux is the multiplexer.

The first possible design choice was to have a 4:16 demultiplexer and an 8:4 multiplexer that, when both are enabled, run an internal clock cycle into the selector lines; this would automatically switch between the selection lines controlling the inputs that go into the multiplexer and the outputs that go out of the demultiplexer. The microcontroller would be responsible for setting the four input lines running into the demultiplexer in an on or off position in various combinations, as needed, to turn on the outputs from the demultiplexer in a sweeping pattern that would supply power from the first output, ending with the last, and then repeating the process as many times as necessary. The Hall Effect sensor grid, which each column acts like an address line similar to how embedded memory addressing is set up, would have this sweeping pattern run through it; the resulting pattern would then go into the eight inputs of the multiplexer in the same sweeping pattern as the demultiplexer. The four outputs of the multiplexer would then send the resulting on and off signals to the microcontroller as feedback. This seemed like a good design due to the fact that the microcontroller would only have to dedicate eight I/O pins to the Hall Effect sensing system. The drawback to this design however is that if the multiplexer and the demultiplexer were not enabled at the exact same time they would be out of sync with each other and the microcontroller would get an exponentially growing amount of false readings which could lead to catastrophic system failure and the eventual in-operation of the Interactive Automated Chess Set.

The final design choice utilized a mixture of the first two possible grid designs to reduce the number of pins used by the microcontroller. The decision was made to replace the 8:4 multiplexer with a multiplexer that would have a single output line, creating a 4:16 demultiplexer and an 8:1 multiplexer version of the second design. This final design works similarly to the second design but it would only require eleven dedicated I/O pins from the microcontroller versus the original fourteen. The down side to this design is that the sensors would be read at a slower rate, but due to the pieces being moved in the time scale of seconds while the columns are read in the time scale of nanoseconds, this issue could be considered negligible. For a more detailed description and specifications for this final design of the Hall Effect sensor grid refer to the Design portion of the Hall Effect sensor grid in Section 3.6.

The final design choice utilized a mixture of the first two possible grid designs to reduce the number of pins used by the microcontroller. The decision was made to replace the 8:4 multiplexer with a multiplexer that would have individual selection lines, creating a 4:16 demultiplexer and an 8:1 multiplexer version of the second design. This final design works similarly to the second design but it would only require eleven dedicated I/O pins from the microcontroller versus the original fourteen. The down side to this design is that the sensors would be read at a slower rate, but due to the pieces being moved in the time scale of seconds while the columns are read in the time scale of nanoseconds, this issue could be considered negligible. For a more detailed description and specifications for this

final design of the Hall Effect sensor grid refer to the Design portion of the Hall Effect sensor grid in Section 3.6.

2.6 LED Setup and Controllers

2.6.1 LED Setup

No automated chessboard would be complete without some sort of visual uniqueness to draw a potential player's eye. In this particular project the visual setup of the board is required to be designed not only for aesthetic purposes but also for practicality; the visual portion had to be versatile and also helpful to the player. In order to accomplish this the group decided to use LEDs for lighting effects on each individual square on the board allowing the board the option of giving the player visual cues for many of the important aspects of chess including piece movement, invalid moves, warning of check or check mate, and a pawn changing into another type of piece. The board itself, while the power is off, will look like a flat pane of translucent plastic without any visible squares, but when the power is on the LEDs the pane will light up in a checkered pattern making up the traditional plane of the chessboard and designating between the "white" squares (the squares with light beneath them) and the "black" squares (no light beneath them).

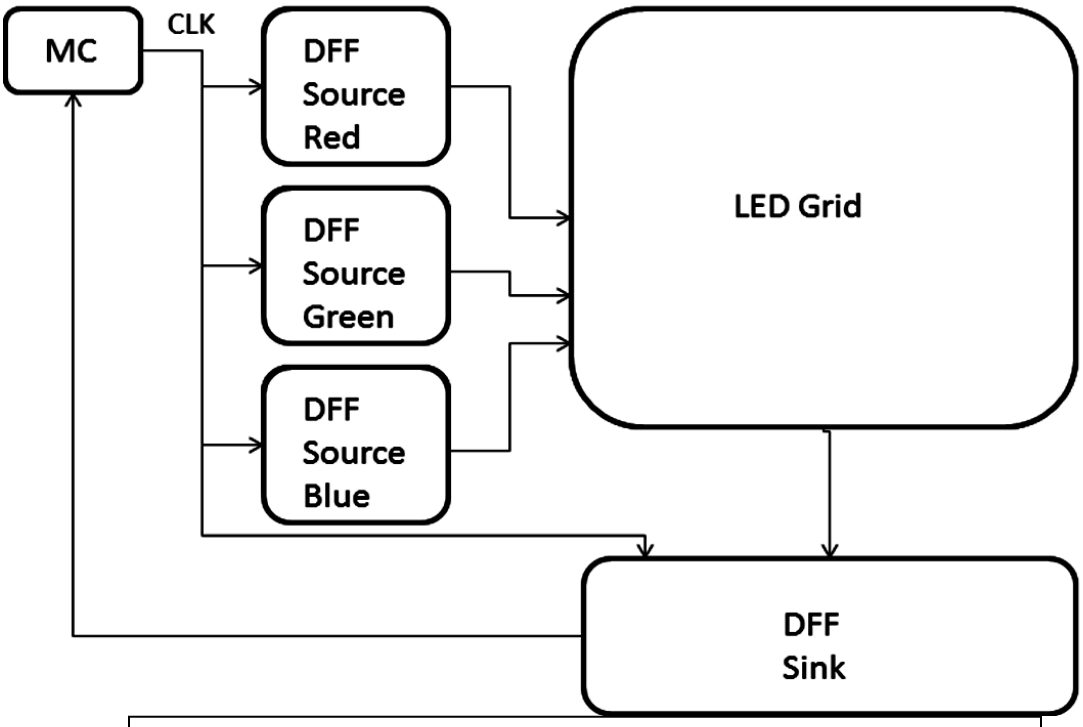

Figure 2.6.1-1: The basic layout for the LED setup and design; DFF represents the D flip flops

The first possible design was to set up the LEDs into an eight by eight grid with each LED having three anodes and a single cathode, technically making a twenty

four by eight grid, with each anode causing the LED to produce either a red, blue, or green light. The grid would be controlled by three sets of eight D flip flops (DFF), one set for each anode, with the clocks daisy chained together to create a source for the LEDs. To create a sink for each LED's cathode a row of D flip flops would also have their clocks daisy chained together as well. The source and sink of each LED would be controlled by the microcontroller, via the D flip flops, and make it possible to control each LED individually as shown in Figure 2.6-1 above.

The problem with this design however is threefold; first the design would take up considerable space, a total of thirty two D flip flops, along with the LED grid and the rest of the system. One other major problem with this design is that there are too many clock cycle lines and if the clock cycles got out of sync with the microcontroller this would lead to the system becoming extremely unstable. The final major concern is that the D flop flops for the LED sources would be extremely problematic due to all the parts, i.e. resistors, capacitors, and so on, that would affect the data line, or the operation of the flip flop itself, would have their own percent error ranges, thus the risk of compounding the percent error range of the overall system could make the system work improperly. It would be far more cost effective to use an integrated circuit driver due to the different aspects of the driver working together with a lower percent error range.

The second possible design has the microcontroller communicating through a serial communication setup, with a LED driver that is designed to drive up to an eight by eight LED grid. The driver would have two more drivers connected to it serially, for better control, and to switch smoothly between all three colors. The first possible design, as described above, was to keep the three anode, one cathode LED, but given that the driver chip needs its sources and sinks separate from the other driver chips, the choice in the type of LED was changed to the single package three colored LEDs, each LED that has 3 anodes and 3 cathode. For a more detailed description and specifications for this final design of the LED grid refer to the Design portion of the LED grid in Section 3.7.

2.6.2 LED Controller

We are planning on using the MAX7219 LED Driver to drive our matrix of 64 LEDs. It is meant for controlling eight 7-segment LED digits, but each LED digit contains 8 separate LEDs. The advantage of using this device is that we will save on pins and hardware to multiplex an 8x8 grid. We began our research process for this device by reading the data sheet.

We find that the device is to be communicated in 16 bit packets, in the format of the table below (Table 2.6.2-1). We begin sending a packet by bringing LOAD low. On the rising edge of each clock tick, a bit will be latched into the digit or control registers. After the 16^{th} rising clock edge and before the next clock edge, LOAD must go high again. The bits numbered D8-D11 contain the register address to store the data in, and the bits numbered D0-D7 contain the data. D15 is the most significant bit.

D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**
X	X	X	X	Address3	Address2	Address1	Address0
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Data7	Data6	Data5	Data4	Data3	Data2	Data1	Data0

Table 2.6.2-1: Serial-Data format of MAX7221 Driver

For our purposes, we will set the “decode” register to “no decode” mode, which means we don’t transfer our data in the form of binary-coded-decimals. We will write to our matrix by specifying a register (0x1 through 0x8) and then applying an address to it. Since the data bits D0 through D7 correspond to the segments of one digit in a 7-segment display, we are able to update an entire column in one 16-bit packet. We found an article that explained how to interface with the device for an Atmel AT89C2051 microcontroller at http://ee.cleversoul.com/max7219.html. It contains source code that will be useful for our purposes.

2.7 Magnets (Pieces)

A chess game would not be possible without the most fundamental part of the game: the chess piece. One would think that this part of the project would be the easiest but due to the computer, via a grabber, picking up and placing each piece instead of a human hand it is not as simple as buying a standard chess set. With the Interactive Automated Chess Set using a robotic claw to move the pieces and utilizing the Hall Effect sensors to keep track of which piece is where, for error handling purposes, a small magnetic field is required. The strength of the magnetic field is also important since the Hall Effect switches are sensitive to a certain range of magnetic fields; this means that the right type of magnet has to be chosen and that the size of said magnet must not greatly affect the height of the piece since this would complicate the use of the robotic claw. The other reason for the magnet is to keep the piece in the center of each square on the board using a section of ferrous metal in the center of each of the squares; this should cut down on potential positioning errors with the chess pieces since the magnet in the bottom of each piece will be attracted to the metal at the center of each square allowing for correct seating of the piece at each and every turn throughout the game.

The first possible design is to get store bought short pieces. A hole would then be drilled into the base of the piece and a small refrigerator magnet, around fifty Gauss in strength, would then be inserted. The sides of the pieces would be filed down to make the pieces flat on the sides so that it would become much easier to grip by the robotic claw. This design has the flaw that if the piece is rotated too much the claw would grip the rounded side and the piece could slip out of the grip of the claw.

The second possible design is similar to the first design with the exception of putting memory foam padding on the grips of the claw. This means that not only the pieces do not have to be filed down on their sides but it also allows the claw get a better grip on the pieces, making it less likely for the pieces to slip from the grip of the robotic claw. For a more detailed description and specifications for this final design of the chess pieces refer to the Design portion of the Magnets (Pieces) in Section 3.8.

2.8 Mechanical Assembly

Our Automated Chessboard is comprised of three overhead motion axes as well as a grabber to pick up and relocate chess pieces one at a time. For this project the Y axis will be defined as the movement either toward or away from the user, the X axis will be defined as movement either to the left or right of the user, and finally the Z axis will be considered as up or down movement of the claw/picker. When the need to relocate a chess piece arises, the microcontroller will have the X and Y coordinates stored in memory so that the Z axis will always travel the same distance up and down to either lift the piece up or set it down, regardless of the location, and the grabber will either open or close as needed. In the research phase design goals of accuracy, repeatability, stability, and reliability will all be considered. The research strategy is first to find and derive the best solution for our application through understanding what possible options that we have available to us. Similar designs, along with learning what parts are available will be studied to obtain knowledge necessary to complete the project to satisfaction.

2.8.1 Similar Applications Studied

The Automated Chessboard assembly is quite similar to a CNC table; we were able to find a CNC engraving machine on the IGUS Inc. website ("igus: plastics for longer life") and it was first studied to get a general idea of how to approach the project. The CNC machine studied used four lead screws to control motion. The table was manually controlled, requiring the user to turn knobs and lock down the lead screws when the desired coordinates were properly obtained. We additionally had the need to automate the lead screw. The machine had a number of designed and machined parts customized to connect the lead screw assemblies together. Furthermore no instructions or documentation was available for the various parts.

The CNC table used two parallel lead screws in the Y axis direction. Using two parallel driven axes in the same direction presents the need to deliver equal and synchronous forces to the lead screws. If the equal and synchronous forces are not obtained the failure mode can result in mechanical binding; mechanical binding will not be tolerated by our design and could lead to the entire system crashing mid game.

The second similar design studied was an automated chessboard on the lets make robots website (patrickmccabe). This application was impressive, but with our design requirements, we would need to take it to the next level. This robot just drops discarded pieces on the side. This design also exhibited the same two parallel driven axes problem that was discovered in the CNC example. After reviewing the problem the easiest solution presented itself: if we could reduce the design down to one driven axis for X axis motion, not only would this action help mitigate the binding problem but we would reduce cost and complexity significantly. The video of the operation of this machine highlighted the stability and accuracy goals. Watching the video showed that our project was obtainable and achievable as well as showing that not only can we do this but we can make significant improvements and learn from their mistakes. The gantry crane was only supported by one linear bearing for each side. This design of the two vertical support members allows for mechanical stop resulting in unpredictable placement with respect to the Y axis. The up and down motion along the Z axis indicated that improvements could be made there as well.

2.8.2 Available Parts Studied

Three suppliers were considered for the task. First IGUS Inc., second was Rollon S.P.A., and finally Sprocket Drive Products and Sterling Instruments Company (SDP/SI). With the knowledge gained from the study of the similar applications in Section 2.8.1, the total numbers of possible methods were narrowed down to four possible options. The four options are lead screws, linear bearings, slide rails with carriages, and a belt and sprocket system.

After obtaining some pricing quotes for the lead screw assemblies, the conclusion was quickly reached that the lead screw approach was highly cost prohibitive. The lead screw approach would require the need to deliver a higher torque from our motors to turn the lead screws. The belt and sprocket method proved to be problematic as well with respect to failure modes. With the need to achieve consistent and repeatable placement of chess pieces overcoming the application of consistent belt tension along with mitigating belt slippage could not be assured.

The slide rail carriage system, as well as the linear bearing option, could utilize a rack and spur gear drive system with the spur gear connected directly to the motor shaft. The rack gear system would insure that each stepper motor step would be translated into a repeatable linear displacement motion. SDP/SI turned out to be a great resource with great information on their products and free AutoCAD drawings available for most products. Rack gears are normally sold in sections of a given length and although multiple rack gears can be arranged to increase the length, we would be forced to scale our design to N times the length of the rack gears. Although it would be possible to cut a brass or steel rack gear section, cutting a section of Stainless Steel would be difficult. Some of the available rack gear sections came with predrilled and tapped screw holes for

easy mounting, but most did not. Without predrilled and tapped mounting holes, we would be forced to drill and tap ourselves or devise an alternative method of securely attaching the rack gear to the assembly. A Stainless Steel rack gear would also be difficult for us to drill and tap.

The research revealed that either the slide rail carriage or the linear bearing system would be viable and decent choices. Although the slide rail carriage is slightly easier to assemble, the linear bearing option requires an end block mount piece to hold the rod in place. IGUS has a Young Engineers (YES) program that donates product to students. The team has applied to the YES program and requested slide rails and carriages for the X and Y axes. The IGUS site had a nice configuration software tool (DryLin® Expert 2.0) to verify that the design meets good engineering methods and practice standards. One of their primary design rules is the 2:1 rule; it was this tool that revealed the design flaw during the evaluation of the similar chessboard studies. If two parallel slide rail carriages or two parallel linear bearings, which were used in this case, they must not span a distance greater than twice the length of the carriage or the bearing. Using two bearings or two carriages on each rail or rod respectively will increase stability and reduce the potential for binding. This presented an interesting challenge to still maintain the 2:1 rule. The details of how our design solved the 2:1 rule will be covered in the design section.

The Z axis (up and down motion) of the chessboard study also revealed design flaws and total redesign was our best option. The chessboard studied used a belt and a hobby type servo motor. When operating in a vertical motion application gravity should be considered and used to our advantage. When considering vertical motion we only need to lift up as long as we can control the descent. For our design we only need to lift up any given chess piece high enough to clear the other pieces; in the case of the studied chessboard the Z axis lifted the pieces higher than needed and was clearly un-stable and unwanted oscillatory motion was easily observed.

The chessboard studied was mostly comprised of lumber with the main cabinet being constructed with plywood. Our design requirements call for the LED and Hall Effect sensor grid to be under the board; this will clearly change our packaging strategy directing us towards some type of frame system that the board itself can be placed in. Lumber products including plywood will make for good reasonably priced and highly available structure materials but other options should also be considered such as PVC lumber. PVC can be light weight and simply glued together. For construction of mechanical frame assemblies such as motor mounting, VEX Robotics Inc. offers a number of good metal structures and hard ware options. Finally surplus supply is a great choice. Although availability will change over time, and we will need to be flexible with design and drawing changes, we have already acquired some surplus supply parts. Aluminum or steel frame members are good options, but our team does not have the welding skills or resources to utilize these materials.

2.8.3 Wire Management on a Machine with Moving Assemblies

Wire management comes into play with our design because of the moving assemblies. We must avoid pinch points and extreme conductor bending; troubleshooting a wire harness with broken conductors that pull apart only when the cable is in a specific position is a nightmare to diagnose. Even worse is attempting to figure out all of the possible system responses due to wire harness failure modes.

We have a few options to help us mitigate these issues by design. One thing we can use is high flex cable that is often used in industrial robot or machine automation applications. High flex cable is always a stranded conductor with a very high strand count; never use solid conductors in a bending application. Another popular solution is to use a flexible wire chain product such as products in the IGUS Inc. Chainflex® line. For some applications wiring can be pulled in through a flexible plastic or rubber tube. When designing any wire management system one of the main goals is to maximize the bending radius thus distributing the bending over more of the conductor length. We plan to carefully layout our sections of flexible wire harnesses in the design stages.

2.9 Motors and Motor Control

The motor and/or propulsion drive system is a critical component to deliver the smooth, accurate, and precise motion of our machine. One of the most significant considerations is to employ an open loop or a closed loop control scheme. The closed loop approach is widely deployed in industrial applications successfully. We must have confidence that our machine will accurately move to the position that the microcontroller has commanded it to. We cannot tolerate error build up because as the game progresses, more and more moves have been made and any error can result in the grabber no longer being able to align with the proper chess piece for the entire rest of the game. Ideally it would be desired to design just one motor control system that could be used for all three axes.

2.9.1 Motors and Actuators

Four types of propulsion were considered in the research phase. The four were the servo motor, the electric induction motor, the linear electric actuator, and the stepper motor. The automotive industry uses electric linear actuators more and more and common uses include opening doors, windows, trunks, hatches, and even extending radio antennas. Anytime automotive parts are widely used, they become available on the after-market. After-market automotive parts are easy to acquire and are typically very cost effective. The ability to simply mechanically attach one end to the movable axis member, and the other end to a stationary frame member, is highly desirable. Automotive linear electric actuators would have plenty of torque to deliver to our load; unfortunately some method of position feed-back would be required. It would not be easy to modify an actuator

to mount an encoder to the actuator motor. Even if the automotive electric actuator came with an internal feedback device, there would be a challenge to acquire enough of the technical details to design electronic circuitry to interface and monitor the position information from the actuator; a secondary feedback system would be required.

The electric induction motor has many advantages including a wide selection and good control under variable frequency drive control (VFD). One disadvantage to the electric motor is that we would have to deal with 120 VAC which would introduce electric safety NFPA 70E issues. VFDs are widely used in industrial applications, usually in larger scale three phase applications, typically with 208 VAC or 480 VAC motors. Noise is also a major concern here too; special shielded VFD cables can be used to reduce the unwanted effects of electromagnetic interference (EMI). Unlike with shielded cable used in communication systems, when it is used in power systems the shield is tied to ground at both ends. VFDs normally use insulated gate bipolar transistors as the high power switching component. Due to the high switching frequencies large voltage transients are generated and the transients can damage the thin varnished insulation of the motor windings; because of this issue motors connected to VFDs must have motor winding wire rated for VFD use.

The servo motor is also widely deployed in industrial applications. DC Servo motors typically use an optical position encoder feedback device which requires a complex closed loop control system. We could clearly design and build our own DC PWM DC motor drive but more than likely we would also need to design and build some type of encoder interface. That would condition the signal before it can be read by the microcontroller. Often industrial applications use what is referred to as a high speed counter card to convert quadrature pulses to a numerical value that the controller can read. The controller would then possibly use a PID control scheme. Servo motors are also commonly used on hobby robot systems but the consumer normally purchases the motor and the motor control drive system together. One critical failure mode to always consider in any closed loop DC motor control system is loss of feedback. If not properly mitigated the motor can runaway undetected which could result in mechanical equipment damage. With AC motor drives the drive can control the motor within about 1% accuracy even without feedback.

The stepper or DC brushless motor was the last option considered. The stepper motor rotates at a fixed angular displacement for every motor pulse; basically a stepper motor can be thought of as a digital motor. Unlike the typical DC motor that is comprised of an armature with a commutator that rotates, and is mounted inside a motor frame with field windings, in the case of the DC motor with brushes and a commutator, the armature is wired so that the armature current switches the polarity of the magnetic field every time the brush crosses over to the next copper bar; in this case DC Voltage is supplied to the motor leads. To reverse the direction you simply reverse the current through either the armature

or the field but not both. With the stepper motor containing permanent magnets in the rotor, a voltage is applied to the fields and the motor shaft will rotate to the next step. If the voltage remains applied, the stepper motor will hold its position acting like a motor brake; this is the result of the alignment of opposite magnetic poles (the south pole in the rotor is aligned with the north pole in the stator for example). The magnetic field created from the electrical current that flows through the stator field coils; if the need to advance to the next step occurs, the stator fields must change.

There are two common styles of the permanent magnet stepper motor: the unipolar and the bipolar. The unipolar motor has center tapped field windings, while the bipolar does not. A quick and simple way to figure out if a given motor is a unipolar or bipolar motor is to count the leads; the bipolar motor will have four leads and the unipolar motor will have five or six leads. The field coil current in the bipolar motor must switch directions for every step that is advanced. In the case of the unipolar motor only one half of the field coil is ever energized at any given point in time and the center tap is connected to the DC common terminal.

The big advantage to the unipolar motor is simplification of the motor control circuitry. The motor control circuit simply turns on one coil and the other turns it off; it then turns on the other coil and this turns the first one off. This process is repeated for every step that is advanced. The disadvantage to the unipolar arrangement is that the center tapped field coil must occupy twice the area compared to the unipolar device. This greatly decreases efficiency, and the torque delivered to the load. The downside to the bipolar motor is the complexity of the motor control circuitry. Normally some form of solid state switching must not only turn on and off the motor fields, but it must reverse the direction of the field coil current flow; fortunately there are a number of good options to accomplish this goal. Reference the Motor Control section (2.9.3) for further details on this subject.

The stepper motor will produce the most torque as it just begins to rotate and the torque is reduced as the speed increases; this is not necessarily a bad thing. Having high torque initially to overcome the coefficient of static friction, and the lower torque while operating under the coefficient of dynamic friction all while the mechanical load has momentum, is an advantageous condition. Motor braking of a standard DC motor can be done by disconnecting the motor from the power source and connecting a resistive load across the motor terminals; this form of braking is referred to as dynamic braking. Basically the motor is turned into a generator where mechanical energy is converted to electrical energy and finally the electrical energy is converted to thermal energy. The disadvantage of dynamic braking is that it fades out at low speeds. After the dynamic braking effect has faded away at lower speeds a secondary method of braking is needed, typically taking the form of some kind of friction braking. In large DC motor applications the DC motor drive can regenerate the power back to the utility. If a

stepper motor that has higher torque at low speeds, and it can even hold the motor in place, the braking problem is easily solved.

2.9.2 Position Feedback

In order to construct a closed loop control system, some form of positioning feedback is needed; in all of the cases previously mentioned above an independent position monitoring system could be deployed. Possible options for the system could be a linear position transducer; MTS Sensors® has the Temposonics® line of high quality linear position transducers which are used in numerous industrial applications. This approach would certainly provide accurate and reliable feedback to the control system. One down side is that now we would have two parallel moving mechanical devices for each of the three axes of motion which would complicate the design. Another major downside is the cost of the Temposonics® line; we would most certainly need to obtain sponsorship from the company in order to make this part choice a viable option.

String pots would also be a suitable feedback method. Celesco Transducer Products Inc. (**www.celesco.com**) has a number of good string pot offerings and their transducers come with 0-10 VDC, 4-20 mA, CANBuss, and DeviceNET® outputs; out of these the 4-20 mA output would be the best choice for our application as the 0-10 VDC would be subject to noise issues from the motor control circuitry, and this unit is just a potentiometer with a wiper that can fail. The CANBus and DeviceNET® options would also require a high level of design work in order to integrate seamlessly with our microcontroller. A second significant advantage to the 4-20 mA feedback is that it features broken wire detection which would offer a great mitigation to the unplugged connector failure mode. Options like CANBuss and DeviceNet are being used more and more in Programmable Logic Controllers (PLCs) to communicate with field devices.

Other position feedback options include DC Tachometers and optical encoders. The main problem with DC tachometers is that they are highly subject to noise as well as the fact that the output is delayed until the magnetic fields can be generated; just as the output is delayed as the motor starts to build speed, the output drops out early as the motor slows to a stop which would leave our monitoring systems and regulators blind for these short intervals leading to the optical encoder as a much better choice. Encoders come with a number of options including Pulses per Revolution (PPR) which typically range from 512 PPR to 4096 PPR while the outputs are normally quadrature with A, A-B, and A, B, C phase selections. Often times they come with inverse outputs where, for example, the A output has a paired up A not output; when an encoder has such complementary outputs both outputs change polarity. The change of polarity results in the need to provide special I/O to read the signal, which is generally referred to as deferential analog input. Optical encoders also provide excellent precise position feedback. These multi-phase devices coupled up with the control system can even detect direction of rotation; all of these features lead to the

optical encoders as a great choice especially when engineering analysis shows that failure modes must be detected and properly mitigated.

2.9.3 Motor Control

Our motor control considerations included DC Pulse Width Modulation (PWM) Dives, VFDs, stepper motor bipolar H Bridge, and the stepper motor unipolar controller; the DC PWM drive is a great senior design project option with a good number of designs to choose from. Most DC PWM drives used a high power semiconductor device for the high speed switching function; this helps the DC PWM drive provide smooth motor control. Not only can a DC PWM drive control motors they can also be used for DC light dimmers. One of the many examples considered at can be found on http://picprojects.org.uk/projects/ppc/index.htm.

The Variable Frequency Drive (VFD) was considered to possibly control a small fan motor re-deployed as an axis propulsion motor; surprisingly enough this possibility is quite feasible. Small 120 VAC single phase VFDs are readily available and reasonably priced. The only major down side to the VFD is that at 120 VAC this is not a good senior design project and not a practical thing for us to use with the Interactive Automated Chess Set. The first of the two stepper motor controllers evaluated was the unipolar drive; this is the simpler of the two main stepper motor controllers. In one example found the coil center tapped terminals are tied to the positive DC supply terminals and the DC common is connected to each of the semiconductor switching devices. This circuit can also be designed with the wiring completely opposite of the first case where voltage is applied to only half of each field winding at a time; to reverse motor direction apply the voltage to the opposite side of each field winding.

Anytime an inductive load is switched off in a DC application there is a magnetic field that must collapse; from the study of magnetic fields we know that an electrical current is used to build a magnetic field. When the electric current is removed there is still magnetic energy stored that must be dissipated from the winding. This magnetic energy will be converted to an electrical current, but in an opposite direction of the current that generated by the magnetic field to begin with. One can visualize this effect as an impulse on the DC common buss with respect to the DC positive buss. By installing what is commonly referred to as a free wheel diode across the coil, with the Cathode connected to the positive terminal and the Anode connected to the negative terminal, adverse effects can be minimized. With the free wheel diode attached at the time the field collapses, and the impulse that is generated, the diode acts as a short circuit for the impulse. Free wheel diodes are often used in stepper motor control applications, just as they are commonly connected across relay coils.

Thermal Dynamics is always an important consideration when analyzing any motor switching component; failure to operate the electronic devices within rated temperature specifications will result in catastrophic component failure. This is an

important issue identified here in the research phase that must be properly mitigated in the design phase. For years motor dive semiconductors have been mounted to heat sinks and, typically, a thermal compound is applied on the surface of the device that is flush with the heat sink surface to maximize thermal transfer. Today as electronic circuitry has been greatly reduced with respect to physical size new heat sink technology is available. The PowerSO-20™ surface mount package is now available and should be considered for use with the Interactive Automated Chess Set. This package style is dependent on taking advantage of the copper PCB traces to dissipate the heat from the surface mount package; this is a great technical advancement for large scale Original Equipment Manufacturers (OEMs) because size, weight, and cost are all reduced. More information on the impressive PowerSO-20™ surface mount package can be found in the following PDF uploaded to the internet for convenience:
<http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/APPLICATION_NOTE/CD00003801.pdf>.

Our team visited Skycraft parts and surplus Inc. early on just to help trigger ideas. On this visit a Sanyo stepper motor was purchased for $5.00 for the sole purpose of being able to play with and learn more about stepper motors; it turns out that this motor was the bipolar style. This proved to be a good call because it led to the inquisitive investigation of stepper motors, stepper motor controls, and an in-depth look into gears. After attending UCF Senior Design day last semester, and talking with students, it was learned that motor control was important to develop early on. Our team has identified the motor control circuit as a critical path that requires bread boarding in order to become a viable part of the system for the Interactive Automated Chess Set.

2.9.4 Stepper Motors

Most stepper motors are driven the same way. We searched around the internet for articles describing how to drive a stepper motor with an AVR, and found a great article at http://extremeelectronics.co.in/avr-tutorials/stepper-motor-control-avr-tutorial/. In order to drive the stepper, we need to drive coils. Since we can only sink or source about 20mA from our microcontroller, we need a motor controller unit. Each stepper is controlled by activating four coils in series.

2.9.5 Servomotors

Our grabber will most likely be a servo motor. Most servo motors have a control signal of a frequency around 50Hz. This equates to 20ms signal period. We rotate it by varying the width of our pulse. To get a concrete example, we use the Futaba S3003 standard servo. The following table (Table 2.9.5-1) is the pulse width versus angle of the servo; we can implement this in software by using a hardware timer that comes with our AVR microcontroller.

Period	Angle
.388 ms	0 degrees
1.264 ms	90 degrees, centered
2.14 ms	180 degrees

Table 2.9.5-1: Pulse period versus servo angle

2.10 Gripper/Claw

The goal of our gripper research is to discover our options relating to making the best selection for a grabber, gripper motor or solenoid, and finally learning how to interface the selected products with our microcontroller. This item was identified as an assembly that we intend to purchase and integrate into our project so as not to spend a lot of time and effort on design work. To fully design and build a griper from scratch would be good project scope for a mechanical engineering project, but not for Electrical and Computer Engineering students. The chessboard that we studied had a custom built grabber but the assembly took up more vertical space than we would ideally like to use in our design. As discovered in our mechanical assembly research, limiting the vertical length of the Z axis can improve mechanical stability. Early on a grabber mechanical assembly was found that looked like it was scaled perfectly for our application; the grabber opened up to 1.3 inches, with a 90 degree rotary action, and was light weight and easily mountable to our chessboard. In this case we purchased the grabber and then chose to engineer it into our design. With the decision to purchase the grabber instead of designing and manufacturing it ourselves, the research focused on ways to drive the grabber open and closed. The hobby robot industry offers three possible solutions: the gear motor, the servo motor, and the electric solenoid.

When the device arrived and we were able to manipulate it, discovering exactly how it worked, we were then able to explore all of our options as to how to specifically integrate it into our system and have it function for a peak performance. The device has two movable claws lined with foam padding. A mechanical pencil could be placed in the grabber claws and it was capable of holding and picking up the pencil by hand without applying a mechanical rotary force. The device had a disk with two round pins which means we could easily install a lever arm that would mate up with the two pins. The end of the lever arm could be fitted with a spring and a solenoid. The concern for this option would be mitigating the mechanical impulse when the solenoid was de-energized and the lever arm was pulled back by the spring. We could design a coupler disk and drive the grabber with a gear motor. This is a practical and cost effective option for us, but we would certainly need both an open and closed sensor. The device is designed to work with a servo motor, and that is a workable cost effective option.

2.11 Audio Design

2.11.1 Audio System

The proposed audio system is an added feature with the purpose of improving the user experience through audio cues as well as added fun factor. In order to understand our options for audio, we did a survey of out of the box audio technologies to integrate into our system. Our goal was to find a system that was simple to implement and loud enough to be heard by users without being piercing.

2.11.2 Magnetic Buzzer

We looked at the CEM-1203(42) buzzer. It is rated at 3.5 volts. It can be driven by a 2048 Hz square wave. It makes a square wave sound. It only takes 1 digital output pin to drive, which is not a problem for us. We would most likely want to use a hardware timer for this, since we want sounds to play as the game is being played. It can be soldered directly to our circuit board. Many reviews of this device have said that it is not loud enough. We can buy one for $2 on Sparkfun. We also found a circuit for driving the buzzer with two transistors and a 3.3k resistor on elasticsheep.com. This will allow us to get the maximum amount of power out of it and can be found on **http://elasticsheep.com/2009/10/driving-a-buzzer/.**

2.11.3 Wav File Player

We found a wonderful tutorial on Sparkfun that would allow us to make a wav file player. The tutorial was found at http://www.sparkfun.com/tutorials/160 . We can build this using mostly off the shelf parts. Table 2.11.3-1 below shows the required parts.

Part	Note
AVR Microcontroller	They used an Arduino
AD5339 8-Bit DAC Breakout Board	Converting parallel digital signals to analog
microSD shield	Data storage
microSD card	Data storage
Speaker	

Table 2.11.3-1: Required parts for wav file player

The price of the parts is in total around $40 not including the microcontroller. The microSD shield is $15, the SD card at $10, and the breakout board at $10. It is

mostly a plug and play operation, with the only a bit of soldering required. There is in essence no design to be had. We connect the microSD shield to an AD5330 breakout board. The microSD shield interfaces between our wav files and the breakout board. The Arduino people have libraries for implementing a FAT16 or FAT32 file system for file manipulation. The AD5330 converts 8 bit digital signals to analog. We connect an AC coupled speaker to the VOUT pin of the AD5330.

The tutorial suggests converting .wav files an 8-bit sample width with a sampling frequency of 22 kHz. We would have to create our own library that uses hardware timers to pulse the speaker at the correct rate. The system would work with the following process:

- Find a WAV file on the SD card
- Create two buffers for double buffering
- Play from the first buffer while reading to the second buffer
- Swap buffers when first buffer is empty
- Repeat until file is complete

Luckily the tutorial doesn't end with Arduino. It discusses how to program it and compile it using WinAVR.

2.11.4 Comparison of Audio Devices

We have discussed two ends of the spectrum of audio. We can either use a buzzer or a wav file player. The wav file player requires more hardware and takes up more pins than the buzzer. The buzzer would be very simple to implement in terms of software and hardware. We would save on pins. At the very end of it all, the design prospect of both pieces is low, with the potential for learning being higher with the kit. If we have enough time and money, the kit would probably be the best bet. If it looks like we already have our work cut out for us, we may just use the buzzer or no audio at all. An overview of the comparison can be seen below in Table 2.11.4-1.

Item of Comparison	Buzzer	Wav File Player
Pin count	1	15
Sound quality	Low	High
Software difficulty	Low	High
Hardware difficulty	Low	Medium
Cost	$2	$40
Design prospect	Low	Low

Table 2.11.4-1: Comparison of audio devices

2.12 Power Supply

No matter what kind of engineering project is being designed in today's world it can be no more useful than a paper weight unless it has a way of getting power to run the actual project; the Interactive Automated Chess Set is no exception. Several considerations were taken into account when doing research for this portion of this project especially the means of getting power as well as converting, regulating, and distributing the power to the rest of the system.

The first design choice had portability in mind. The means of power would come from a battery pack, which consist of many smaller batteries, connected to a power regulation board. A variety of voltages are needed across the project ranging from 12 volts to 3.3 volts, the motors needing the most power and the variety of chips needing the least. The next step was to split the voltage into 12, 5, and 3.3 volts by having a chain of regulators with one regulator feeding the next regulator as shown in Figure 2.12-1 below.

Every component of the project requiring power would plug into this board to get their various power needs. The issue with this design is that the power drawn from the motors, which each motor can use upwards of three amps a piece, is an immense strain on the battery pack; this means that since there are several other systems also dependent on the battery pack, the longevity of the Interactive Automated Chess Set's battery life would be minimal. There are batteries that can handle this load but they cost hundreds of dollars and, due to chess often requiring many hours to play to a definitive finish, the more affordable batteries would run out of power after a short time of continuous play.

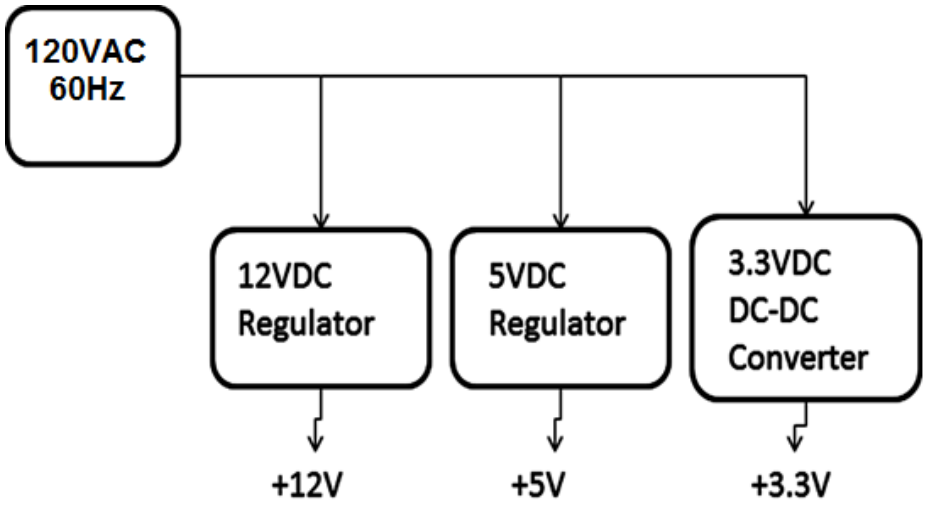

Figure 2.12-1: The basic layout for the first voltage regulator design and how it splits the incoming 12 volt power source from the battery to give out 12, 5, and 3.3 volts.

For the second possible design choice it was decided to use power from the wall outlet. This design requires the 120V AC, 60Hz from the wall to be converted to

12V DC and then be regulated into 12, 5, and 3.3 volts as needed for the rest of the project. An AC to DC converter board would need to be integrated into the converter portion of the board and the first design could be incorporated to finish the rest of this design as shown below in Figure 2.12-2. This design was ultimately tossed out due to the time consuming process to integrate in the AC to DC converter and the cost for something as fundamental as the power for the project.

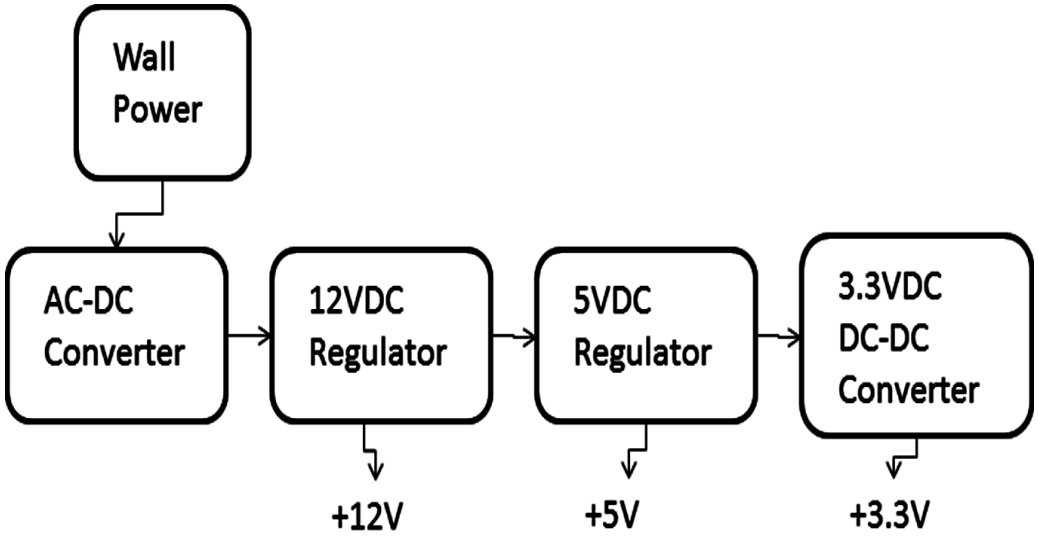

Figure 2.12-2: The basic layout for the second voltage regulator design and how it splits the incoming 12 volt power source from the wall to give out 12, 5, and 3.3 volts.

For the third possible design choice the power supply board will be designed to work like a regulation board allowing the power supply to come from a generic American wall outlet and the conversion from AC to DC voltage will be accomplished using a standard laptop computer power cord; this will provide approximately nineteen volts and nine amps. The voltage will be regulated into 12, 5, and 3.3 volts on the power board, the same as in the first two designs, and the current regulation for the 12 volt line that goes to the motor control board will be regulated in the motor controller due to the high current demand of the motors; the rest of the current regulation will be done on the power supply board itself.

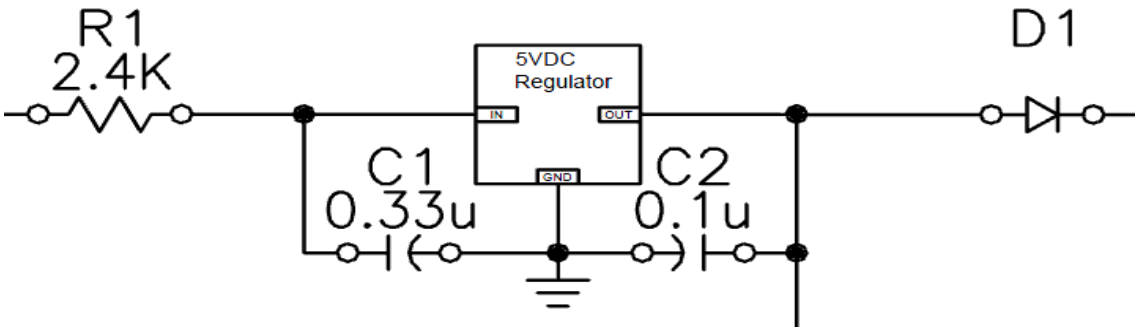

Figure 2.12-3: The five volt, 7805 positive voltage, regulator with TO-220 model casing

The five volt regulator is a 7805 positive voltage regulator, a TO-220 model casing, the schematic view shown in Figure 2.12-3 above, and the 3.3 volt DC to DC converter is a MAX710 surface mounted chip as shown in Figure 2.12-4 below; the schematic diagram showing how the two parts are connected together can be found in Figure 2.12-5 below. The voltage input range of the 5 volt regulator is 8 volts to 20 volts, giving an output voltage range of approximately 4.6 volts to 5.3 volts at about 5 milliamps typically (Refer to datasheet for more details). The 3.3 volt DC to DC converter chip can convert an input range of 1.8 volts to 11 volts to either a 3.3 volt or 5 volt output. For our project's use we want the output to be approximately 3.3 volts, so the chip logic would have to be configured to our needs, the logic consisting of the SHDN' (Shut Down), STBY' (Stand By), and 3/5' (3V or 5V output option) pins. All three of these pins need to be set high, 5 volts, so they are wired to the output of the 5 volt regulator. When the SHDN' is set high the chip is set from shut down mode to active mode, effectively turning on the chip. The STBY' pin is set high to prevent the chip from going into a standby mode for uninterrupted operation. The 3/5' pin is the pin that gives the option of a 3.3 volt output or a 5 volt output. Since we already have a 5 volt regulator the pin would be set high so that the output of the chip is 3.3 volts. The rest of the chip configuration is given from the datasheet for a "typical" operation (Refer to datasheet for more details). This design was rejected however due to the fact that our project needs a power converter that will give multiple reliable DC voltage sources with a large range of current capabilities, such as with the motors, the LED grids, user interface, and other parts of the project. Given our level of experience with power supplies it became apparent that stability of the circuit would be questionable, thus this design was rejected.

For the fourth design we decided it would be more cost effective to use off the shelf parts, this will give better stability to the power for our project. For a more detailed description and specifications for this final design of the power supply refer to the Design portion of the Power Supply in Section 3.13.

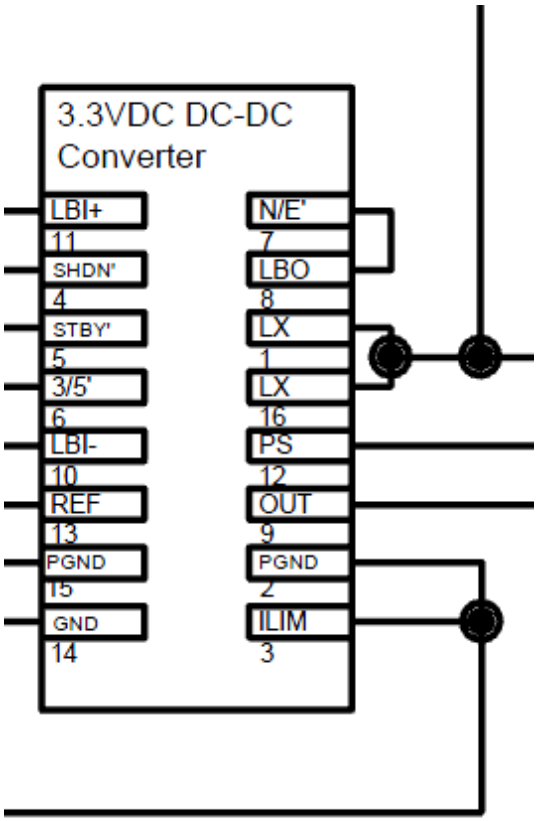

Figure 2.12-4: The 3.3 volt DC to DC converter; a MAX710 surface mounted chip

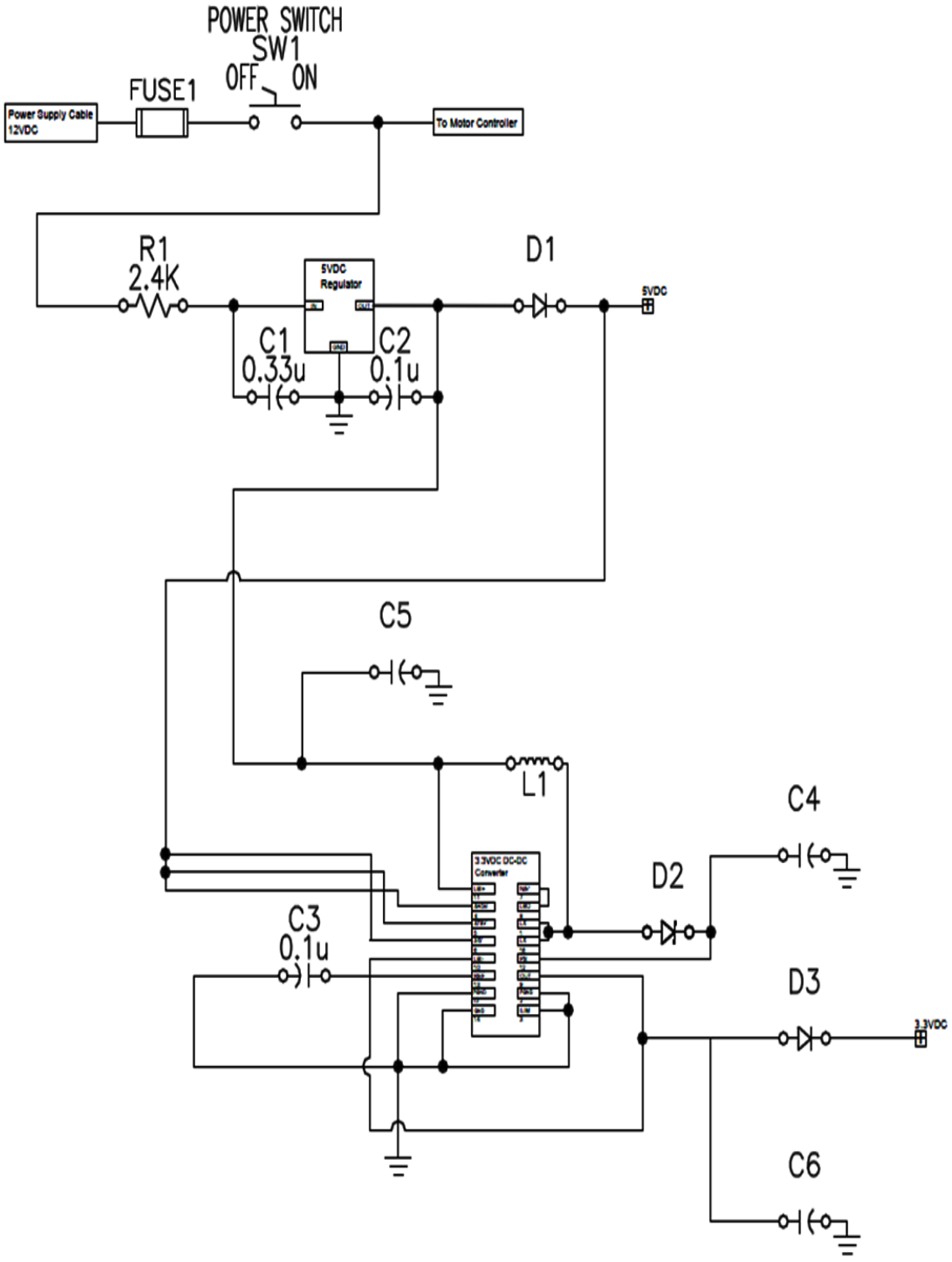

Figure 2.12-5: The schematic diagram depicting the connection of Figure 2.12-3 and Figure 2.12-4. This image was created using AutoCAD educational software.

3 Section 3 Design

3.1 Physical Chessboard

Our design requirements call for 1.5 by 1.5 inch cells as shown below in 3.1-1 and 3.1-2. When the two grave yard zones are counted, the board layout itself must be at least 12 X 21 inches. Additional margin is needed on both sides for full X axis motion. We chose to integrate both of the HMIs into the board design itself. The playing surface and the two HMIs are housed in one wood box approximately 24 x 24 x 2.5 inches. The top panel is removable for servicing and we have mounted hardware to the top panel so that they are concealed from below. Wood was chosen for cost, ease of use, and attractive looks. The box will also nicely and neatly contain all of the wiring for the LED grid. The top panel has been cut out so the plastic playing surface sheet could be installed, as well as cut so that the two LED HMI displays and their corresponding buttons could be held. The two grave yard zones are not physically marked but the picker and the software is able to place the pieces to the side where they will not interfere with the game play. We decided to go with Durex from Lowes for our plastic board and we sprayed a frosted coating onto it to allow for the LED lights beneath it to defuse as needed. The wiring harness comes out of the bottom of the box, and has adequate service loop length inside the box. When the top panel is lifted up the service loop allows for the panel to flip over or be moved over to the side for servicing.

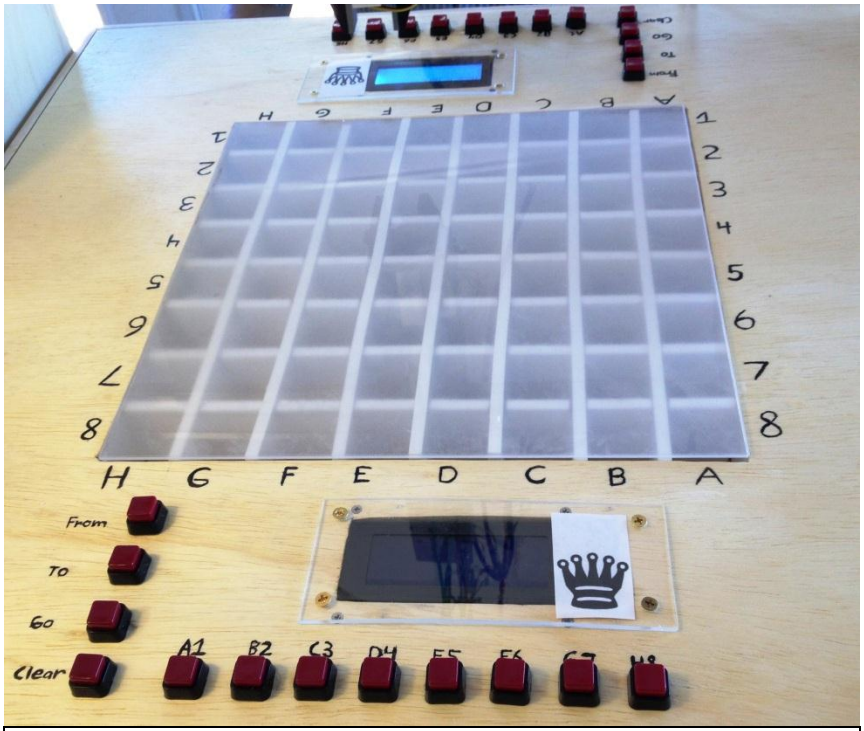

Figure 3.1-1: The actual playing grid from the perspective of HMI #2

We purchased 1.5 x 0.25 inch plastic rectangular molding strips so we could construct a vertical grid and place it under the playing surface. This grid matrix separates each cell so light will not mix with adjacent cells. Quarter inch clearance holes were drilled along the center line at 1.5 inch intervals. Then slits were cut from one side inward to the ¼ holes. The strips are interlocked together to form the grid. The interior walls will be left white to better reflect the LED light.

Figure 3.1-2: The full set up of our Interactive Automated Chess Set before the memory foam was attached to the claw prongs

3.2 Microcontroller

3.2.1 Requirements

In order to create a working robotic chess system, we needed a microcontroller or network of microcontroller and development environment with the following specs:

- Running an onboard minimalist chess engine
  - Large amount of RAM to perform AI searches
- Driving our I/O module
  - Requires around 60 I/O pins
  - Requires multiple hardware timers
- Large development community
- Easy access to prototyping boards
- Chip can be programmed on the fly
- Preferably able to code in C++

3.2.2 Design

We chose the Atmel XMEGA based on many factors. The first factor that determined the size of our microcontroller was whether we were going to offload the AI to a server. We decided against it in order to simplify the hardware and to maintain a self-contained system. Therefore, we knew that our system needed a microcontroller with a large enough memory to contain an entire chess engine. The chess module's memory footprint is substantially larger than the I/O module. This automatically eliminated the MSP430, the PIC microcontroller, and the ATMega328 microcontroller. Because of this decision, it automatically followed that we were going to need a standalone microcontroller instead of an I2C bus.

This left us between the Texas Instruments Stellaris LM4F and the Atmel XMEGA. The Stellaris MCU was a little more heavy duty; it had more pins and four times as much SRAM. However, our team preferred Atmel due to the large user community on the internet. Websites such as http://avrfreaks.com dedicate themselves to publishing code for AVR enthusiasts. Beyond that, Atmel has its own library called the Atmel Software Framework. This library allowed us to easily interface with most peripherals. Also, even though we are not using the tools directly, the open source Arduino community is based on Atmel hardware. Since the team is experienced with Arduino, we could get a good feel for our limitations with the hardware we were using. The device also had object-oriented support, although we didn't require it. In retrospect, we probably didn't need as many pins or as much memory as we got through our surface mount chip, but it's better that we overshot.

3.2.3 Programming

We decided to go with the Olimex AVR MKII Clone as our main programmer. It is a multi-programming tool with the ability to power the board's programming and the ability to choose between 5v or 3.3v targets. There are no debugging capabilities using the AVR MKII as it is entirely a PDI programmer. We programmed it essentially using a two-wire PDI programming protocol via the PDI breakout pins on our board. We also bought a second AVR MKII clone called the ZeptoProg II. This device was a lot less sturdy, but it had more features. It could also function as an oscilloscope and logic analyzer via a software package that comes with it; because of that, we found the device useful but ultimately still used the Olimex to program. We decided against buying a real AVR ISP MKII because they are at least twice as expensive and some of the older models cannot program using PDI.

3.3 Software

3.3.1 Requirements

Our software system is rather large, so our requirements differed from those of a typical AVR project such as a mobile robot. We have many components to work with so separation of concerns took priority. We needed an architecture that was both lightweight and powerful. The requirements we used were as follows:

- Modular design to ensure separation of concerns and a debug capability
- Message Passing Controller architecture for further ease of development
- Minimalist enough to run using 8K of SRAM.
- Integrated test suite

3.3.2 Software Architecture

Our software system consisted of several modules with a message passing architecture. In our case, the main module is the controller of the system. It interfaces between the chess module and the I/O module. The chess module is the model of our system. It contains internal representations of the chessboard and runs the AI processes. The I/O module contains device drivers that interface to each electronic subsystem.

We considered creating a structure called ChessMove or ChessLocation. It would have allowed us to keep a representation of chess moves that is abstracted from the I/O devices and chess engine. ChessMoves or ChessLocations could be passed between the I/O module, main module, and chess module. We decided against this because we ended up using many different representations for chess moves. We used a string-based chess-move representation, for example "a2a4". We also used absolute board positions to specify crane movements, for example {0,0} -> {13, 2}. There are a total of 14 board positions that the crane can move in the x direction, which includes both graveyard squares. We also had to use a completely different (1-based, such as {8,8}) system for specifying LED locations.

Since we ended up with three systems for location specification and there were only two locations in code where the conversion is made, it would have been over-engineering to create a system for converting between the different representations.

The next module we created was the chess engine. It contained several global variables to control the interface to the outside world. Side is the side that is currently moving. Move is defined as the move being input or output from the engine. PromPiece is the requested promotion piece, in our case it is always queen. MaxDepth controls the search depth limit, and Post is our debugging flag. The engine contains a function to start the engine, called InitEngine. The next function, InitGame, starts a new game by setting the side variable to the starting side. The function Think causes the AI to think up a move from the current position. The move is stored in the Move variable. The DoMove function performs the move currently in the Move variable and toggles the current side of the engine. The ReadMove function converts an input move into the type of move used by the engine. PrintMove prints the current move to standard output. In our case, it sends it over serial which we will be debugging through. The Legal function checks the move stored in Move for legality. The ClearBoard function empties the entire board, and the PutPiece function is called to put each piece on the board. We renamed DoMove to player_move and created a new function called ai_move that encapsulates the AI processes to create a cleaner interface to the main program.

The next module we created is IO module. It contains all of the functions that interface to the individual electronic subsystems, such as the sound subsystem, the LED controller, the Hall Effect grid, and the motor controller. The motor controller is further abstracted into a composition of stepper motors and other motor control equipment. It has the internal functions, move crane, move stepper, and activate clamp. The only function that it exposes to the IO controller is move piece. All of the classes discussed above can be found in Figure 3.3.2-1 below.

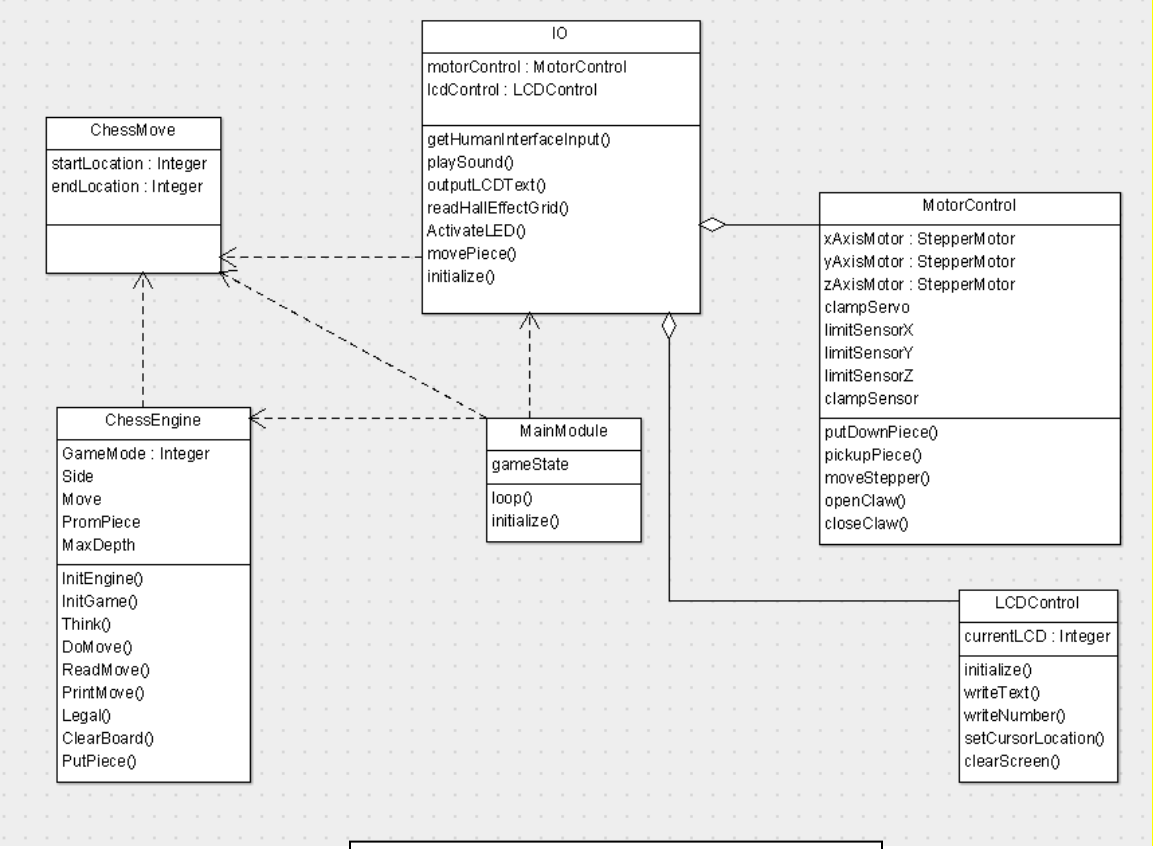

Figure 3.3.2-1: Class diagram.

3.3.3 Main Module

3.3.3.1 Requirements

The main module is the controller for the entire application. As such, messages must be passed back and forth between it and the I/O and chess engine models. We chose this method because it clearly separates the responsibilities of each piece of our program. Since we are using a single microcontroller, we don't have physical boundaries between components and thus it is beneficial to separate them in the software.

- Main module is the message passer in message passing architecture
- Uses only high level method calls to describe process
- Keep track of game mode: AI vs. Human, AI vs. AI, and Human vs. Human

3.3.3.2 Process Flow

The description of our software process begins with the main loop. We begin by calling initialize. This calls the initialization routines for our I/O module, chess engine, and the internal state of the main module. We will assume that the pieces are set up properly on the board. We wait for the user to press the start/reset button before beginning our game loop. At this point, we run the white player's turn followed by the black player's turn until the game ends. We will have different processes for each turn depending on whether white is an AI or human. Our goal is to have human vs. human, human vs. AI, and AI vs. AI abilities. The easiest and first candidate for our game will be human vs. AI and so that is what we will focus on first; a diagram depicting this flow can be found below in Figure 3.3.3.2-1.

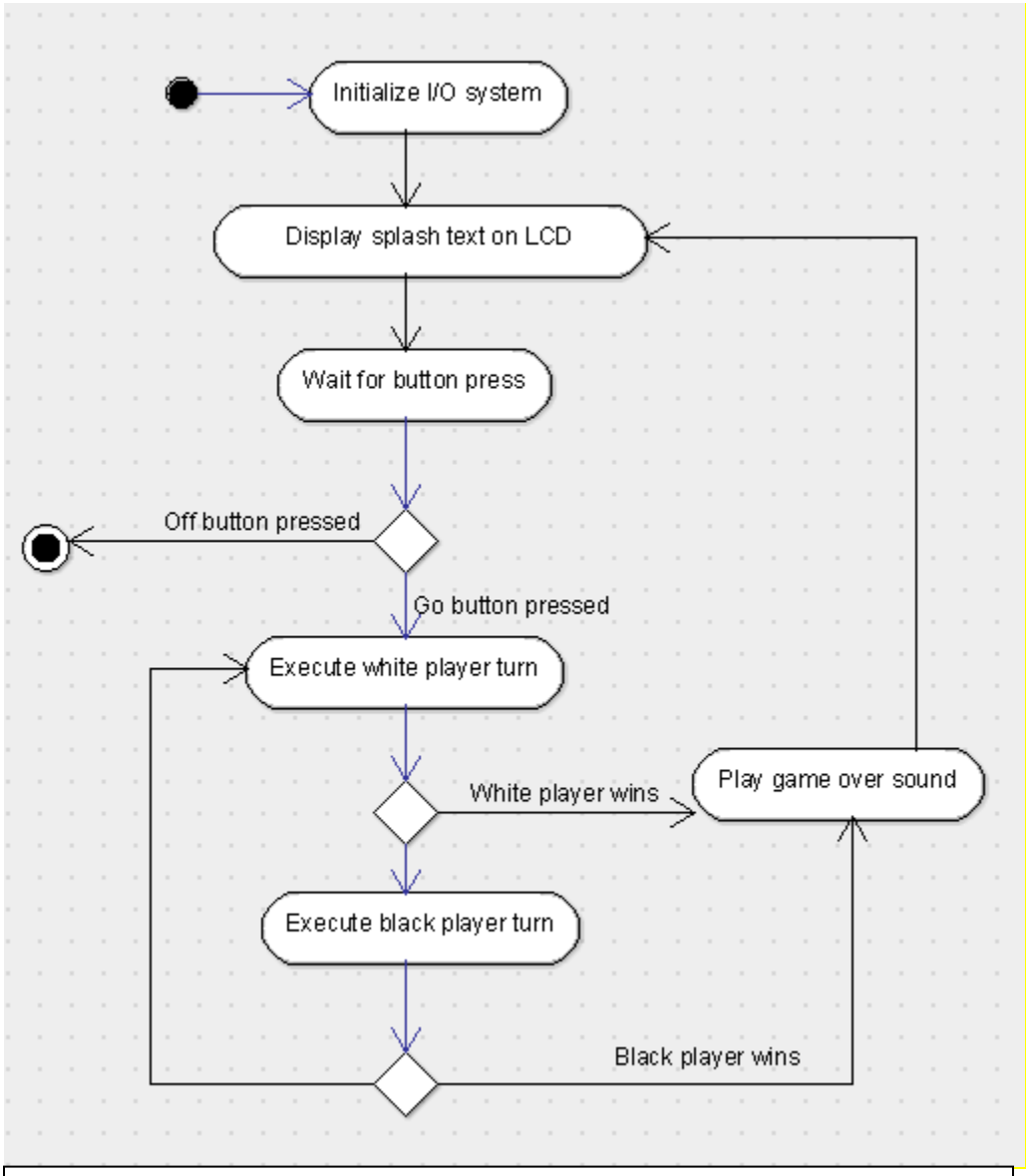

Figure 3.3.3.2-1: Main module activity diagram

3.3.4 Player's Move

The figure below (Figure 3.3.4-1) demonstrates the process for message passing in the case of executing a valid player controlled move that captures no pieces. In this case, the main module must send a message to the I/O module prompting the LCD screen to display a message to the user. It will repeatedly poll the I/O module for a player move. The module will wait until the player has typed in a correct move and send it back to the main module. At this point, the main module must send a message to the chess engine to move the player. The chess engine will return the result of the move. This will normally be one of three options

1. The player move took an opponent's piece
2. The player move was valid and the board is updated
3. The player's move was invalid.

If result 1 was returned, the main module must inform the I/O module that it must send the piece to the graveyard. At this point if result 1 or result 2 was returned, the main module must inform the I/O module that it must move the piece. This includes sending a message for the motor controller to pick up the piece and a message for the motor controller to move the piece to a certain location. If result 3 was returned, the main module loops back to the beginning of the process and plays an error noise. It also will display an error message.

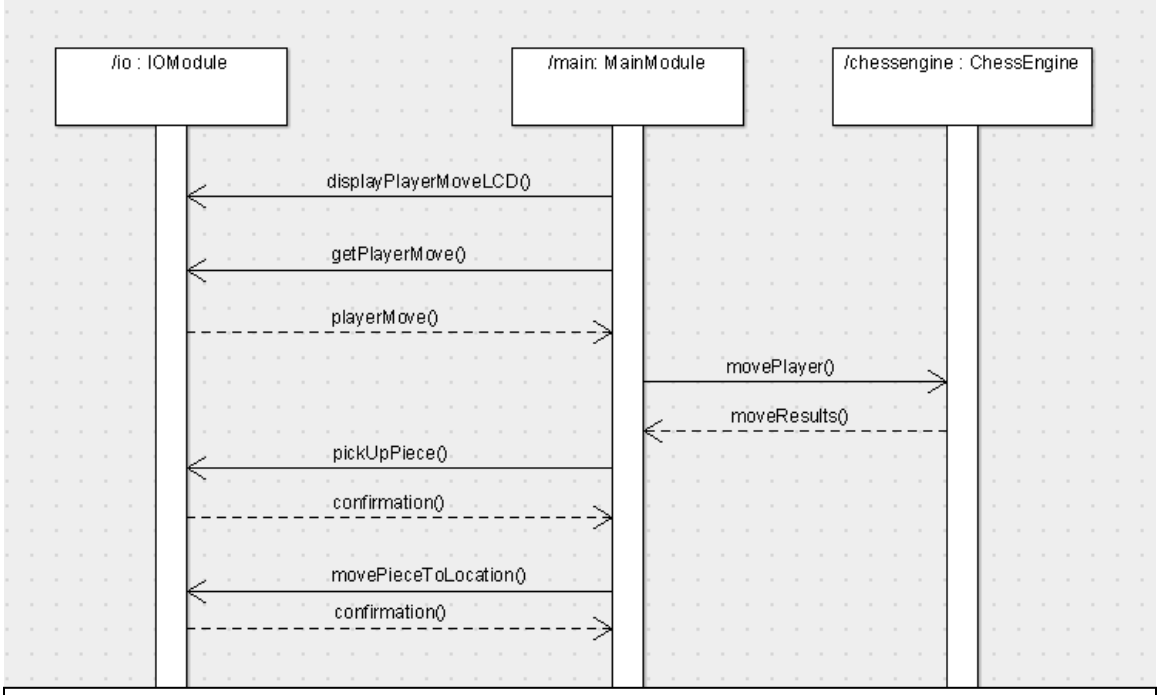

Figure 3.3.4-1: Sequence diagram of a single player's turn for a single move with no captured pieces

3.3.5 AI Move

The figure below (Figure 3.3.5-1) demonstrates the process for message passing in the case of executing a move based on the AI engine. First, it will poll the engine for a move. The engine will decide its move and return details about the move including the start location, end location, and the location of the captured piece, if applicable. At this point, if a piece has been captured, as in our example, we will send a message to the IO module to pick up the captured piece. Once the I/O module confirms the pickup, we send it a message to move that piece to the graveyard. Once that action is confirmed, we send a message to the crane to pick up the proper piece. Finally, we send a command to move the piece to the destination location.

If any of the confirmations comes back positive, we alert the user of the mechanical system malfunction with a motor controller error code we have defined in the section “Motor Control” and await a “GO” button press to continue.

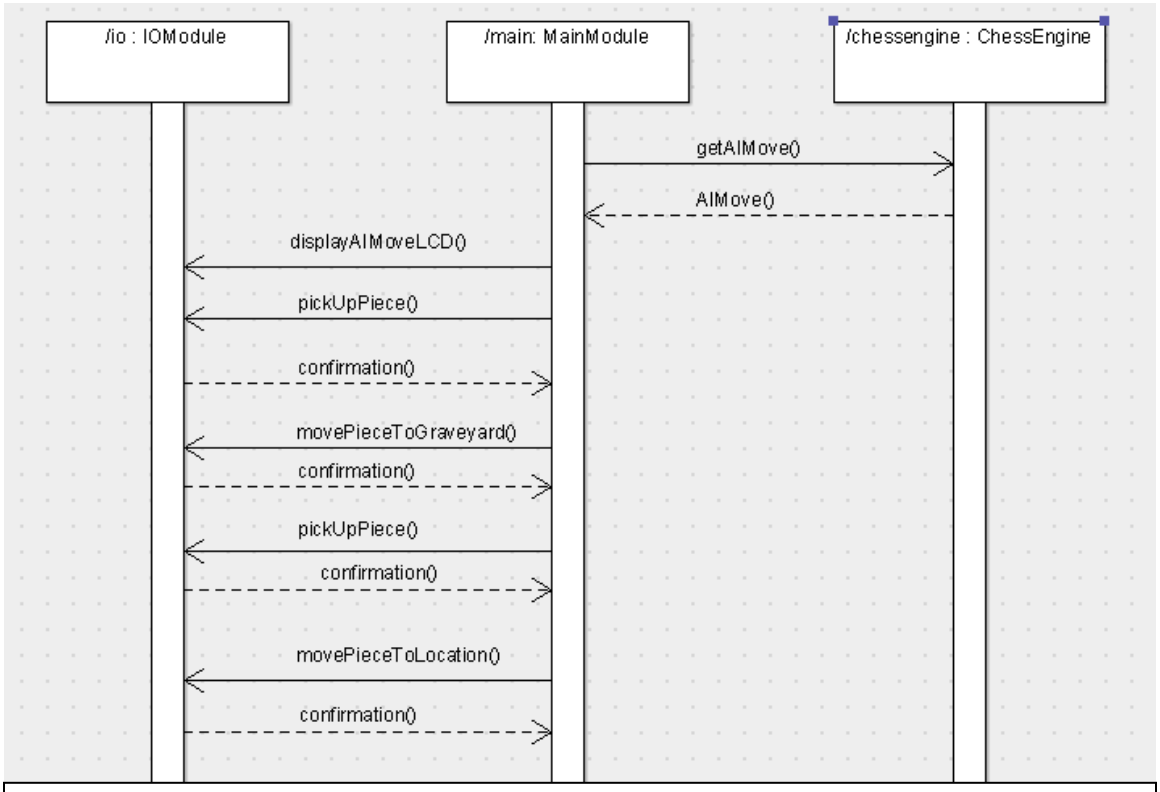

Figure 3.3.4-1: Sequence diagram of an AI’s turn for a single move with a captured piece

3.3.6 Chess Module

3.3.6.1 Requirements

The chess module contains an engine as well as an AI engine. Our chess module needs to take up around 4KB of SRAM so that we have enough memory for the I/O module with some to spare. This means that not many data structures can be stored in memory such as large hash tables or heaps. Our chess module needs to be customizable to allow for player vs. player, player vs. AI, and AI vs. AI matches. The most important mode is player vs. AI, so we will opt for systems that excel at that and then move toward those that can do the other two. Our chess module should require the least amount of hardware possible.

3.3.6.2 Design

We decided against offloading our AI to a server. Having an elegant AI was not the highest priority on our list. Since the game is meant to be fun, having a master level AI is overkill. Therefore, we don't need as much processing power. Also, a large hardware and software difficulty would be incurred in designing a module that connects to a remote server. This would add to our I/O module while subtracting from our chess module. We would have to worry about interfacing with a standard chess protocol, which would require setting up an interface on our microcontroller. Also, we would still have to set up an engine on a server, which would mean more programming on a computer. The biggest issue with putting the engine on the microcontroller is the questionable ability to debug it.

We had a close match in choosing which chess engine to use, but in the end, we decided to use the Micro-Max code. The code is open source, terse, and takes up very little program memory. Since it is well documented and it has already been ported to AVR controllers before, the time cost should be very low. The runner up was a self-implemented system. A self-implemented chess engine would have a decently low time cost, be very flexible, and we could tailor it to our resource requirements. With the Micro-Max code we can attempt to tailor it to our resources, but there is a possibility it will be too large. The self-implemented system would be more prone to bugs. It would certainly be more minimal in functionality; we would not implement en passant or castling. There is also the possibility of biting off more than we can chew if we chose to implement the whole engine. So to be safe, we chose the Micro-Max.

Andre Adrian's Micro-Max port to Atmel ATMegas will be used as a starting point. It disabled the hash table to reduce memory usage and it fused the logic to a standard Winboard driver interface, which is much easier for us to work with. This interface will form the interface for our entire chess module, which relieves us of some programming burdens.

The biggest difficulty will be hacking the code to work with player vs. player and AI vs. AI mode. In all probability, we only implement player vs. AI functionality in our system. In order to implement player vs. player mode, we need a way of checking the legality of the user-input moves. We also need to keep track of the state of the board. When the different functions of the chess interface are called, internal logic must make sure they are valid to be called within the game mode.

3.3.6.3 Data Structures

The chess engine uses a few simple data structures to organize the data of the application as shown below in Table 3.3.6.3-1 and Table 3.3.6.3-2. Pieces are encoded as 8-bit integers, which are in reality bit patterns. Setting the 8-bit indicates a piece is white, and setting the 16-bit indicates a piece is black. This way it's simple to check if a piece is white or is black; p&8 == 1. The 32-bit indicates that a piece is in the original position, which is used for checking whether we can castle. This bit is called the "virgin" bit The three lowest bits are used to encode the piece type. White pawns and black pawns are different pieces since they move differently on the board. One moves "upstream" while the other moves "downstream".

Bit	7	6	5	4	3	2	1	0
Meaning	Unused	Unused	Virgin bit	Black flag	White flag	Piece MSB	Piece	Piece LSB

Table 3.3.6.3-1: Mapping of bit positions to meaning in piece representation

Least significant 3 bits	**Piece**
000	Empty space
001	Upstream pawn
010	Downstream pawn
011	Knight
100	King
101	Bishop
110	Rook
111	Queen

Table 3.3.6.3-2: Mapping of least significant 3 bits to pieces in piece representation

The squares of the chessboard are designated using the ‘0x88’ system, which means that the board has 8 ranks of 16 squares as shown below in Table 3.3.6.3-3. Only the first 8 squares of each rank are valid. We use a one dimensional array to keep track of the entire board. We put each rank of the board in the array in sequence. This is a “hack” to make the 4 lowest bits indicate file while the higher order bits map to the rank. Our array keeping track of the board is 16 * 8 = 128 bytes wide. This representation is called 0x88 because if we bitwise-and a location with 0x88 and the result is nonzero, we know the piece is invalid. This is because if the 0x80 bit is set, the rank of the piece is invalid and it has fallen “off the board”. If the 0x08 bit is set, the file is invalid as well due to the layout of our memory.

Location	Rank No.	File No. (x = invalid location)
0x00-0x0F	1	12345678xxxxxxxx
0x10-0x1F	2	12345678xxxxxxxx
0x20-0x2F	3	12345678xxxxxxxx
0x30-0x3F	4	12345678xxxxxxxx
0x40-0x4F	5	12345678xxxxxxxx
0x50-0x5F	6	12345678xxxxxxxx
0x60-0x6F	7	12345678xxxxxxxx
0x70-0x7F	8	12345678xxxxxxxx

Table 3.3.6.3-3: Layout of board array

3.3.6.4 Initialization and Interface Functions

The chess engine will be initialized by calling the initEngine and then initGame functions. The initGame function will be passed an enumeration specifying the game mode. A mapping of game modes to their enumerated numbers is in the table below. initGame will call clearBoard and putPiece to set the pieces on the game board for a new game. The game mode enumeration as well as the chess interface funtions and required game modes to call can be found below in Table 3.3.6.4-1 and Table 3.3.6.4-2 respectively.

Game Mode	Number
Player vs. Player	0
Player vs. AI	1
AI vs. AI	2

Table 3.3.6.4-1: The game mode enumeration

Function	Game Mode Required	Description
InitEngine	0, 1, 2	Program start-up initialization
InitGame	0, 1, 2	Initialization to start a new game
Think	1, 2	Think up move from current position and stores it in Move variable
DoMove	0, 1, 2	Perform the move in Move and toggle Side
ReadMove	0, 1	Convert input move into engine format
PrintMove	0, 1, 2	Print Move on standard output
Legal	0, 1	Check move for legality
ClearBoard	0, 1, 2	Make board empty
PutPiece	0, 1, 2	Put a piece on the board

Table 3.3.6.4-2: The chess interface functions and required game modes to call

3.3.7 I/O Module

3.3.7.1 Requirements

Our I/O module should have a one-to-one mapping of functions to physical devices. This means that every device should have its atomic actions (moving a servo, moving a stepper, reading the Hall Effect grid) mapped to software. For more advanced devices such as the motor controller, a single function can perform an entire process. As an example, the motor controller should have functions for more complex actions such as picking up a piece at a certain location. The interface should be high enough level that the controller module does not have to know anything about the underlying implementation of the hardware.

3.3.7.2 Initialization

In order to use the I/O module, we must first initialize it. This consists of playing the game start sound to test the audio driver, initializing the LCD display to two

line mode for text output, and blinking the LED matrix. We will also move the motors to the origin in case they have been moved while the machine is off.

3.3.7.3 Motor Control

Our motor control system consists of a self-contained module that interfaces with the three motor-driving servos, the clamp servo, clamp feedback sensor, and motor overshoot sensors. In order to drive the motor, we must begin by moving to the origin. This ensures that compounding error does not stack up with our motors. We don't have highly accurate encoders, so it's important that we can track our exact position as closely as possible to avoid messing up the state of the game board. To move to the origin, we serially activate the X and Y axis motors, waiting for each one to hit a limit switch before we kill power to it. We could have driven the motors simultaneously, but that would involve trickier programming. If we get the serial motor control down, we can save parallel motor control as a future improvement.

Once the motors are both at the origin, we begin by moving the X axis motor to our destination. In order to find the exact distance to move, we have to figure out the total length of the board as well as the total distance travelled per stepper motor pulse. The distance will have to assume that the motors don't slip. If at any time during this move the limit switch trips, it means that we overshot our destination and our motor controller should return an error. When our motor controller returns an error like this, it is up to the microcontroller to re-issue the command to move. The motor will return motor error code M4, overshoot.

Once the X axis motor is at the proper location, we move the Y axis motor in the same way. If it trips a limit switch, we return the error code M4. At this point, we should begin to open the clamp. Once again, if we are able to drive multiple devices concurrently we will. If not, the next thing that needs to happen is the opening of the clamp via the servo motor. Once that is done, we drive the Z axis motor downwards a predefined amount. The Z axis motor is the same as the X and Y axis motors, with its own limit switch. The limit switch is designed to trip before the claw hits the board, as this could knock over a piece. If this switch is hit, we return error code M4. If the switch does not trip, we will once again activate the clamp until the clamp sensor feeds back positive. If it does not send back a positive signal, we return motor code error M1, cannot pick up piece at location (X, Y). At this point, the motor will either send the piece to the graveyard or send it to another square on the chessboard. In essence, they are the same motion. First, the Z axis motor will reverse itself until it is at its zero point. This point will be governed by a limit sensor tripping. Once that is true, it will attempt to return to the origin point. It does this by moving the X and Y axis motors serially, checking for limit switches. If at any time during this motion, the clamp sensor returns a false reading, we will continue to poll it for a couple more readings. If it seems that the piece is lost, error code M2 or M3 will be returned, depending on whether it's moving to the graveyard or a location on the board. If

the piece is not lost, the motors will move to the desired spot and deactivate the servo.

The previous discussion demonstrates why we must have two internal motor control functions: one for picking up a piece from location (X,Y) and one for putting down a piece at location (X,Y). When picking up a piece, we must open the clamp before moving down on the Z axis. When putting down a piece, we must open the clamp after moving down on the Z axis. The overall process diagram for picking up a piece can be found below in Figure 3.3.7.3-1.

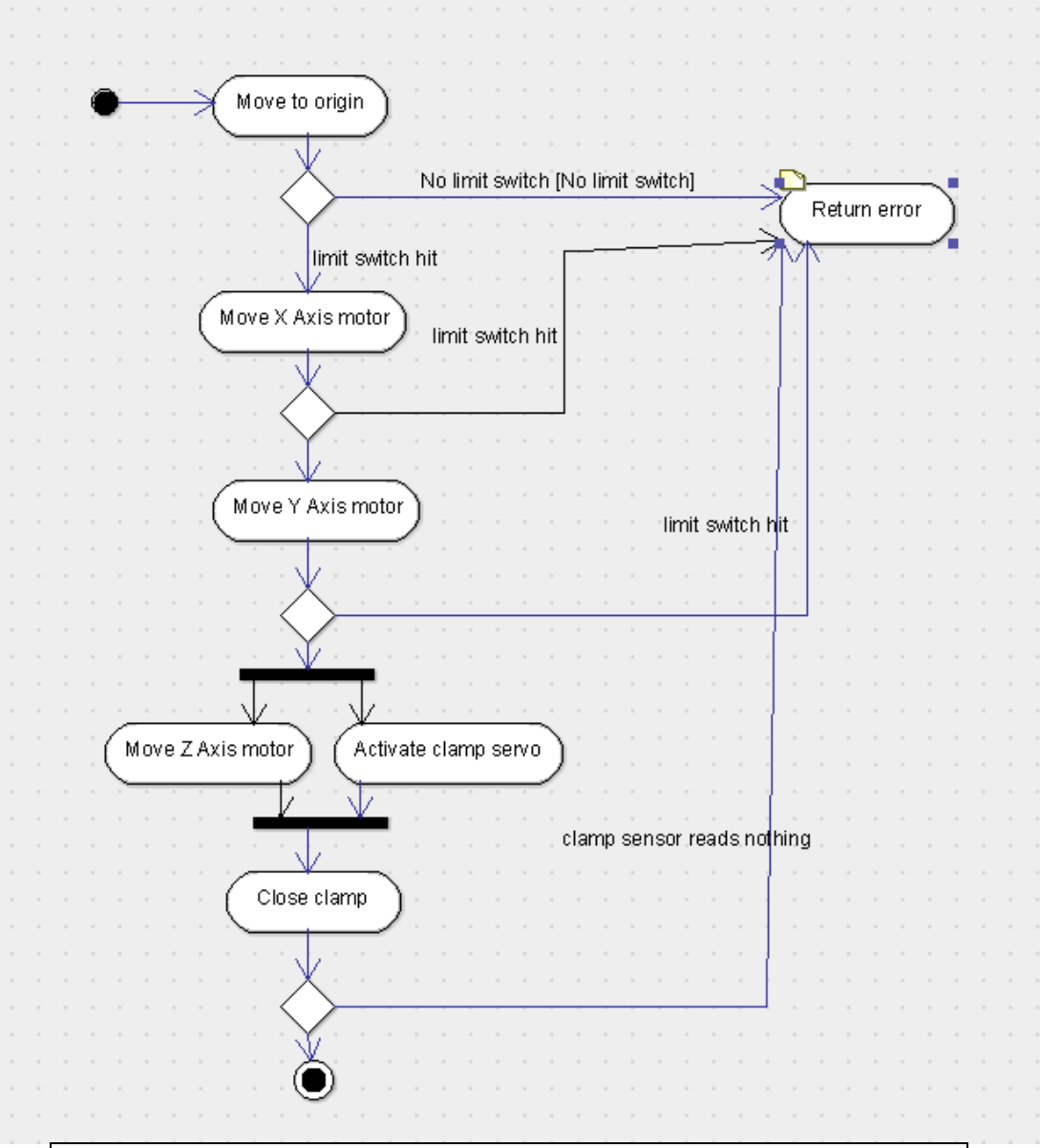

Figure 3.3.7.3-1: Process diagram for picking up a piece

3.3.8 Error Codes

Error codes will be vital when troubleshooting during the testing phases of the Interactive Automated Chess Set and after. Some of the error codes we plan on using can be found below in Table 3.3.8-1.

Error Code	Description	Method of Detection
M1	Unable to pick up piece at location (x, y)	Clamp feedback sensor
M2	Unable to move piece to location (x, y)	Clamp feedback sensor
M3	Unable to move piece to graveyard	Clamp feedback sensor
M4	Motor overshoot	Motor overshoot sensors

Table 3.3.8-1: The error codes, their meaning, and method of detection

3.4 User Interface

Our Design Specifications and Requirements call for two user interfaces, HMI # 1 and HMI #2, located directly opposite of each other. Our philosophy is to keep the design simple, practical, and cost effective. We clearly considered all the possible user interactions and tried to make our product as user friendly as possible. After considering our options we decided on a panel with twelve individual push buttons, utilizing momentary normally open switches, and a LCD screen display was the both the easiest to design and implement as well as the most user friendly. If we had decided to use individual switches, round mounting holes would have been necessary; cutting out twenty four square holes with a Dremel type tool would have been a time consuming and tedious task. Even though the switches shown below in Figures 3.4-1 through Figure 3.4-7 (and Figure 2.4-1 in the research section) have a square outline, the mounting holes are round and easy to drill. The layout shows the four function switches, the eight zone selection switches, and the twenty digit by four line LCD display. Our HMIs are designed for user clarity and simplicity. To reduce the total I/O count the user is asked to push the letter of the destination X coordinate then select the number of the Y coordinate allowing the same eight switches to be utilized for both functions.

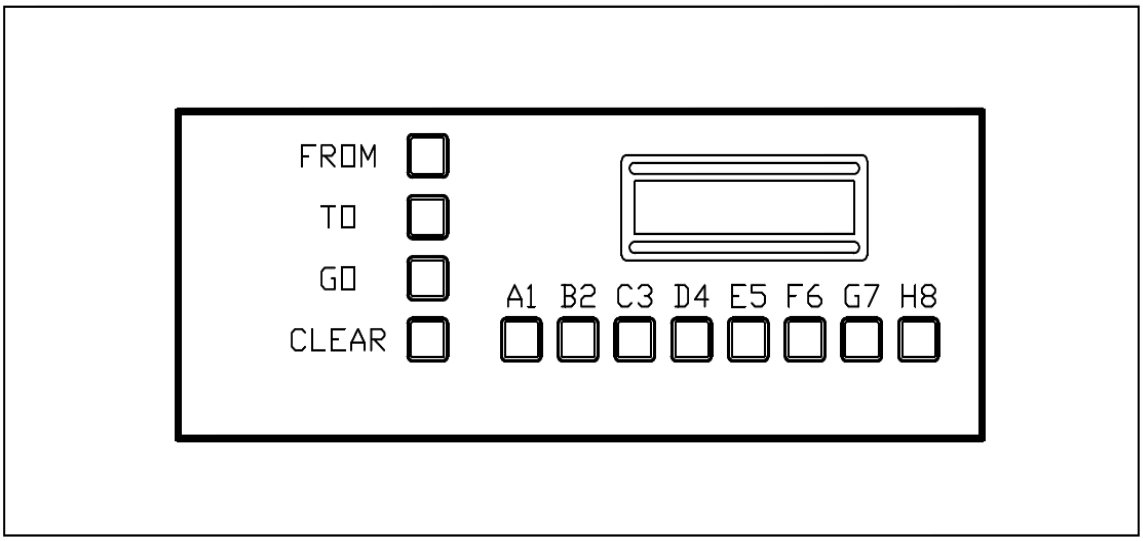

Figure 3.4-1: The HMI panel design for the Interactive Automated Chess Set. This picture was drawn using AutoCAD educational

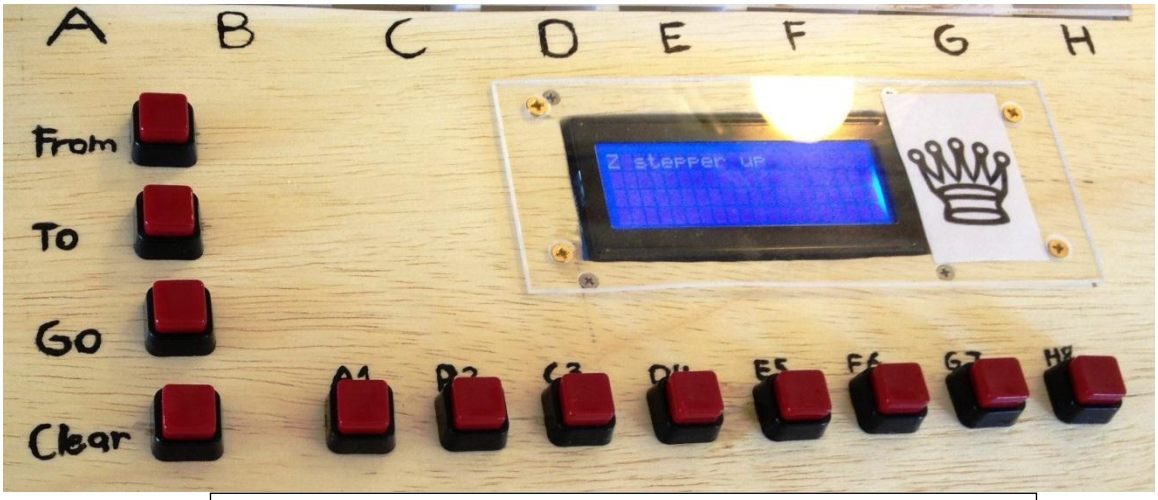

Figure 3.4-2: The actual HMI #1 panel and buttons for the Interactive Automated Chess Set

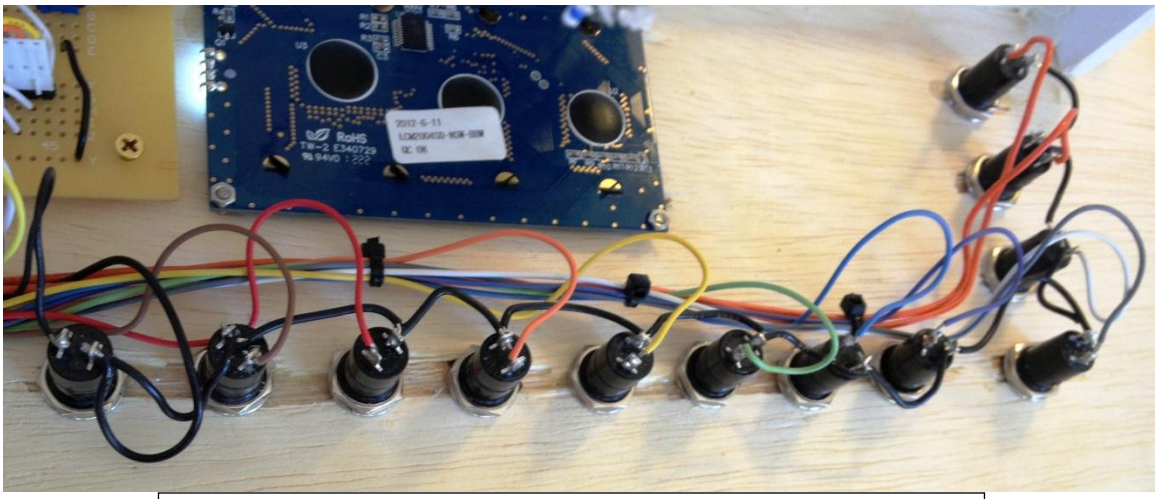

Figure 3.4-3: The underside and connections for the actual HMI #1 panel and buttons for the Interactive Automated Chess Set

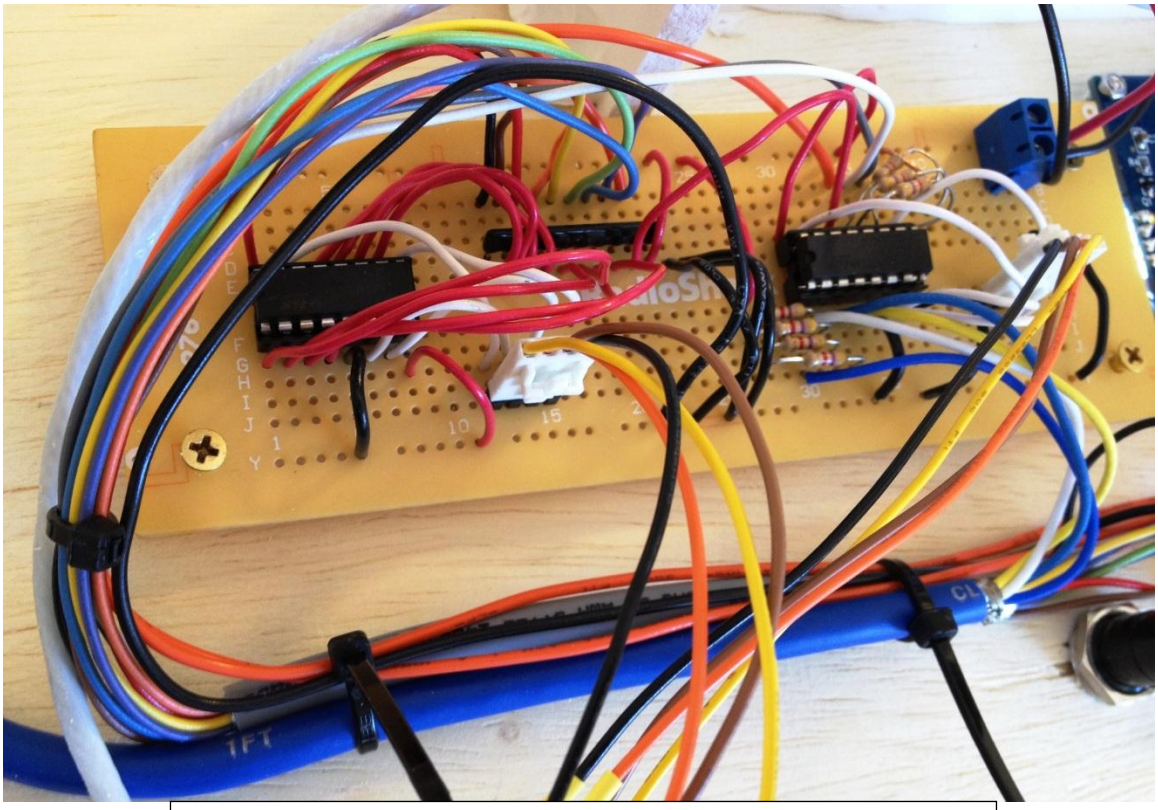

Figure 3.4-4: The circuit for the actual HMI #1 panel and buttons for the Interactive Automated Chess Set

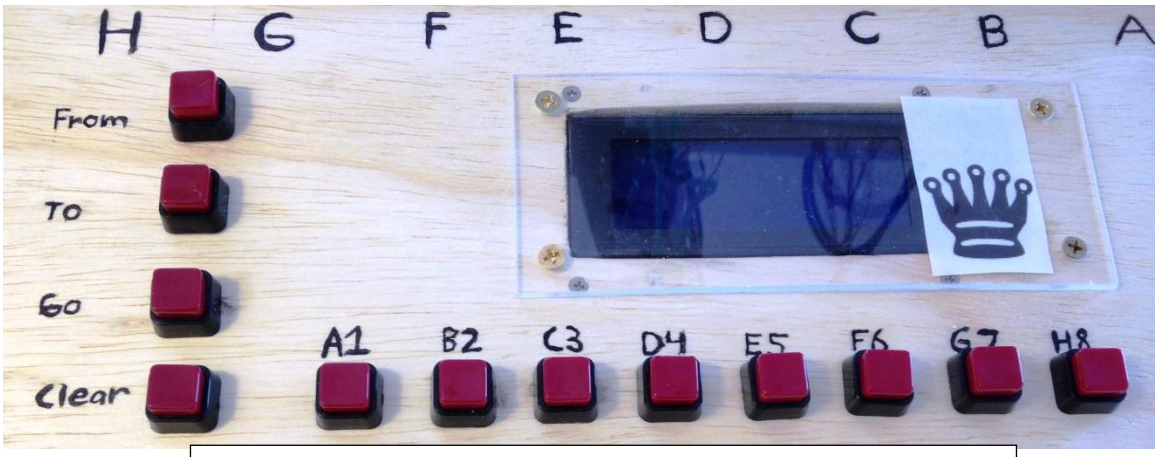

Figure 3.4-5: The actual HMI #1 panel and buttons for the Interactive Automated Chess Set

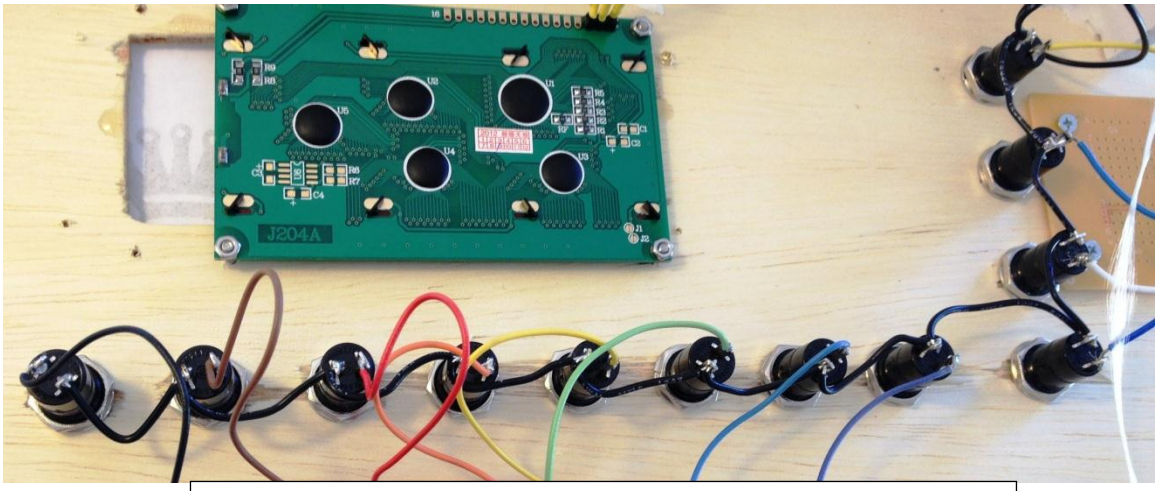

Figure 3.4-6: The underside and connections for the actual HMI #1 panel and buttons for the Interactive Automated Chess Set

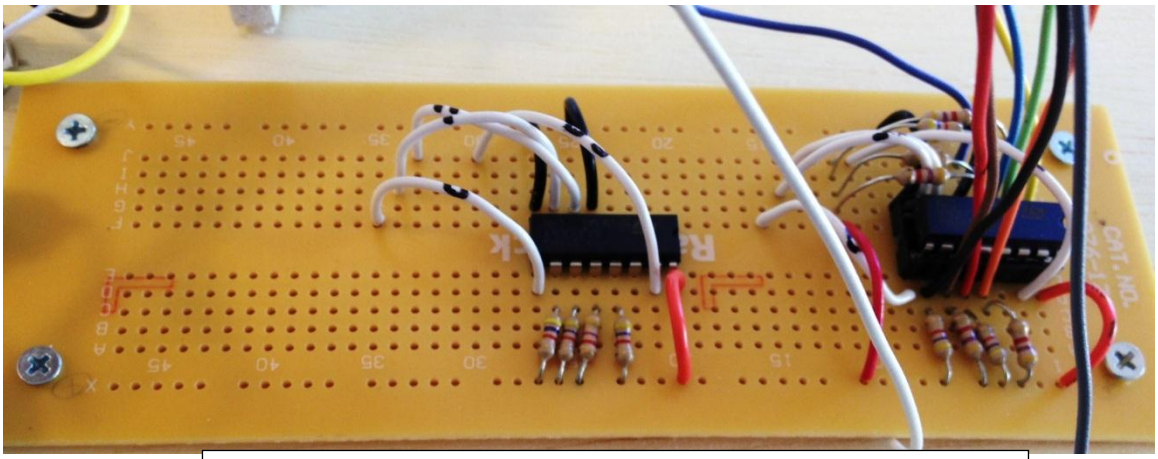

Figure 3.4-7: The circuit for the actual HMI #1 panel and buttons for the Interactive Automated Chess Set

Originally our plan was to use normally closed switches, but finding the normally closed alternative at the same cost just was not an option. The solution to this logic dilemma was to simply use a 4.7KΩ pull up resistor configuration. When the switch is pushed it will connect the input to DC common, and the 5 VCC in now dropped across the 4.7 KΩ resistor, resulting in a .01 mA current flow. As per the spec sheet with a 6 volt input the input current is 1µA, so therefore the input Impedance is around 6MΩs. Figure 3.4-8 shows how the 4.7k resistor networks are electrically connected at the I/O input. We used an eight channel 16 pin DIP resistor network to reduce components, and simplify the hardware layout.

Even after reducing the switch count to twenty four inputs, the I/O count is still a problem; this allowed us an opportunity to figure out a creative solution using multiplexers to attempt to reduce the I/O count. Our solution was to use three ST Microelectronics M74HC148 eight to three Line Priority Encoders. This cut our total I/O count for the HMIs down to thirteen from the original twenty four. This action would also have the effect of a hardware interlock for unwanted switch inputs. One of the three encoders is connected to the eight HMI # 1 zone input switches while the second encoder is connected to the HMI # 2 zone input switches. Finally the third encoder will interface with each of the four function switches from both HMIs. The line priority encoder is a device that, after being enabled, latches the first input to go to a low level state. It then converts the input number to BCD thus the 8 to 3 multiplexer designation. The microcontroller knows when it wants user input from one of the HMIs and will enable the appropriate encoder IC; the microcontroller also knows when the user has made their selection and will then pull the priority flag output (GS) to a low logic level. Each of the three 8 to 3 MUX integrated circuits has its own GS output, but we chose two or the three together with diodes. One microcontroller input is used to monitor the priority flag status; in the event the user makes more than one selection at any given time, such as pushing Go and Clear at the same time, the encoder will output the higher of the two BCD addresses. If the user has entered an unwanted selection the user can use one of the function switches to go back and correct the input selection; this is the advantage to prompting the user with the LCD screen. If the second user pushes one of the zone switches when not being prompted nothing will happen because that specific encoder will not be enabled at the time. Our design also mitigates the switch bounce condition. Figure 3.4-8 below shows the I/O interface with just HMI # 2. Reference the Appendix for a full wiring diagram of the entire project. In Figure 3.4-8 the first four I/O input lines going to the top 8 to 3 MUX are for the four function switches of HMI # 2. For each MUX the one Enable and the three microcontroller BCD discrete inputs are shown on the full wiring diagram. Microcontroller I/O points can also be referenced in the I/O List.

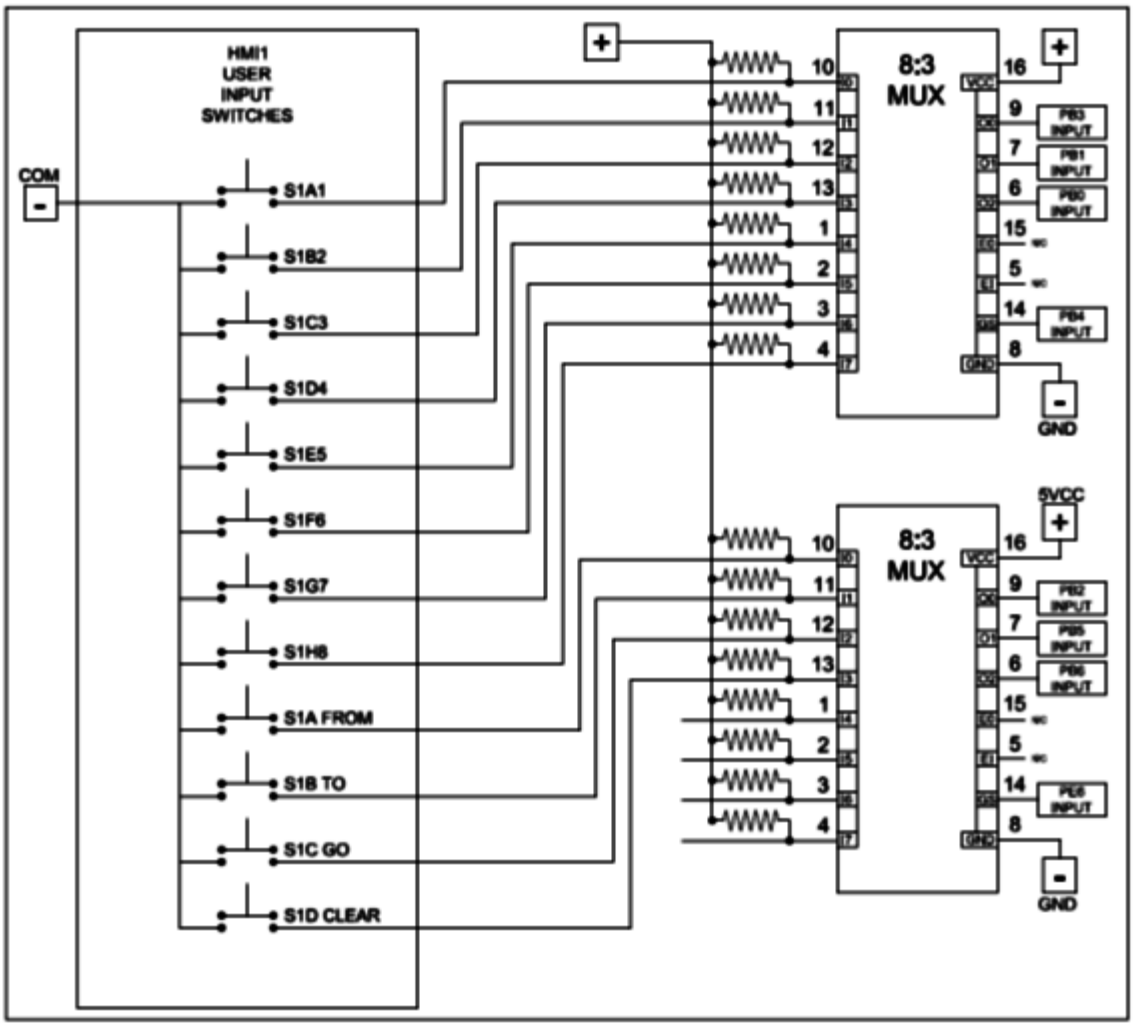

Figure 3.4-8: The design for the 5KΩ resistors and how they are electrically connected to the I/O panels. This picture was drawn using AutoCAD educational

Each of the two LCD screens require six I/O lines; four lines for data bits, one line for the reset, and one line for the enable. In addition to the I/O lines a 10KΩ potentiometer is used to control the back light intensity level. Most LCD displays also have a read write input, but a number of examples shown online did not have the read write input connected. I believe the reason is because there is not much need for the microcontroller to read from the display.

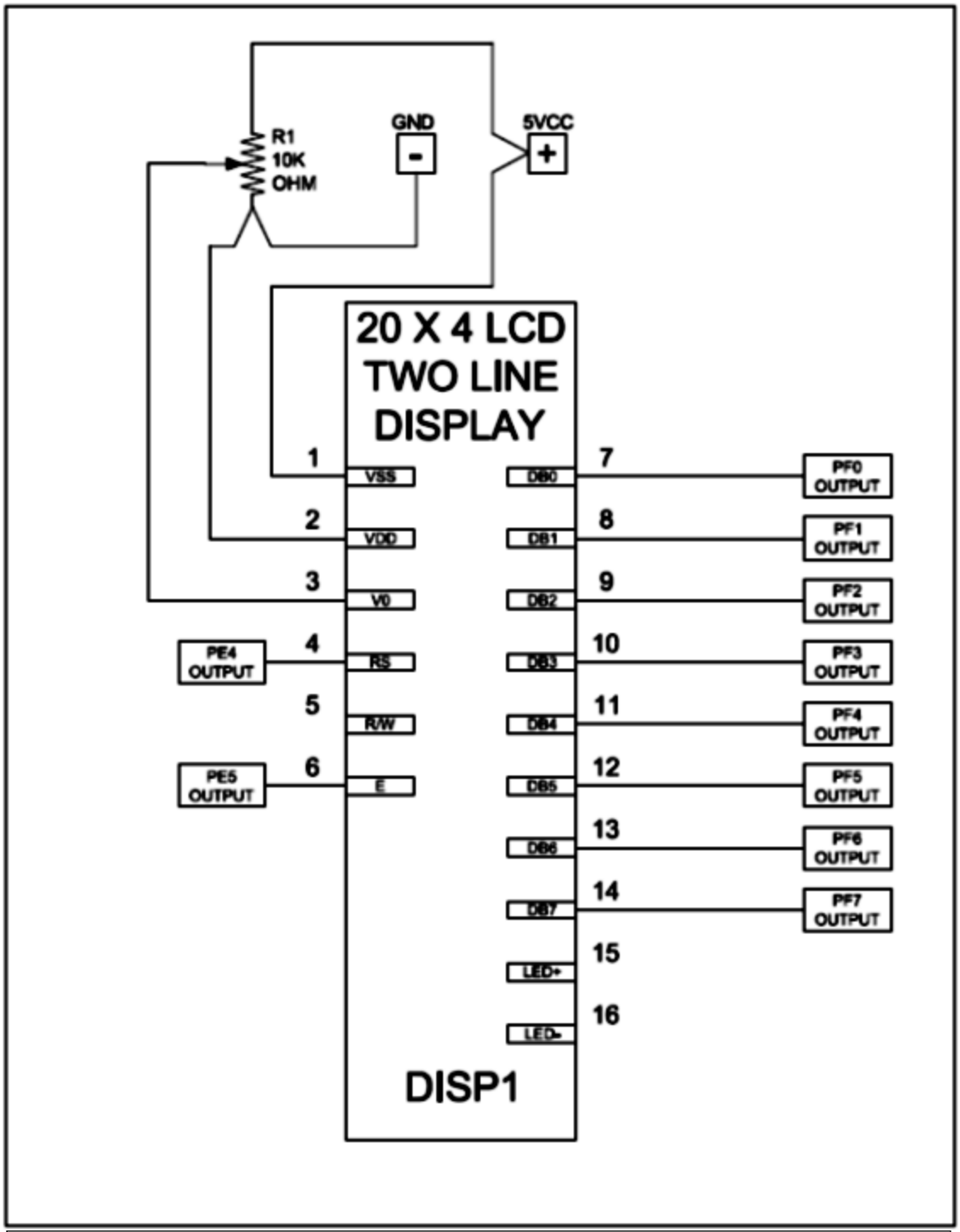

Figure 3.4-9: The electrical connections for the HMI # 1 user LCD twenty digit by two line display. This picture was drawn using AutoCAD educational software.

The LCD displays were one of the first priorities to have working in the development and test phases. Good messaging will helped when things went wrong and allowed us to trouble shoot the problems much easier than trying to

guess would have been. Figure 3.4-9 above shows the electrical connections for the HMI # 1 user LCD twenty digit by four line display. The display is configured for four bit operation, but can be configured for eight bit operation. We chose to go with the four bit operation to reduce the total I/O count. We mounted the 10KΩ potentiometer under the HMI due to the fact that once the intensity adjustment has been made there should not be much variation. Each of the characters displayed on the screen are represented by a two digit hexadecimal number that is cross referenced over to the ASCII table. If the programmer wished to print a capital A the ASCII hex code is 41. Suppose the display was to be operated in eight bit mode, then the 41 can be sent at the same time. In four bit mode the 4 must be sent and then the one must be sent. With the high speed operation on the microcontroller we do not expect this to be a problem though if speed had become a problem we could have always multiplexed the data to one HMI at a time.

Please note that the HMI #2 is for display purposes only at this time as we did not have the time to get the Player vs. Player function up and running but all of the physical components are there and ready and simply waiting on the programming component.

3.4.1 LCD Interfacing with the AI

There are two LCD displays that we needed to interface with. In our research section, we discussed the low level details of interfacing with an LCD device and came to the conclusion that it is quite burdensome. In lieu of this, we searched for a library for AVRs and found that Extreme Electronics provides such a library. This library, “liblcd”, was modified for our project to work with the two displays instead of one. We created a class, LCDControl, which contains the functions described in our research section regarding the LCD controlling library. It also contains a member variable that keeps track of which LCD we are controlling, called currentLCD. This variable can be set to either Player 1 or Player 2, and it will change the mapping of pins the controller is driving.

3.4.2 Human Input Panel Interfacing with the AI

We simply implement one function for the human input panel interface. getHumanInterfaceInput blocks the system and induces a polling loop for user input. Whenever a user presses a button, we issue a call to the LCD interface to display recognition of their button press. When a user types “A1”, we will display a message on the interface notifying the user which buttons they have pressed and asking for confirmation. Once confirmation is received, getHumanInterfaceInput will return a ChessMove structure to the main module for further processing. If the move is not confirmed, we will loop back and wait for the user to re-enter the move before proceeding.

3.5 Hall Effect Sensors

The final design chosen for the Hall Effect sensor grid, as mentioned in Section 2.5.2, is a 4:16 demultiplexer and an 8 to 1 multiplexer. The demultiplexer output pins, a latching version, would act like a current source for the Hall Effect sensor grid; it is controlled by the microcontroller via four input pins, an enable pin, and an enable latch pin. If the power was cut from the enable pin the last output pin that was on would stay on until the current is cut from the enable latch pin. The multiplexer input pins act like a current sink for the Hall Effect sensor grid, feeding back to the microcontroller through its output pins.

The Hall Effect sensor grid itself should be a twelve by eight grid. The grid is designed as to not only accommodate the chessboard but also the “graveyard” for the pieces that have been taken by the opposing player. The graveyard’s sensors consist of the first two and last two columns of the grid, allowing the center of the grid to be for the chessboard itself. Each sensor in the grid should be mounted in a twelve by eight box grid and the boxes under the graveyard will only house the Hall Effect sensors while the boxes under the chessboard should house both the Hall Effect sensors and the LED lights.

For the microcontroller to have feedback, which allows it to read which of the sensors is active and keep track of which piece is where, the demultiplexer should be controlled to activate, column by column, while the multiplexer would be, at the same time, reading in which of the sensors are on and off; this information should be sent in a bit pattern format to the microcontroller. The designed Hall Effect sensor grid, showcasing the 4:16 demultiplexer and the 8:1 multiplexer, can be seen below in Figures 3.5-1, Figure 3.5-2, and Figure 3.5-3 below; Figure 3.5-1 and Figure 3.5-2 are zoomed in views of the essential parts of Figure 3.5-3.

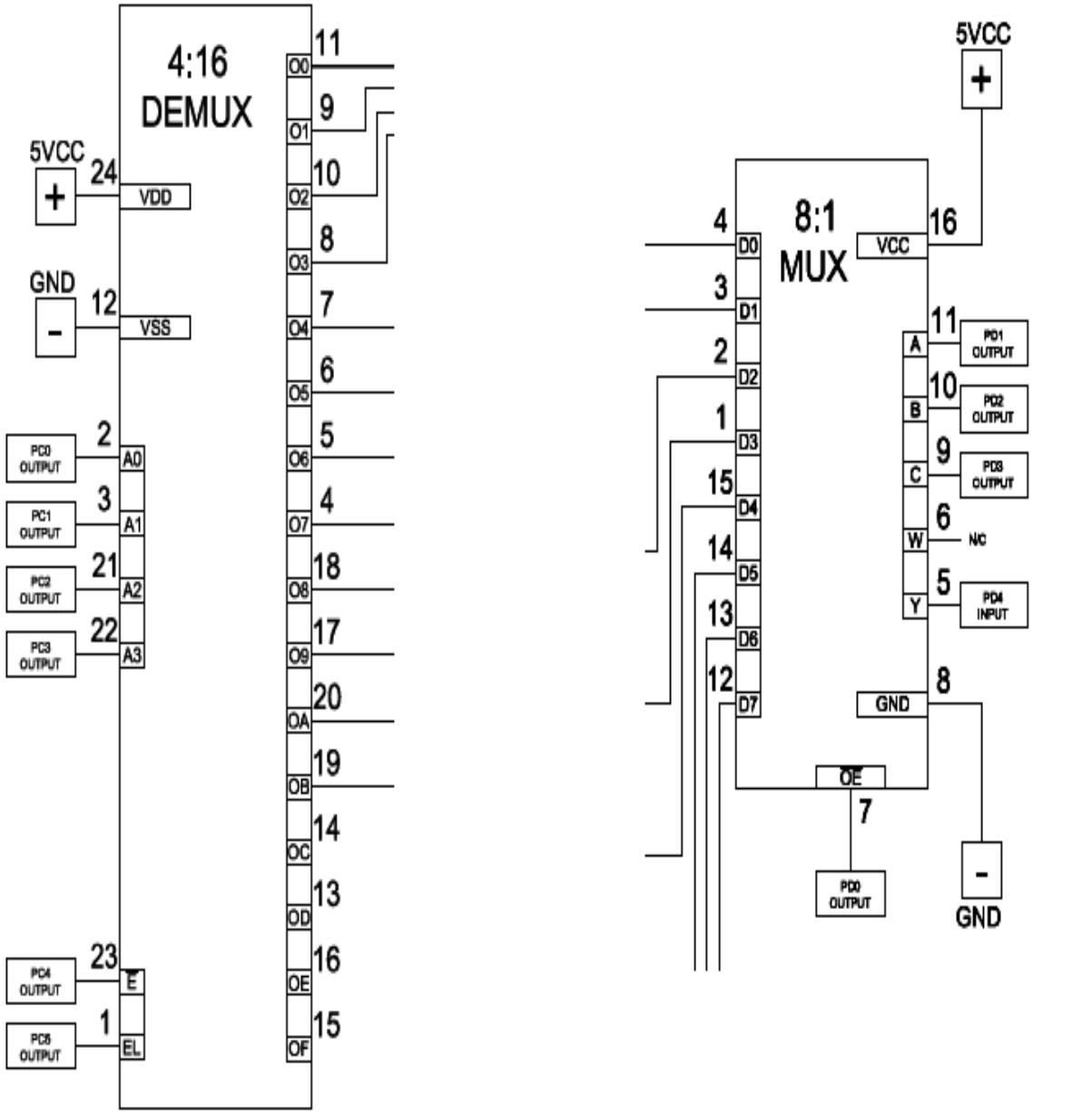

Figure 3.5-1: A close up view of the 4:16 demultiplexer and the 8:1 multiplexer utilized in the schematic for the Hall Effect sensor grid. This figure was created using AutoCAD educational software.

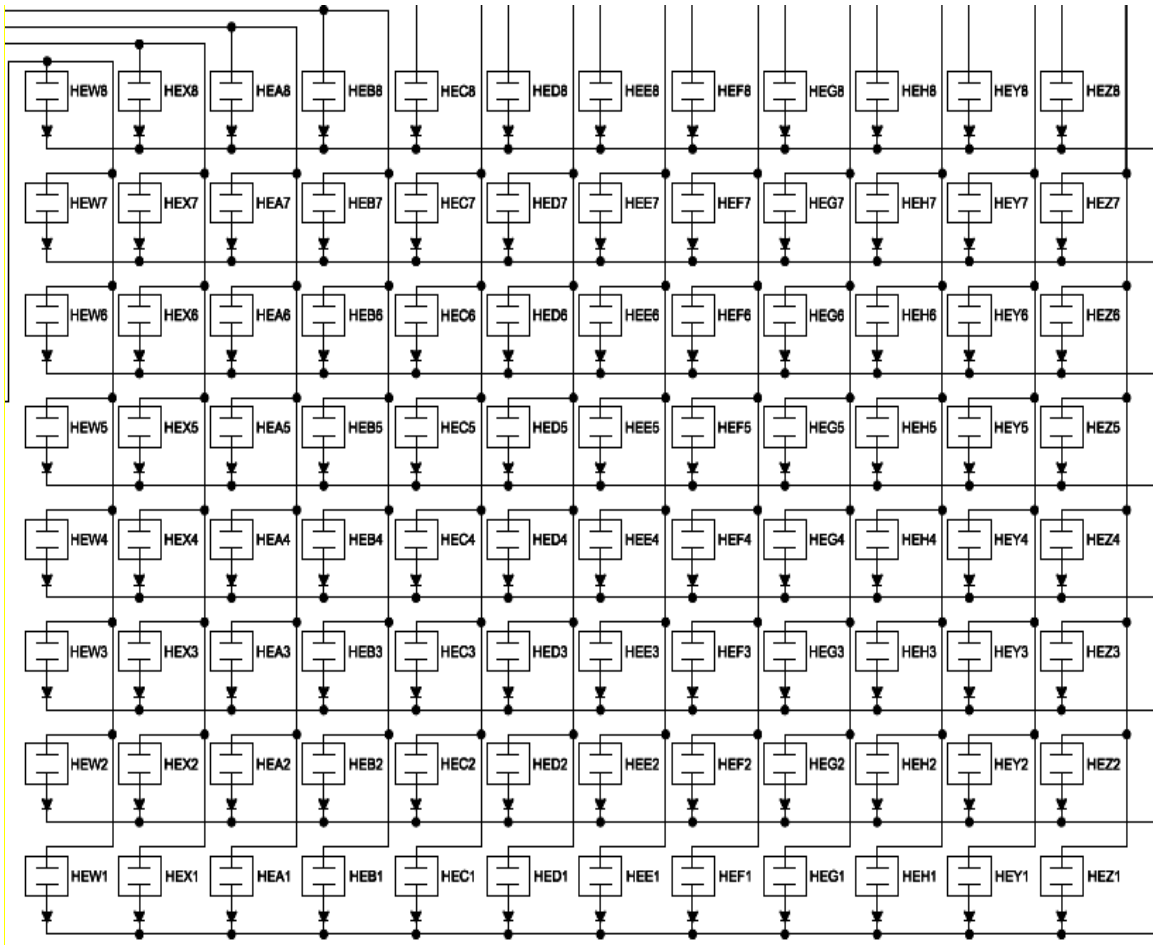

Figure 3.5-2: A close up view Hall Effect sensor grid. This figure was created using AutoCAD educational software.

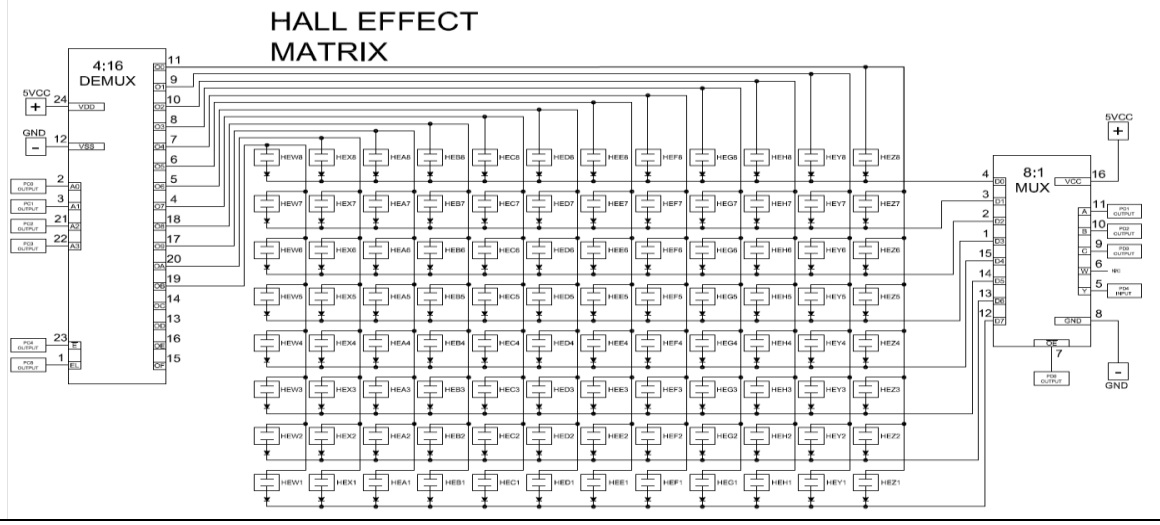

Figure 3.5-2: The full view of the final Hall Effect sensor grid. This figure was created using AutoCAD educational software.

3.5.1 Hall Effect Sensor Interfacing with the AI

The Hall Effect sensors are arranged in a 12x8 grid. Therefore, we would need to interface with a 4:16 multiplexer to activate the column that we are reading. Once we activate a certain column, we would need read the output from an 8:1 multiplexer. We can read the value of a single square on the matrix by setting the 3 control lines on the multiplexer. In software, we would create a function readHallEffectGrid(null) that counts up from 0x0 to 0xC for the 16:4 demux and 0x0 to 0x7 for the 8:1 mux, reading the output of the 8:1 mux to an array in the 0x88 format described in our chess engine section. Another function, readHallEffectGrid(location) should take in a number in the “0x88”” format described in our chess engine data structures section and poll that location on the hall effect grid using the method described above.

Please note that due to time, money, and space constraints we were unable to actually build and implement the Hall Effect sensor grid as designed and specified above.

3.6 LED Setup and Controllers

To reiterate from the research portion of this section (see Section 2.6), the microcontroller used for this project communicates, through a serial communication setup, with a LED driver, MAX7219 24 pin DIP, that was designed to operate up to an eight by eight LED grid. This driver has two more drivers daisy chained to it serially, utilizing pins one, twelve, and thirteen on each chip, to control all three colors as shown in the schematic diagrams found in Figure 3.6-2, Figure 3.6-3, Figure 3.6- 4, Figure 3.6- 5, Figure 3.6- 6, and Figure 3.6- 7 with the final wired product shown in Figure 3.6-8 below.

The serial daisy chain was configured to the microcontroller by linking with the Data Input (Din) pin on the driver that handles the red LEDs; then the Data Output (Dout) pin from the red LED driver linked to the Data Input of the driver that handles the blue LEDs, then the Data Output pin of the blue LED driver linked to the Data Input pin of the driver that handled the green LEDs, completing the chain and causing all three LED colors to be linked. The clocks (CLK) were also linked together for the serial connection; the microcontroller linked its clock to the red LED driver’s clock, and that clock was then linked to the clock of the blue LED driver, and then finally the blue LED driver’s clock was linked to the clock of the green LED driver and was causing all three LED colors’ clocks to become linked as well. This LED driver chip, shown below in Figure 3.6-1, acted as both a source and a sink, turning one LED on at a time rapidly which created the illusion that there were multiple LEDs on at the same time. The three drivers were used not only to produce the individual colors but was also utilized in mixing colors, such as the black squares being the LEDs that are simply shut off and the white squares using all three color LEDs to create the color white. There were other color combinations that would be mixed to represent other things such as

warnings that a player is in check, a pawn is changing into other pieces when it reaches its opponents side of the board, and invalid moves. This was changed to just a single color used for the squares due to the red LED's creating a slight voltage drop when their on making the rest of the LEDs much dimmer.

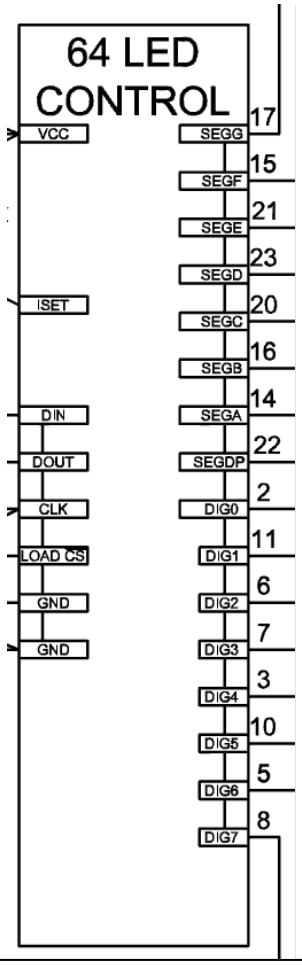

Figure 3.6-1: The LED driver chip utilized in Figure 3.6-2, Figure 3.6-3, Figure 3.6-4, Figure 3.6-5, Figure 3.6-6, and Figure 3.6-7 for the control of the red, blue, and green LED lights. This figure was created using AutoCAD educational software.

The LEDs were mounted in a box grid that is located under the board; each box mounted under the chessboard not only served as a place to mount the LEDs (and Hall Effect sensors had they been included) but also acted as a light shield in order to keep the lights segregated and prevented the light from one square leaking into another. The grid consisted of flat rectangular plastic boards that have slits cut into them so they piece together into a box grid. The box grid for the LEDs contained twelve by eight boxes and only the boxes under the playing chessboard have the LEDs mounted in them.

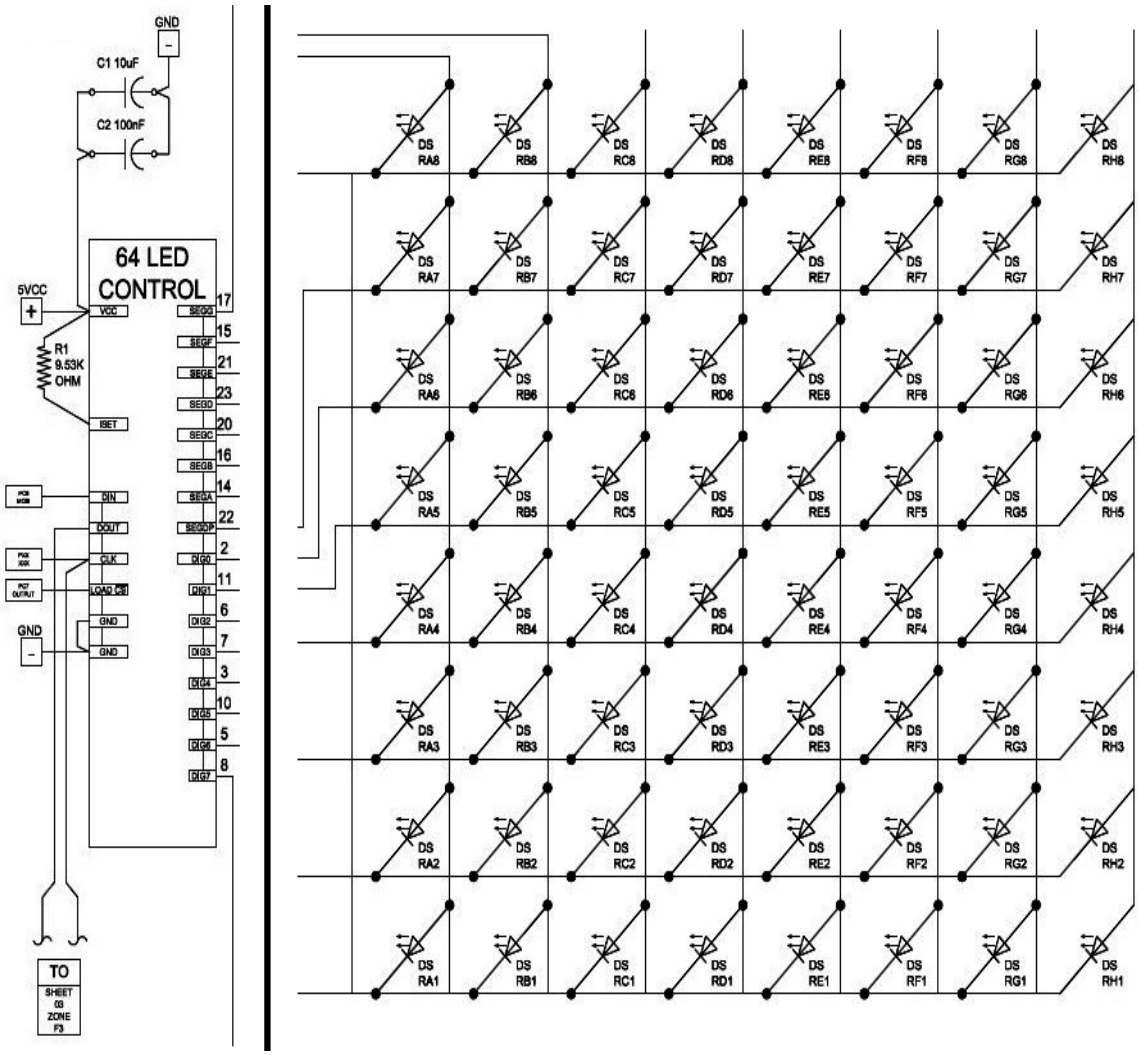

Figure 3.6-2: The essential components for the red LED schematic zoomed in for easy reading. This figure was created using AutoCAD educational software.

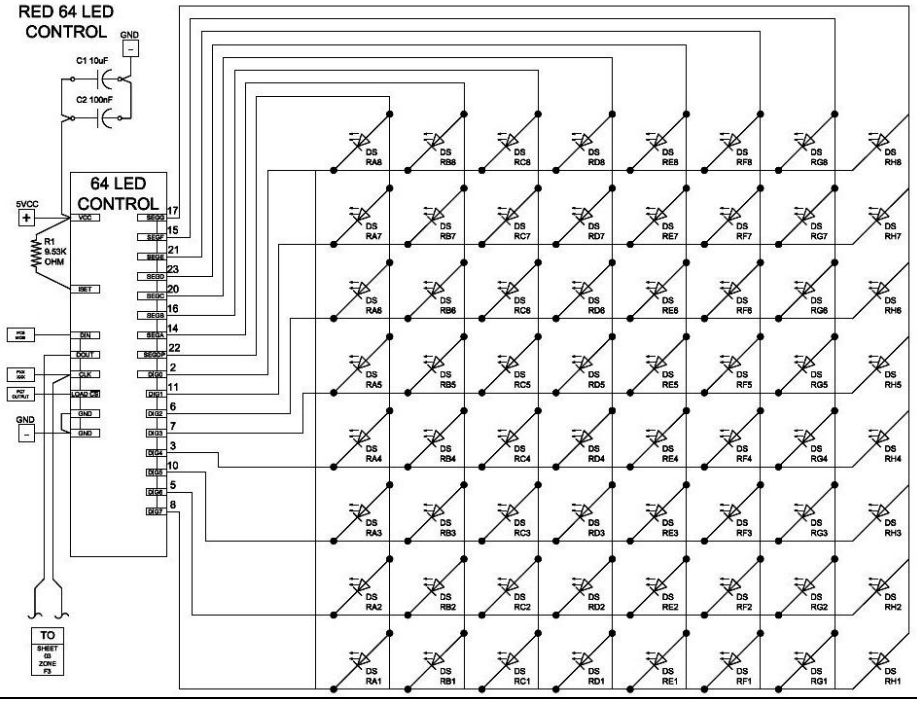

Figure 3.6-3: The overall red LED schematic. This figure was created using AutoCAD educational software.

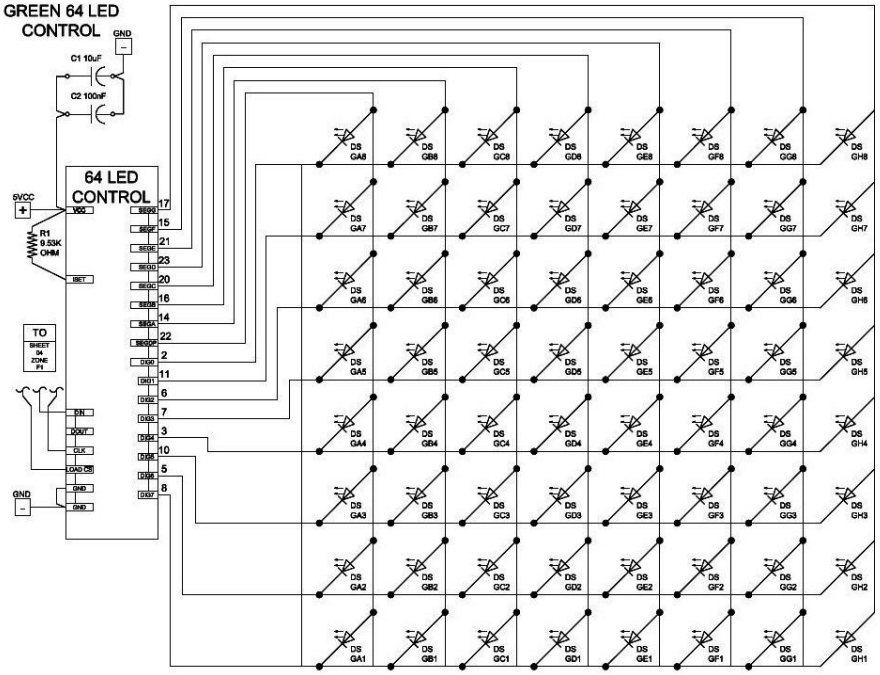

Figure 3.6-4: The overall green LED schematic. This figure was created using AutoCAD educational software.

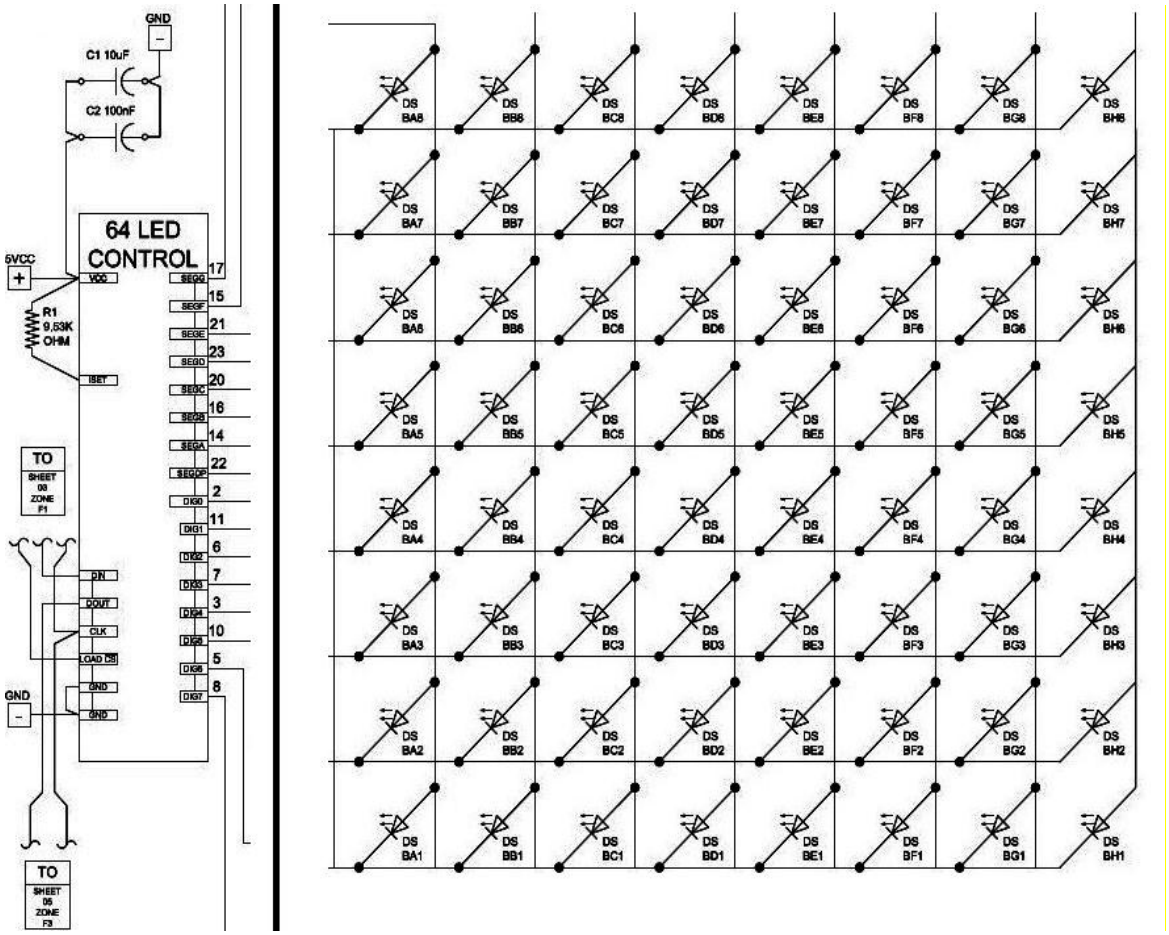

Figure 3.6-5: The essential components for the blue LED schematic zoomed in for easy reading. This figure was created using AutoCAD educational software.

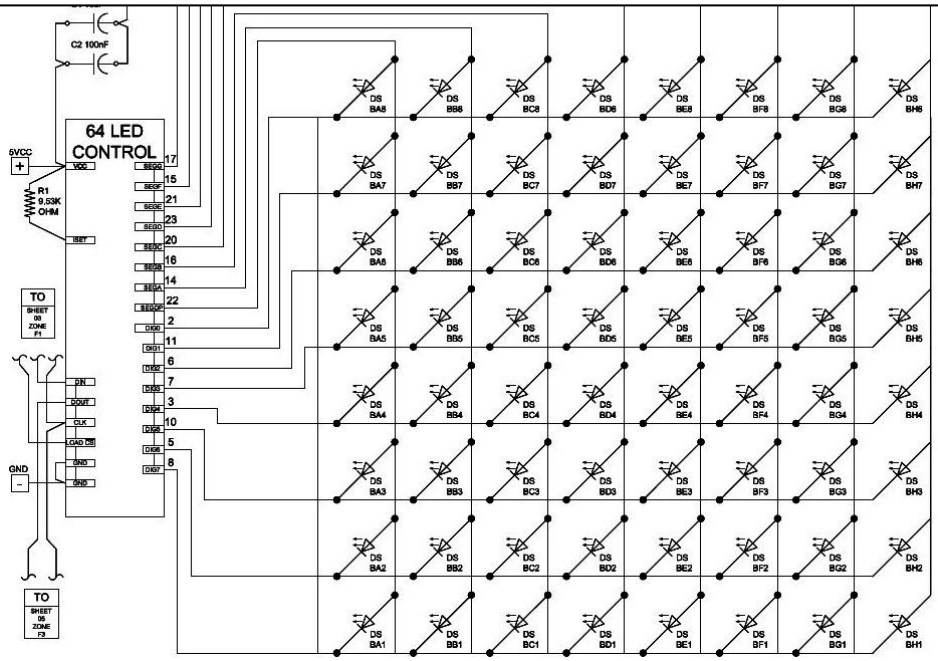

Figure 3.6-6: The overall blue LED schematic. This figure was created using AutoCAD educational software.

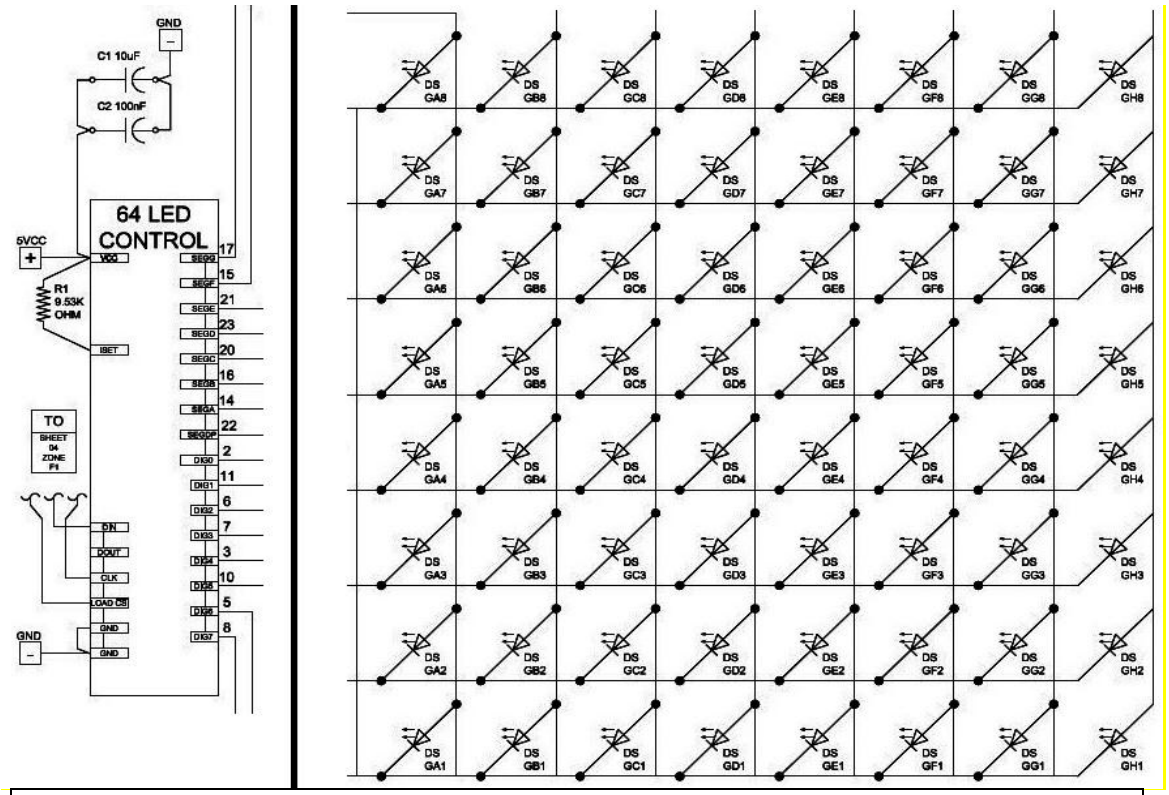

Figure 3.6-7: The essential components for the green LED schematic zoomed in for easy reading. This figure was created using AutoCAD educational software.

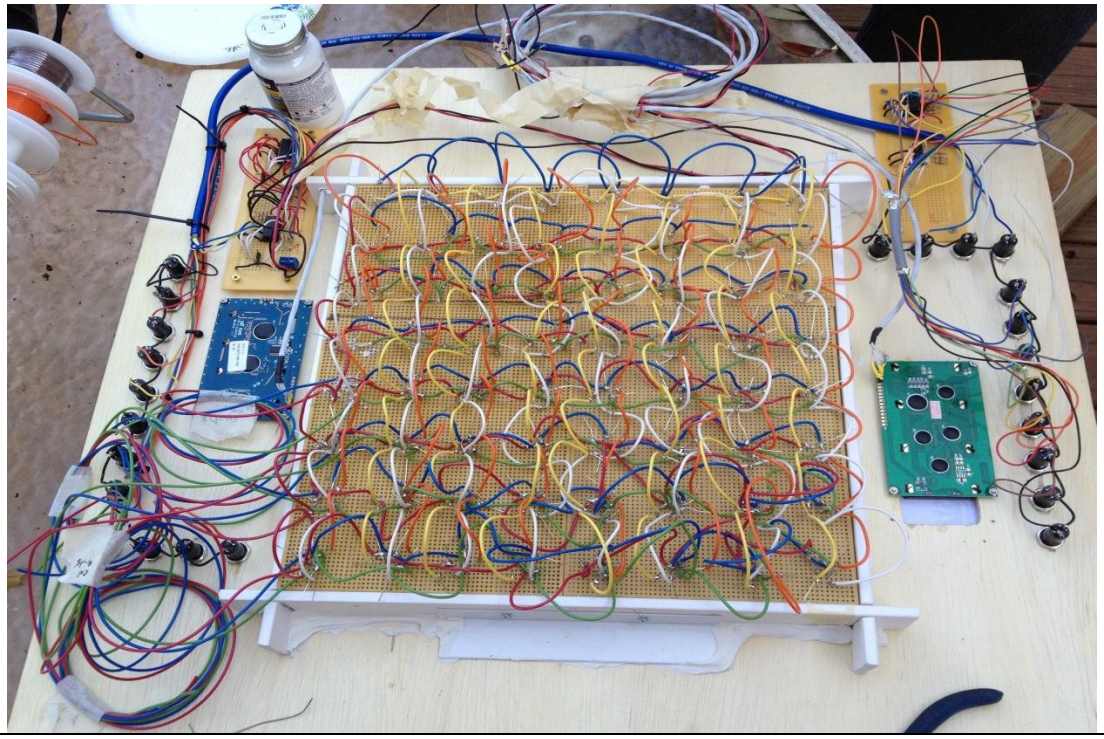

Figure 3.6-8: The overall wiring of the 192 LEDs utilized for our project without the insulation.

3.6.1 LED Control

We implemented the LED controller by modifying a library we found in our research section for the Max7219 LED driver. It contains functions such as MAX7219_Write, MAX7219_SendByte, MAX7219_LookupCode (which we won't need since we're not displaying actual digits), MAX7219_Init which must be called by the I/O module before using the controller, MAX7219_Clear, MAX7219_SetBrightness, MAX7219_DisplayTestStart, and MAX7219_DisplayTestStop. These functions are a good starting point. It allows us to initialize, perform a test, set the brightness of our driver, clear it and send bytes to it. We will be sending bytes in a format that maps each byte to a rank of the chessboard. There are three LED matrices to control separately.

There are many configurations that the LEDs will be used for. When a player selects their piece, a message will be passed to the LED controller to light up their possible move paths with the color blue. Once a player selects a move, the final location will be highlighted in green.

3.7 Magnets (Pieces)

As mentioned in Section 2.7, the claw has memory foam attached to the inside of the prongs, so the chess pieces were able to be store bought pieces. The site wholesalechess.com had the ideal chess pieces we needed for our project at a reasonable price. The chess piece set that our group decided to go with is a full wooded dark wood and light wood (to signify the black and white) set pieces, the sizes ranging from the tallest piece (the king pieces) at 2½ inches to the smallest piece (the pawn pieces), at 1½ inches. To prevent scratching of the chessboard we chose pieces that came with a felt pad on the base of each piece. Originally we intended to remove the felt and drill a hole in the bottom of each chess piece and insert a magnet into the hole we created but since we were able to find the website above with pieces already made to those specifications so no modification of the pieces we ordered was necessary as shown below in Figure 3.7-1 and 3.7-2.

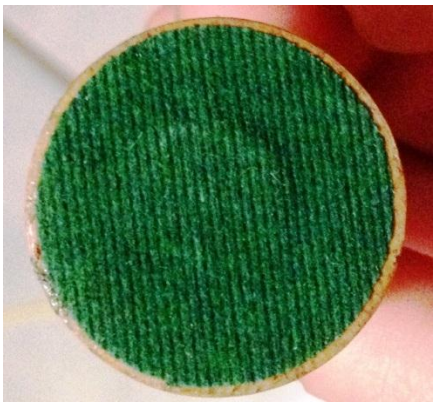

Figure 3.7-1: The felt bottom of a piece showing the magnet embedded within the piece

Figure 3.7-1: The chess set we chose for our project from http://www.wholesalechess.com/chess/sd_magnetic_wood_pieces

The magnet in the bottom of each chess piece was originally chosen for two very specific purposes: to activate the Hall Effect sensor under each square (not in our final product) in order to keep track of the individual pieces on the chessboard and to magnetize to a piece of Farris metal under each square (also not in our final product). The magnets are flat, typical refrigerator type magnets, around fifty Gauss in strength. The purpose of the piece magnetizing to the Farris metal was to keep the chess piece relatively centered on the square. Each time the claw picks up a piece and places it down it could, potentially, set the piece down at least slightly off from the center of the square which would increase the possibility that the next time the claw picks up the piece it would not securely latch on to said piece which could lead to another offset landing or dropping of the piece all together but with the accuracy our programming and claw has demonstrated after repeated testing, combining with time, space, and monetary restrains, rendered the pieces of Ferris metals unnecessary.

3.8 Mechanical Assembly

As stated above in Section 2.8 the X axis will be defined as movement either to the left or right of the user and the Z axis will be considered as up or down movement of the grabber. The playing surface is made up of 1.5 by 1.5 inch squares arranged in an eight by eight matrix. Along either side of the playing grid a two by eight matrix will be located to place discarded pieces. The two discard

matrixes will be located 1.5 inches offset from the sides of the board, approximately one square's width. The Y axis must be able to traverse across rows 1-8 with a total displacement of 10.5 inches. The X axis must be able to traverse across columns A-H plus five of the discard columns, for a total displacement of 19.5 inches. The Z axis must be able to lift any piece high enough to clear the highest piece on the board but there is not any need to lift the piece much higher than that to provide a safe clearance margin. The motion track system must allow for full travel of the carriage plus some amount of over travel margin. Over travel detection will be needed to kill the motor drive in the event that an over travel condition occurs and before encountering the mechanical hard stops. The Z axis will be driven by a spool attached to a larger spur gear driven by a smaller spur gear on the motor shaft. All three axes of motion will require home limit switches to calibrate the mechanical system with the control system. When the control system boots up after power has been restored all three motion axes must be driven to the home position. Figure 3.8-1 below shows the lower front view with the Y axis stepper motor mounted to the moving table that is the base for the A frame structure. The Y axis rack gear can be seen attached to the bottom side of the stationary chessboard frame structure. The stepper motor bracket will be properly shimmed to apply sufficient upward force from the spur gear to the rack gear. Figure 3.8-2 shows the concept drawing of the slides under the board that allows for the A frame to move.

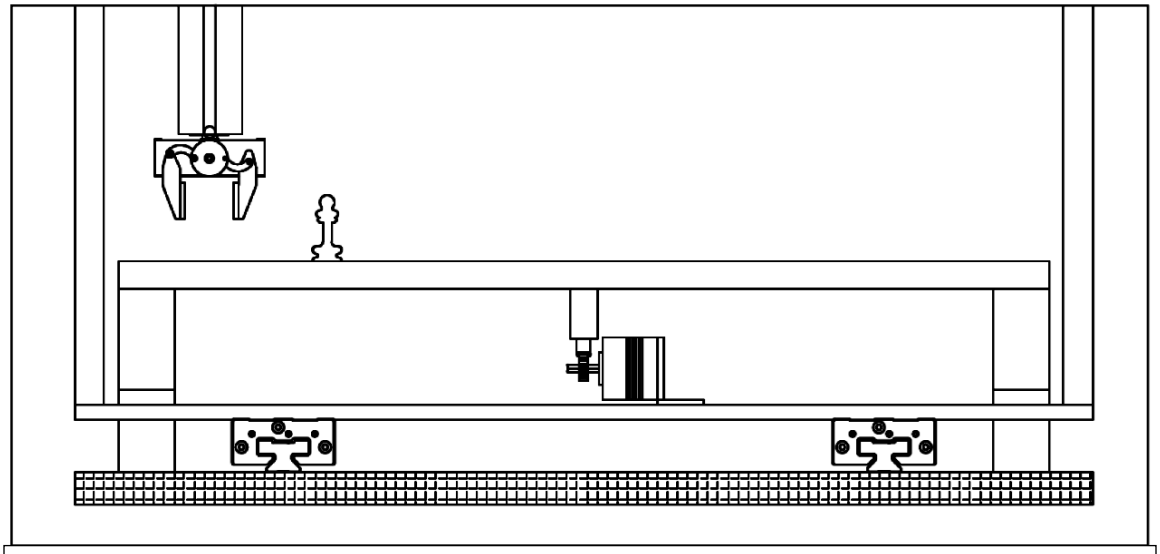

Figure 3.8-1: Y Axis linear slides and stepper motor. This picture was drawn using AutoCAD educational software.

In the research phase, design problems were clearly identified with the study of a similar chessboard; the Y axis in that example had possible binging issues and failed the 2:1 ratio test. With just one linear bearing for each side the vertical members acted as a leaver arm on the bearings. Any slop in the bearings would be multiplied by the height of the vertical members. Our design will used an A frame structure that is moved as a structure along the Y axes. The A frame

structure is driven from under the chessboard with just one motor and our chessboard will bridge or straddle the base of the A frame while the legs or feet are positioned closest to the players. This design approach proved to be more ridged with respect to tolerances and result in better positioning accuracy. We have also reduced the number of motors down to one for the Y axis of motion. The cut away side view of our design concept can be seen in figure 3.8-3. The diagram clearly shows that we have a ridged structure, and have solved the problem with the Y axis tolerance. Along the upper section of the A frame structure is the rack gear that is fixed on the structure. While the stepper motor with the spur gear attached to its shaft moves along the X axis it is also a part of the hanging assembly. In the cut away found in the Figure below two pinch wheels can be seen providing good lateral alignment along the section of tee rail below the sliding carriage. The two pinch wheels were not needed for lateral stability as was anticipated in the design phase. We were able to offset the center of mass for the sliding X axis assembly this approach applied a force to the spur gear toward the rack gear. The force assures proper mating of the gears with no observed slippage. Vertical motion will occur as the result of the stepper motor spur gear rotating the larger spur gear above; causing the spool to rotate winding, or unwinding the two lift cords which will lift or drop the grabber assembly. The front view of figure 3.8-2 shows the Z axis grabber in the lower position. The height adjustment of the A frame structure was made on each of the four corners by adjusting the ¼ 20 mounting screws (shown in figure 3.8-2). This adjustment allowed us to apply uniform force to the Y axis spur gear, as it traverses across the Y rack gear.

Figure 3.8-2: The gears and sliders located under the board that allow the A frame to move unimpeded.

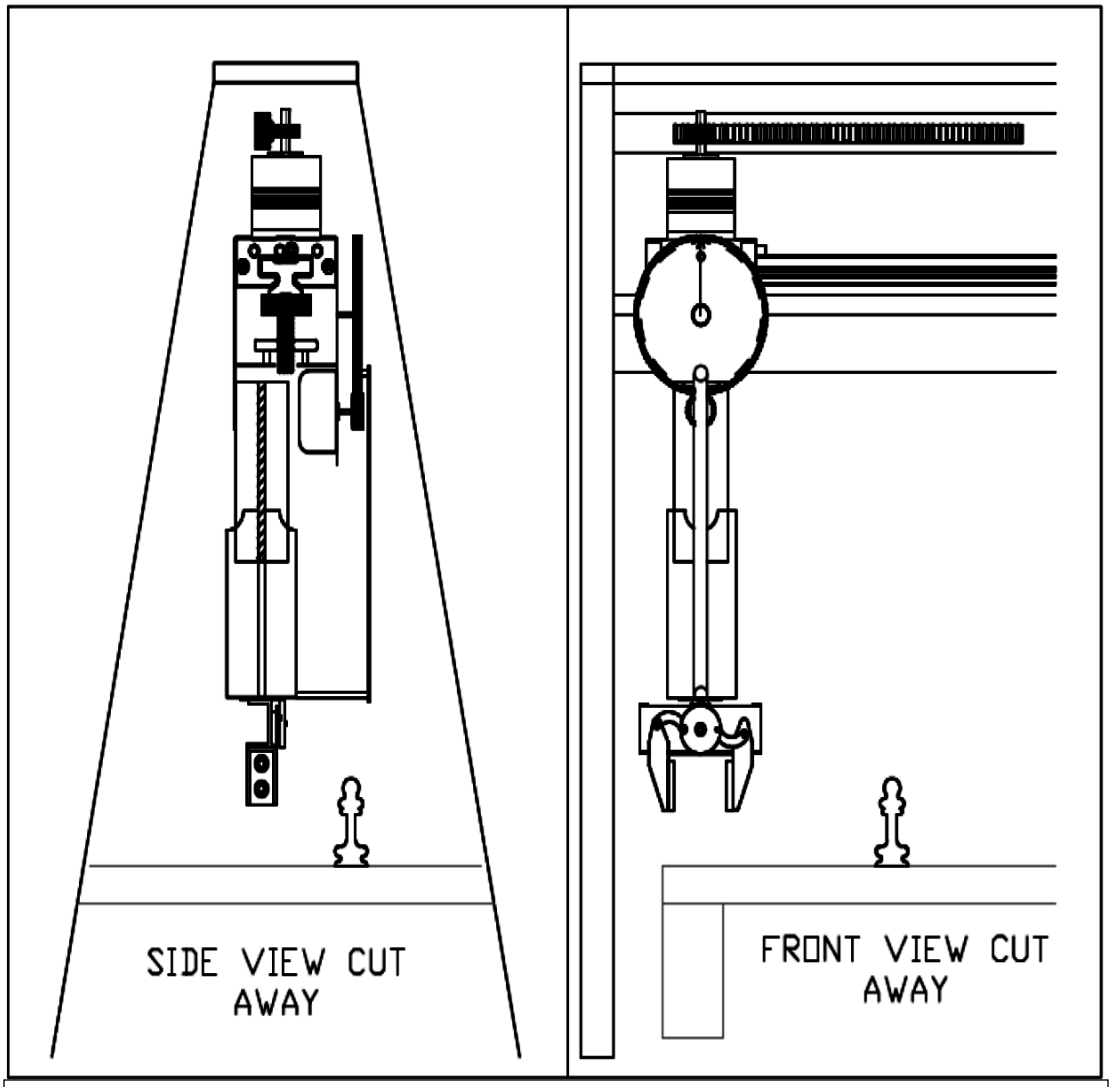

Figure 3.8-3: Grabber and Z axis cut away views. This picture was drawn using AutoCAD educational software.

3.9 Motor Control

We selected the bipolar stepper motor for all three motion axes. The bipolar stepper motor proved to be a highly cost effective solution at just $14.00 each for all three axes of motion. The stepper motor eliminates the need for feedback allowing us to operate our control system in open loop mode. This resulted in significant cost savings, and lead to a greatly simplified design complexity for the control system. The bipolar motor provides more torque to overcome the coefficient of static friction when starting motion compared to the unipolar motor. The motor controllers can be seen below in Figure 3.9-1 and Figure 3.9-2.

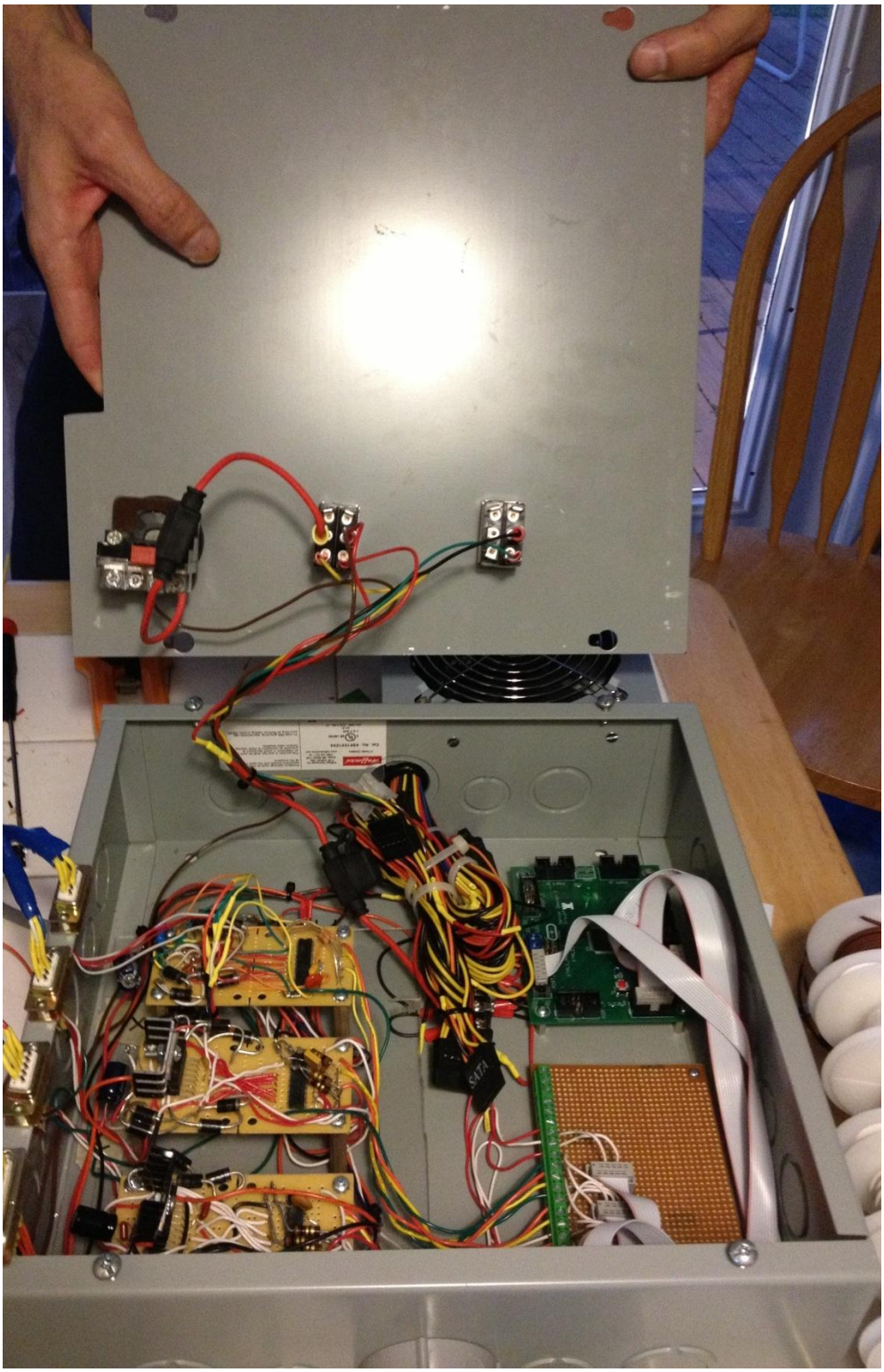

Figure 3.9-1: The housing with the Interconnect, three Motor Controllers, and the Microcontroller mounted inside

Figure 3.9-2: A close up of the housing with the Interconnect, three Motor Controllers, and the Microcontroller mounted inside

The motors were selected with the consideration of what spur gears can be mounted to them. All of our stepper motors are 12 Volt shave a step angle of 1.8 degrees, or 200 steps per revolution; this ultimately leads to no gear boxes being required in our design. The motors have a 5 mm shaft bore size allowing for good spur gear selection choices. The diameter of the spur gear will affect the total displacement traveled in one revolution thus the diameter of the spur gear is a dependent factor on how much resolution of control we have. The size of the square is 1.5 inches by 1.5 inches, and we desire to obtain 95% accuracy in placement of the chess pieces. An Excel spread sheet was created with various spur gear diameters. From this tool we generated a table comparing resolution, gear diameter, displacement per motor step, and step tolerance.

3.9.1 Gear Selection

We selected the 1 Mod Flexirack with the Aluminum extruded track for our rack gear for both the X and Y axes of motion. The benefits of this choice are that we could easily cut the gear material and the track to the length of our design. Mounting holes were drilled into the Aluminum track, although we had to countersink the holes, and use flat head screws to allow for a flush surface for the Felxirack product. The Flexirack product line for spur gears is very limited, however, and only comes in 6 mm bore sizes; we have a 5 mm bore size for our motor shaft. Fortunately gears come in standard sizes and therefore a number of alternative plastic spur gears were available that met our requirements. Our preference was to have a set screw type spur gear, so that the gear shaft slippage can be eliminated. For the Z axis we used two 0.8 Mod spur gears with the gear driving the spool of significantly larger diameter to allow for mechanical advantage in motion to lift and lower the grabber. Due to the difficulty of drilling stainless steel gears were not desired.

3.9.2 Motor Control

Our design intent was to modularize construction, and standardize parts as best we can. Our motor control design highly leverages that philosophy. All three stepper motors are 12 Volts, bipolar, with a 1.8 degree step angle therefore one motor control design can simply be used three times. We used subminiature D connectors on the motor control output terminations. When an issue arises with a motor we would plug the motor into another motor controller. The microcontroller input signal leads could also be swapped.

We considered the motor control circuitry a critical path to successful operation, and we identified the need to bread board the circuit in the research phase as stated in Section 3 above. If last minute problem with infant mortality of a component had occurred we would have the need to make a quick repair. This is the reason we chose to avoid surface mount integrated circuits and, by extension, avoid the PowerSO-20™ package. Instead we selected the Multiwatt 15 package so that we can easily solder a new one in place if the need arose for our motor Duel Full Bridge Driver. This driver, the L298N from ST Microelectronics, is designed to switch the stepper motor field on and off as well as reversing the current flow direction as needed. The driver is basically a fully integrated H bridge; paired with the ST Microelectronics L297 Stepper Motor Controller IC the I/O and code complexity of the microcontroller is greatly reduced as our design as a whole is complex enough and has consumed substantial microcontroller I/O.

With the selection of the Multiwatt 15 driver package it was also determined that we should mount it to a significant size heat sink and apply thermal compound between the component and the heat sink to assure that no thermal over heating issues arise. Heat sink temperature was checked in the testing phase. The

results lead us to install a larger heat sink on the Y axis driver. By the design of our machine in general the duty cycle of our motor controllers are greatly reduced. The operation of simultaneous motion of the X and Y axes is physically possible, though it was not a design requirement. We chose not to explore simultaneous motion after we built the machine, and were in the testing phases. If only one axis moves at any one given time the individual motor controller duty cycle is reduced to well below 50% just by design. If something unforeseen happened and we had to make a new motor selection, our motor control selection has the ability to vary the motor voltage from 7.5 VDC up to 46VDC. The X axis motor control is shown in Figure 3.9.2-1 below. This design was derived from the St Microsystems L297 and L298 spec sheets. This same design is repeated three times, once for each axis. Reference the full wiring diagram in the Appendix to view connections for the other motion axes.

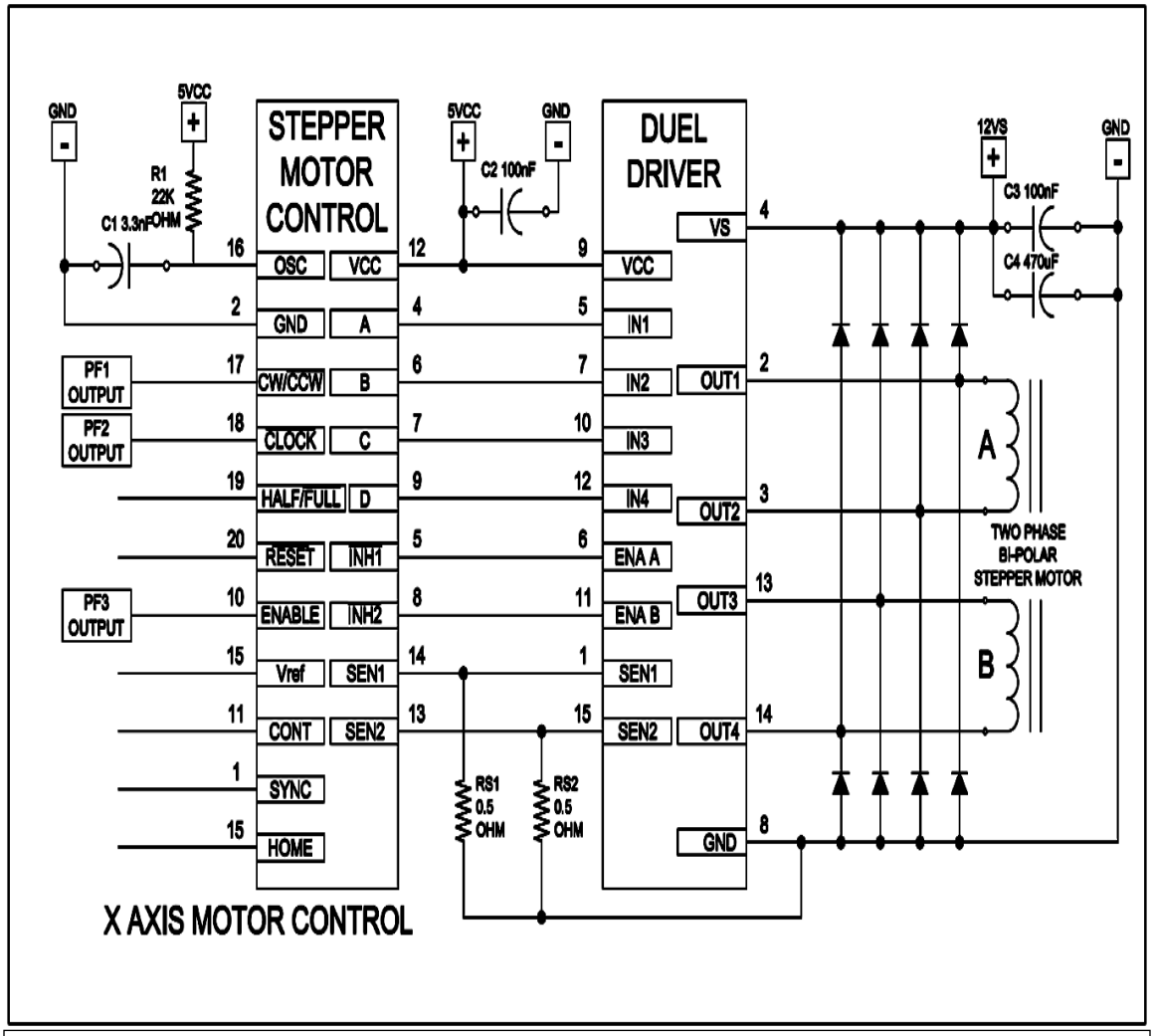

Figure 3.9.2-1: X-axis motor circuit wiring diagram. This picture was drawn using AutoCAD educational software.

3.9.3 Control Circuits for Motion Control

With any motion control system the need for an emergency stop must be considered; what if there is a problem and the user needs to quickly kill power to the machine? We have decided that we at least need to include a switch to disconnect the 12 volts DC from the three motor drives. For testing and troubleshooting activities the need arises to have the microcontroller powered up and running while knowing that no motion can occur in order to prevent any damage to the project as it was being tested. Our design required that we mitigate the over travel condition for both the X and the Y axes, and as such, our solution for this condition is to install four limit switches, two for each end of both the X and Y axes..

Under typical operation the normally closed limit switches should never be activated. We chose to connect all four over travel limit switches in series in the 12 Volt DC motor bus circuit. Ideally it would be nice to have one input for each over travel limit switch but the need to conserve I/O comes into play and outweighs this desire. The advantage to this strategy is that our over travel monitoring circuit is completely independent of the microcontroller. Furthermore single point failure modes of the microcontroller do not need to be considered in the control system design. If any one of the over travel limit switches is activated the 12 volts is removed from all three motor drives.

The need also arises for the machine to have the ability to calibrate the motion axes with respect to some fixed point on the machine. This is needed because if the microcontroller loses power, how will it know where the axes are located without absolute feedback? This is also the mitigation for error accumulation. After every game the machine returns to the home position; our solution for this problem was to install a home sensor on the X,Y, and Z axes. The home position is centered about three inches outward from the center of cell A1. In the test and adjust phase we determined that we would not need to add a down position sensor. When the spool unwinds the sliding gripper unit rests on a hard stop, and just has the force of gravity acting on it.

Grabber open and close monitor points have also were eliminated in our final design. The reason was that we were able to make the needed adjustments on the mechanical side. Figure 3.9.3-1below shows our control circuit for this motor control.

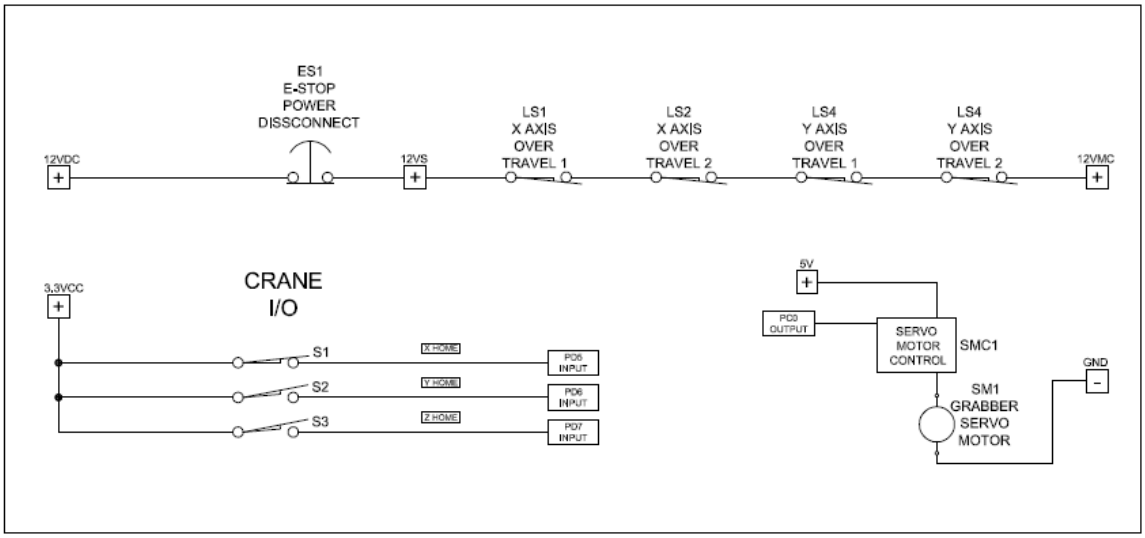

Figure 3.9.3-1: Control circuits for motion control. This picture was drawn using AutoCAD educational software.

3.10 Picker/Claw

Having made the Grabber device selection in the research phase, we also chose to the servomotor to actuate it. The servomotor can be operated in an open loop mode similar to the way we operate our stepper motors. The difference between the two is that the servomotor is controlled by the duty cycle of the signal pulses sent from the microcontroller. The servomotor that we selected has an angular rotation range of about 100 degrees. We can change the opening by varying the duty cycle of the command pules from 1ms to 2ms, with 1.5ms being the center position. During testing we discovered that we could mechanically set the 100 degrees that the grabber would travel, so we set it to max open position with no signal applied. We varied the signal duty cycle to close the grabber. We planned to fit the grabber with open and close sensors, but it turned out there was no need for this.

We modified the gripper by adding fingers to extend the gripping range, to accommodate the diverse heights of all the chess pieces. We also added memory foam pads to the fingers to compensate for the diverse shapes of the different chess piece figures. Over clamping of even a single chess piece could result in damage to our servomotor. We mitigated this condition with the mechanical adjustments. In the test and adjust phase we evaluated the clamping pressure required to consistently pick and release all pieces. If the grabber enters into an over clamping condition the motor current should increase approaching the stall current values listed in the manufacturer's specifications sheet. The final design for the grabber can be seen in Figure 3.10-1 and Figure 3.10-2 below.

Figure 3.10-1: The total grabbing mechanism for the Interactive Automated Chess Set

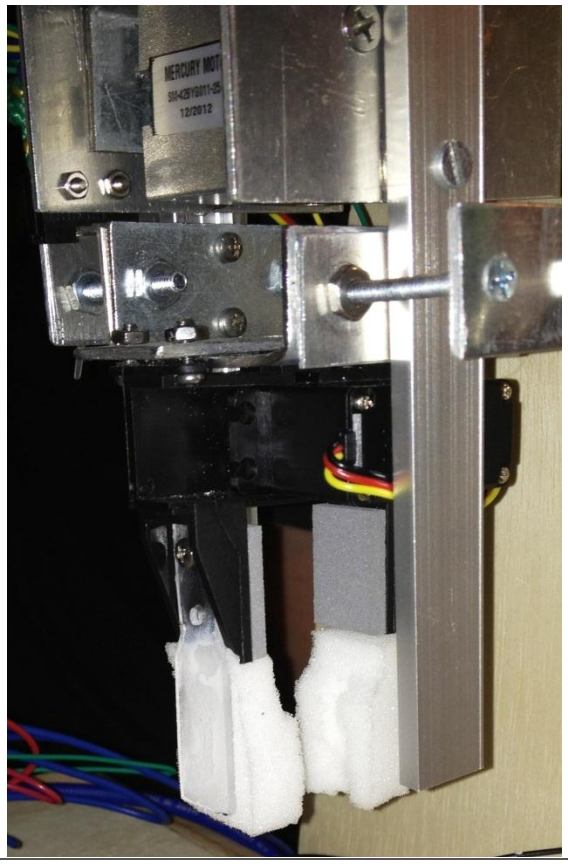

Figure 3.10-2: A close up of the grabber with extenders and memory foam added for extra grip on the smaller and oddly shaped pieces for the Interactive Automated Chess Set

3.11 Audio Design

3.11.1 Audio Interfacing

We originally planned to create music with our buzzer, so we researched a table of frequencies for different sounds. We decided to map out at least two octaves in our memory as shown below in Table 3.11.1-2. Given a mapping of notes to frequencies, we could store songs in our memory as arrays of notes and rests as shown in Table 3.11.1-1 below. For simplicities sake, when the notes are being played there would be a standard interval of a sixteenth note at 120 BPM for each note in the sequence. This would have allowed us to make our songs quick and up-beat. We planned on having a separate sound sequence for taking a pawn, taking a higher ranked piece, check, checkmate, game start, user input, and error. We planned to store these note sequences as arrays of frequencies in memory. The frequencies below are defined as enumerators. Our I/O interface has a function, playSound() that we planned to be able to take in the enumeration of an event and would play the sound accordingly. After much

deliberation, consideration, and deliberation we decided against including an audio feature due to time, available space, and money.

Event	Note Sequence
Take a pawn	G4 - B4 - C#4 - C#4 - G4 - B4 - C#4 - C#4
Take a highly ranked piece	C#4 - E4 - C#5 - C#5 - C#4 - E4 - C#5 - C#5
King in check	G4 -E4 - D4 - D4 - G4 - E4 - D4 - D4 - G4 - E4 - D4 – E4 - D4
Checkmate	C5 – C5 – C5 – C5 – G#4 - A#4 - C5 – A#4 - C5
Game start	C4 – C4 – A4 - A4– G4
Error	B4 – B5 – G#4 – G#5 – A#4 – A#5
User input	C#4 - F#4 - F#4 - A4 - C#4 - F#4 - F#4 - A4

Table 3.11.1-1: Note sequences for events in the map

Note	Frequency
C4	261.63
C#4	277.18
D4	293.66
D#4	311.13
E4	329.63
F4	349.23
F#4	369.99
G4	392
G#4	415.3
A4	440
A#4	466.16
B4	493.88
C5	523.25
C#5	554.37
D5	587.33
D#5	622.25
E5	659.26
F5	698.46
F#5	739.99
G5	783.99
G#5	830.61
A5	880
A#5	932.33
B5	987.77
C6	1046.5

Table 3.11.1-1: Frequencies of commonly used musical notes

3.12 Power Supply

As mentioned in section 2.12, the power supply was store bought to reasonably ensure that the power for our project will remain stable and reliable as shown in Figure 3.12-1 below. The power source will still be wall power; the power supply was a Glite Computer power supply, 460W, with the outputs being +5V, +12V, and +3.3V. The 3.3V, 5V, and 12V outputs will be used, the rest of the connecters were left disconnected. The high amperage needed for the motors and the relatively high amperage needed for the LED grid was provided by the power supply, 20A max for the 5V line, 3.3V line, and 18A for the 12V line.

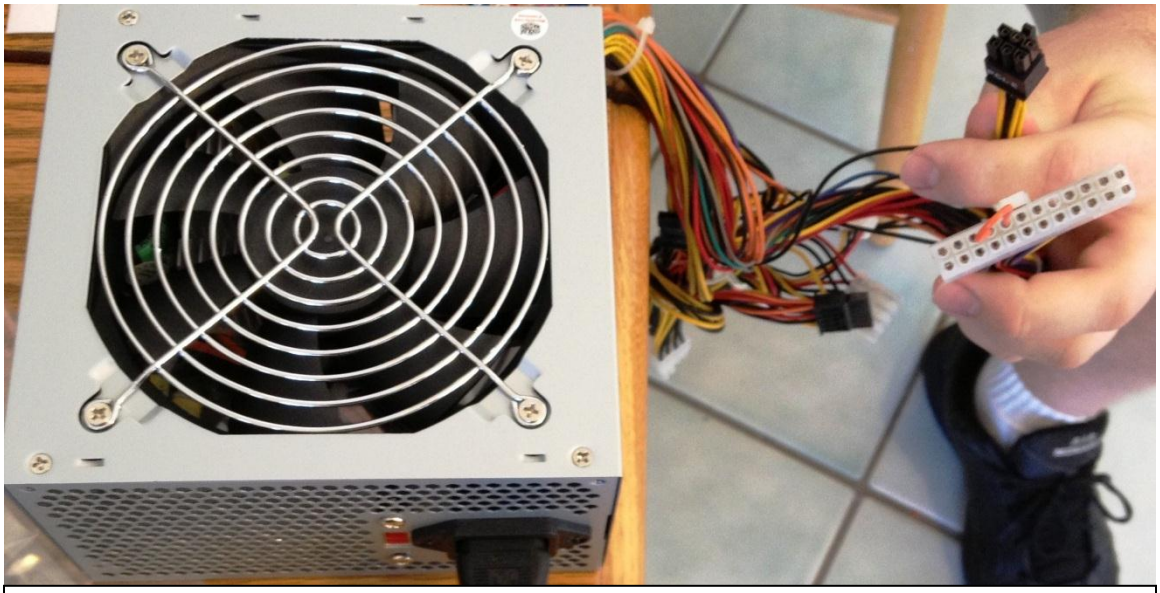

Table 3.12-1: The chosen power supply for our Interactive Automated Chess Set

4 Section 4 Prototype and Testing

4.1 Build and Implementation Strategy

We as a group identified the critical path tasks and prioritized them. The only way to succeed at this task is to maximize team work and delegate tasks while working in parallel. We want to build the mechanical assembly as well as design, build, and test our microcontroller PCB, and start bread boarding as much of our project as soon as possible. All three of these activities should start as soon as the design documentation phase is completed. Our group purchased an evaluation board with the same Atmel Xmega 128A1U Microcontroller IC, so that software development can take place. We also purchased the components and started to build one test motor control circuit early in Senior Design I. Once this effort was completed successfully software testing was be done on the motor control interfacing. Building of the actual mechanical assembly was started in late

July, and was worked on during the semester break to help utilize the maximum time available to us.

Our strategy was to construct the electronic assemblies in a modular format. The reason for implementing the modular strategy is twofold: the first is to simplify trouble shooting efforts and the second is to minimize the foot print of our PC boards. We used DB9 connector to connect the motor leads to the motor control boards to achieve the modularity we desired. When troubleshooting electronics or software code the divide and conquer method is a powerful tool. In the test and adjust phase we were able to start with the microcontroller and one HMI circuit. The HMI LCD display was our view into what is going on with the processer; we added messages for troubleshooting and HMI switch verification. Each sub system was brought on-line and tested one at a time; this plan helped us to know and keep track of when and where a problem were introduced. Our electronics will be divided into the power supply, the microcontroller, two HMIs, three motor control drives, and the LED lighting grid. The power supply and the microcontroller must always be functioning or none of the sub systems will function and as such will take priority; each of the other systems could be disconnected. Our power supply will supply three output voltage busses, 12VDC, 5VDC and 3.3VDC and if a short occurs on any of the power busses we will be able to isolate the area as needed to find the root cause. We tried to find components available in the DIP package when available to make bread boarding and repairs easier. Figure 4.1-1 below shows this modular layout philosophy in the system interconnect diagram.

We selected a microcontroller with a high I/O pin count because our application is highly dependent on discreet I/O. Unfortunately the Xmega 128A1U that we are working with is not available in the TQFP package. This is a 14mm x 14mm square 100 pin with surface mount pins. The IC also comes in the ball gate array (BGA) package; the ball gate array is an interesting option that we did not select. Finding the right BGA socket was challenge.

One of our team members designed a small simple PCB just for the microcontroller and this PCB had just basic functionality on it. Things like the reset circuit, power filter components, and remoting of the I/O pins to header connectors as required. Unlike the MSP 430 where the IC can be removed from an IC socket and programed in a development board we do not have that option; a programing port was be needed. We used the Olimex AVR-ISP-MK2 programing dongle to program our micro controller. The Olimex contained all of the programing support circuitry, so that we did not have to develop our own programing interface. We saw this as a critical path and prioritized this effort. Important lesson learned from the students in the previous semester’s Senior Design class was that too many groups waited too long to have their PCBs built, and programing interfaces were a challenge; this is something we clearly want to avoid.

Once we began the test and adjust phase of the project and started to move our motion based axes. Although we have calculated gear ratios and resolution of motion the experimental values obtained in testing were successfully cross-checked with the calculated values. The values were right on the money, and this was a great way to verify that we had no issues with torque, or skipping gears. We assumed if we experienced constant scaled motion, and just have a reasonable error tolerance between the two data sets, we can conclude the error is just component tolerance. If however we have inconstant motion, and the measured values vary with the calculated scale, then we would need to look into the possibility that our stepper motor system is operating a resonate frequency. If that was the case we can consider deploying a frequency step function on the software side. We were exercising the motion axes and taking carful measurements throughout the testing phase. We did have to play with the clock speed to obtain this reliable consistent motion. We discovered that as the gripper assembly moved out to the far ends of the X axis, that the center of gravity changed and binding could been seen while traversing on the Y axis. This problem was mitigated by setting the Y axis up for full steep mode. There was the possibility after making a move to an adjacent cell the results may be within the margin of error. If however we make a move spanning the length or width of the board, what was a small error will be multiplied by 6, 7, or 8 times; this is why consistent scaling is so important to our results. The set up and scaling of the grabber servomotor will also require carful due diligence on our part.

Run time testing exercised the LED matrixes. With the LED grids simple wiring errors could be made; this presented us with the opportunity to deploy a good wire color coding plan. Using color coded wiring reduced assembly errors, and help with tracing out wires during troubleshooting efforts.

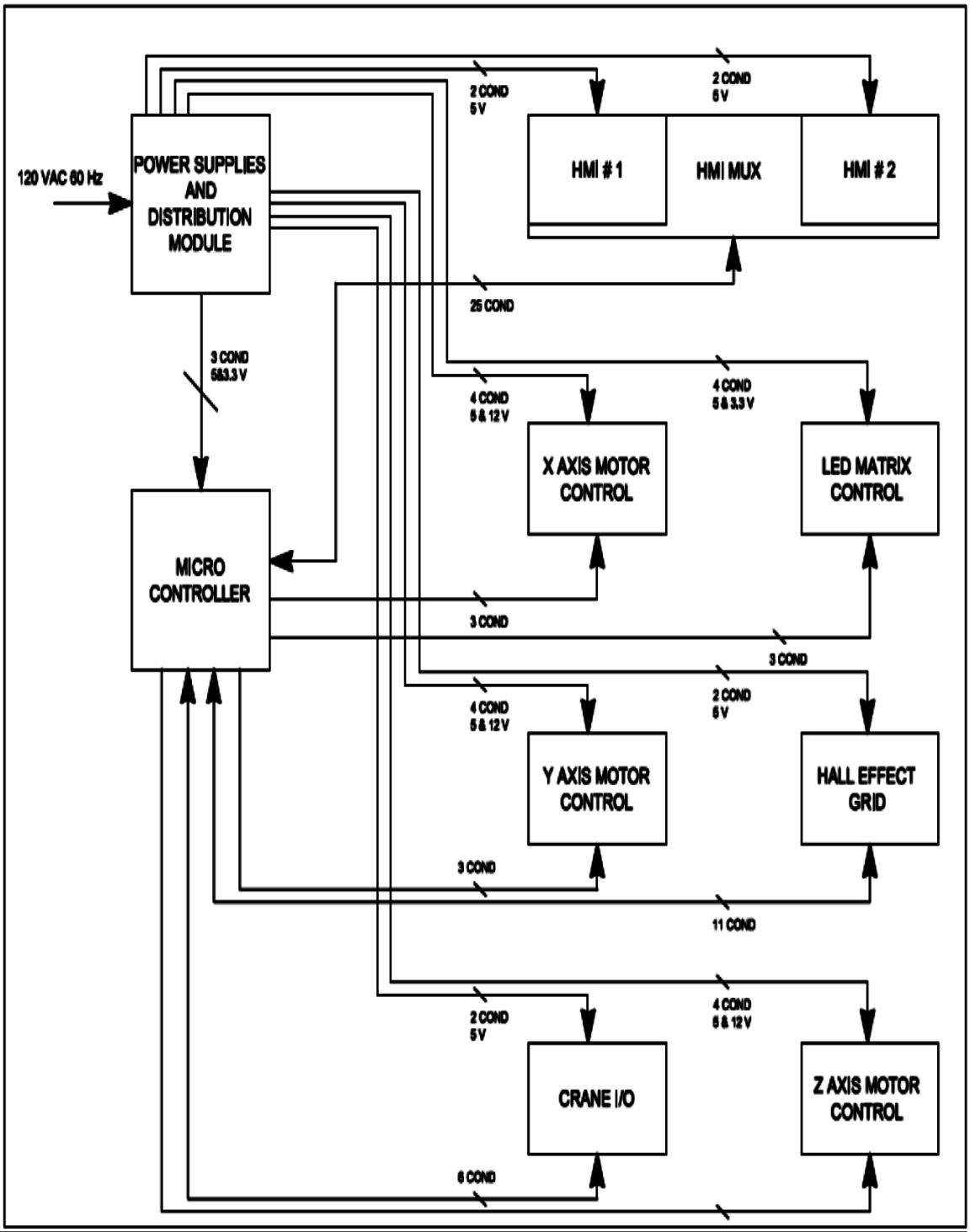

Figure 4.1-1: System interconnect diagram. This picture was drawn using AutoCAD educational software.

The real test of a good design comes when the end users who know nothing about your device start to use it. This is where you will discover things related to ease of use as well as equipment durability.

4.1.1 Theory of Operation

4.1.1.1 From the User's Perspective

Once everything was connected, built, and tested thoroughly, the Interactive Automated Chess Set looks like a rectangular box, elevated off of the table, with an A frame attached underneath the board and the crane suspended above the Set by the two arms that extend off of the A-Frame which allows for complete freedom of movement along the X, Y, and Z axes. The surface where the actual game play will commence looks like a plain sheet of plastic with no distinguishable markings other than the alphanumeric headings (1 through 8 and A through H) painted into the wooden frame on all four sides that frame the playing grid; embedded in the wooden frame, facing each user, are the human machine interfaces, labeled Player 1 and Player 2, complete with LCD screens and push buttons for movement selection.

In order to turn the system on and begin a new game, two switches must be thrown to provide power to the system and any button may be pressed from the Player 1 HMI terminal.

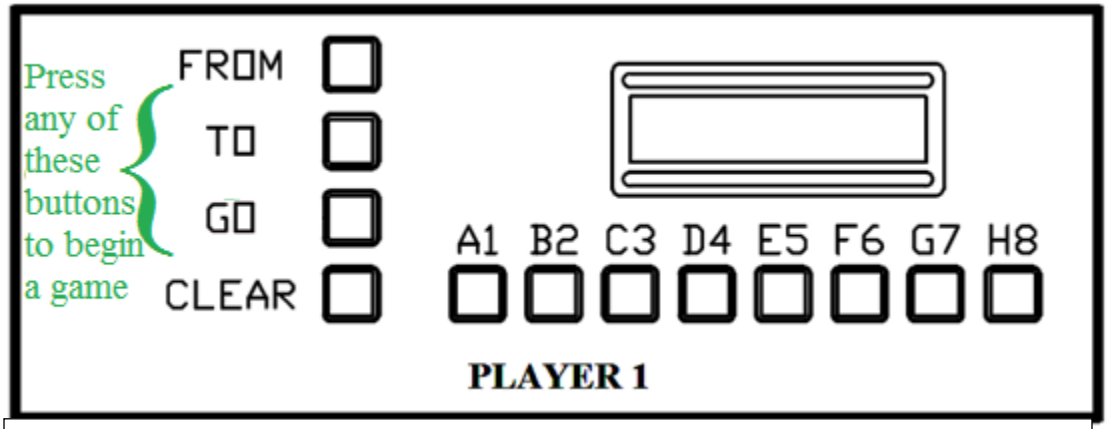

Figure 4.1.1.1-1: The HMI terminal for Player 1 highlighting all of the buttons that begins the game for the Interactive Automated Chess Set. This picture was drawn using AutoCAD educational software.

4.1.1.1.1 Player versus Player

Although all of the mechanical infrastructure is set and ready for this mode of game play it is not yet programmed and as such is not being implemented at this time although the game play should operate similarly as follows once it is up and running (with the displayed messages the same as the Player in Section 4.1.1.1.2 Player versus Computer below).

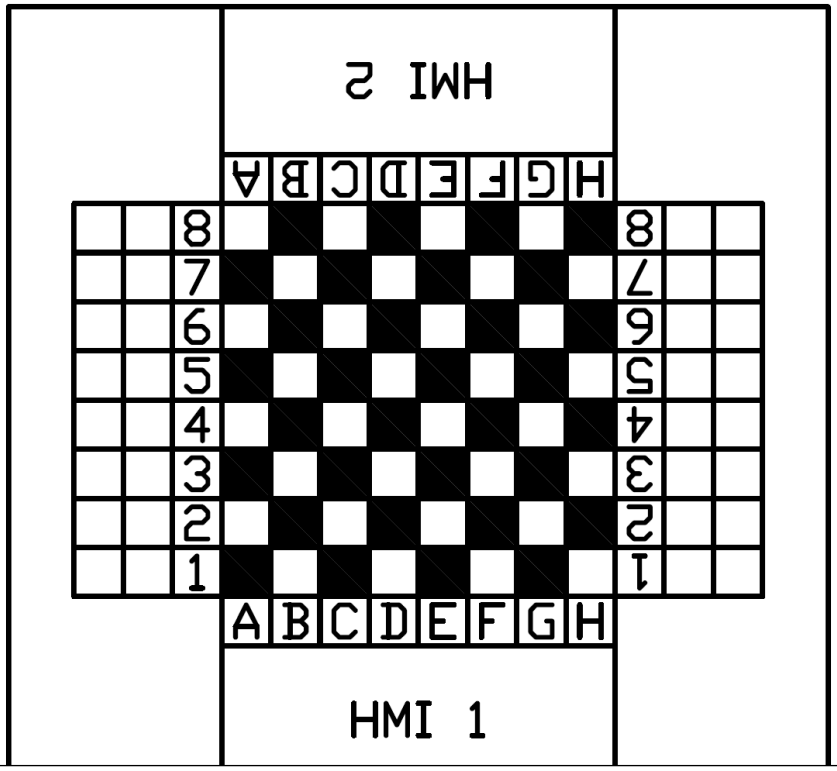

Figure 4.1.1.1-2: The chessboard that will be used for the Interactive Automated Chess Game depicting the locations of Player 1's terminal (HMI 1) and Player 2's terminal (HMI 2). This picture was drawn using AutoCAD educational software.

The terminal for Player 1 will prompt the user to make their first move ("Please press the FROM to begin a game" as shown below in Figure 4.1.1.1- 3) and the user will press the FROM button as shown below in Figure 4.1.1.1-4; the terminal should then display something to the effect of "Please select the letter corresponding to the piece you wish to move" (Figure 4.1.1.1-5) and, after the letter is selected, the terminal will prompt the user to "Please select the number corresponding to the piece you wish to move" (Figure 4.1.1.1-6). After the piece to be moved has been selected, the screen will prompt the user to check if the location corresponding to the piece they wish to play is correct ("You have chosen A2; Is this correct?" as shown below in Figure 4.1.1.1-7). If the input displayed is incorrect, or the user decides to change their selection, the user will then press CLEAR as shown in Figure 4.1.1.1-8 below and the terminal will again prompt the user to "Please select the letter corresponding to the piece you wish to move," going from there until the user is satisfied with the selection.

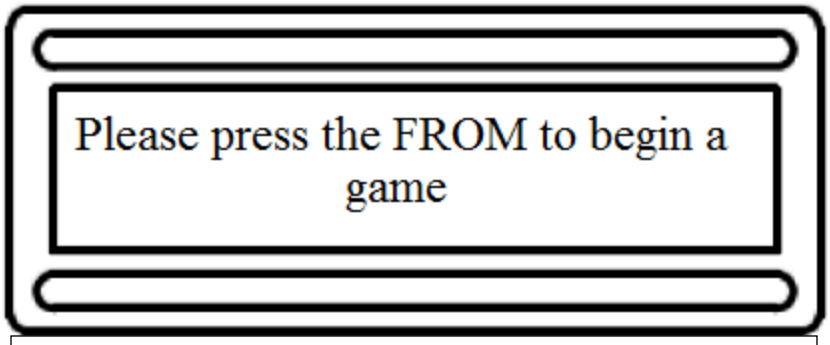

Figure 4.1.1.1-3: The LCD screen prompting the user to press the FROM button to begin a game.

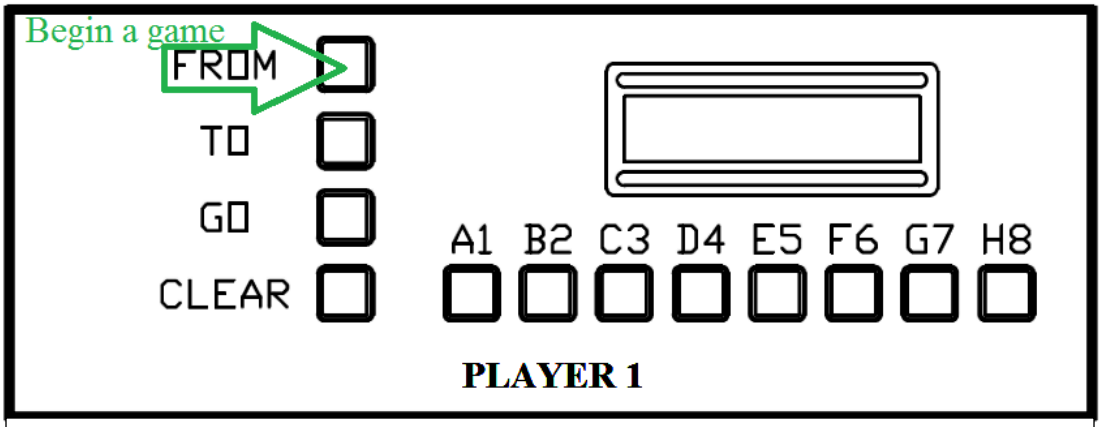

Figure 4.1.1.1-4: The HMI terminal for Player 1 highlighting the FROM button as the button to begin the game. This picture was drawn using AutoCAD educational software.

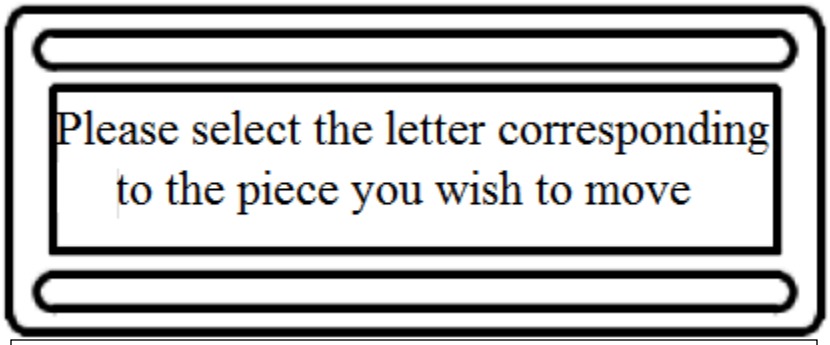

Figure 4.1.1.1-5: The LCD screen prompting the user to select the letter corresponding to the piece he or she wishes to move

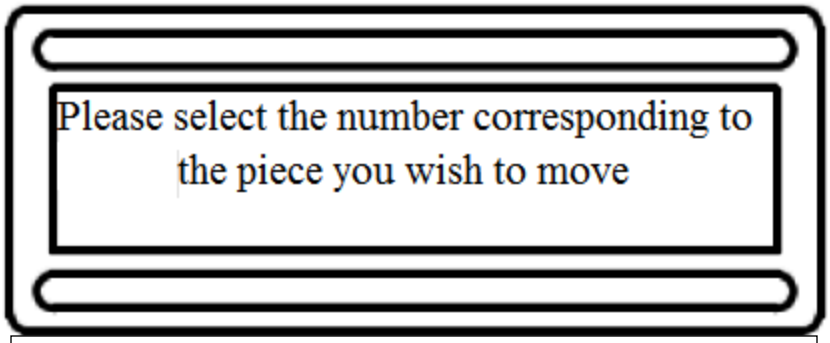

Figure 4.1.1.1-6: The LCD screen prompting the user to select the number corresponding to the piece he or she wishes to

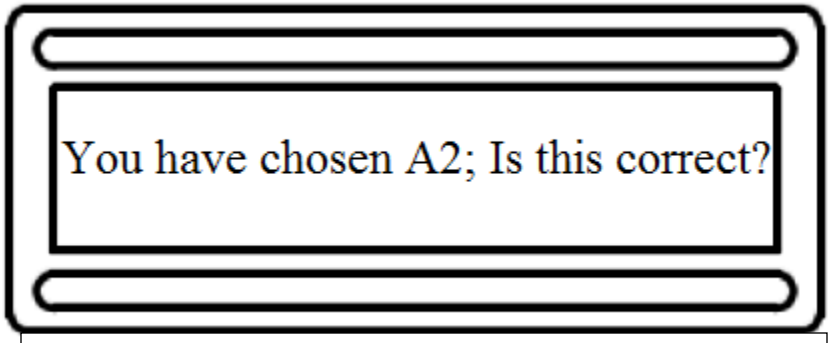

Figure 4.1.1.1-7: The LCD screen prompting the user to review the move he or she has chosen.

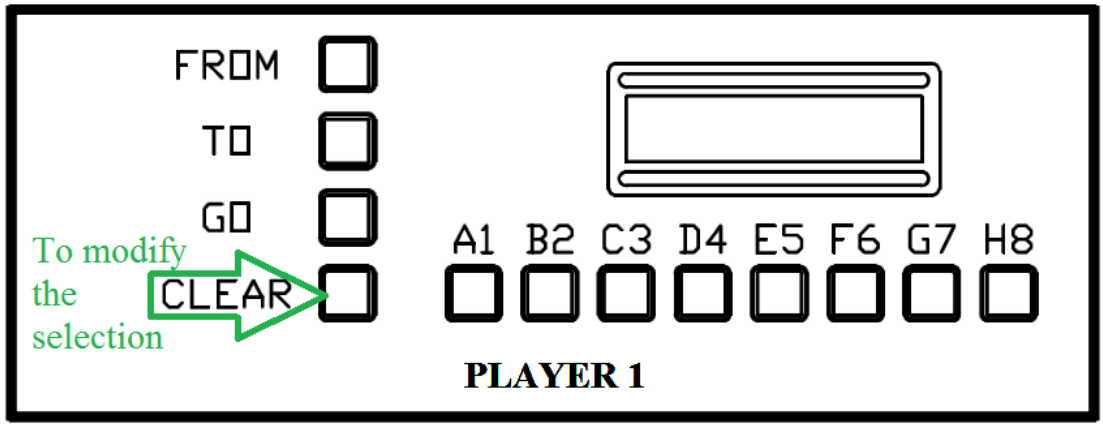

Figure 4.1.1.1-8: The HMI terminal for Player 1 highlighting the CLEAR button. This picture was drawn using AutoCAD educational software.

If the coordinates have been entered correctly then Player 1 will press the GO button as shown in Figure 4.1.1.1-9. The screen will then prompt Player 1 to press the TO button to select a destination, as shown below in Figure 4.1.1.1-10 and Figure 4.1.1.1-11, and the terminal will prompt the user for the destination ("Where would you like to move A2? Please select the letter corresponding to the

square you wish to move your piece to" (Figure 4.1.1.1-12) then after the letter is entered, "Please select the number corresponding to the square you wish to move your piece to" (Figure 4.1.1.1-13)). The terminal will again prompt the user to check if the location corresponding to the destination of the piece they wish to play is correct ("You have chosen A3; if you wish to move A2 to A3 please press GO now" (Figure 4.1.1.1-14)). If the input displayed is incorrect, or the user decides to change their selection, the user will then press CLEAR and the terminal will again prompt the user to "Please select the letter corresponding to the square you wish to move your piece to," going from there until the user is satisfied with the selection.

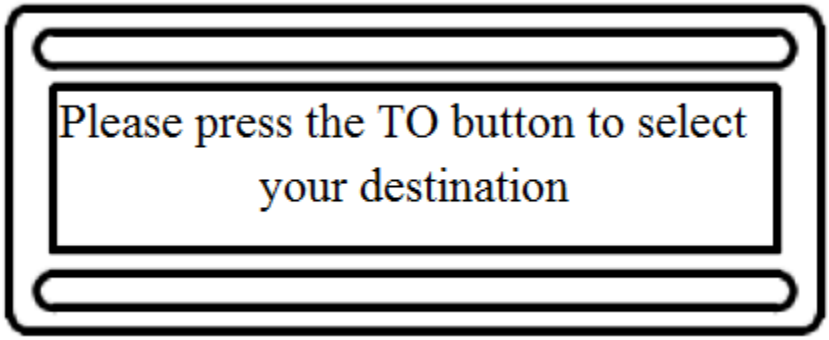

Figure 4.1.1.1-9: The LCD screen prompting the user to press the TO button to begin selecting the destination for the chess piece chosen

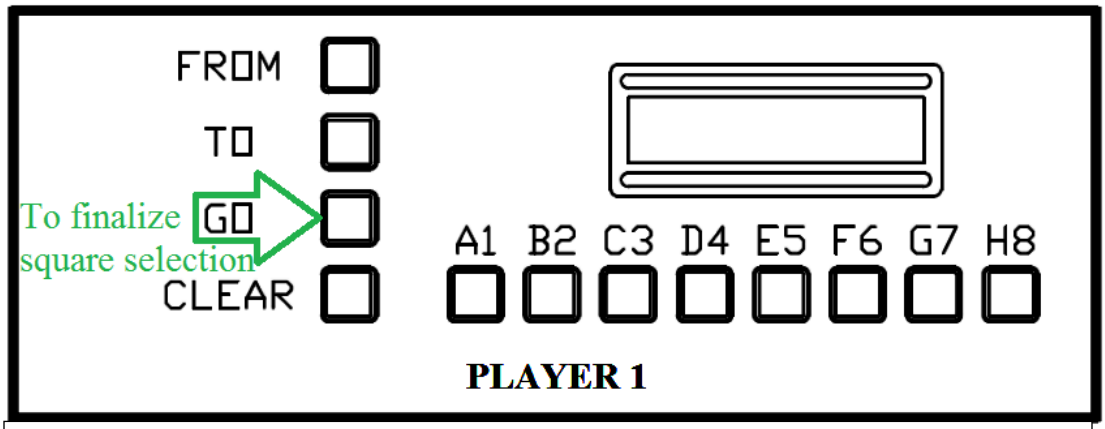

Figure 4.1.1.1-10: The HMI terminal for Player 1 highlighting the GO button for the purpose of finalizing the square selection. This picture was drawn using AutoCAD

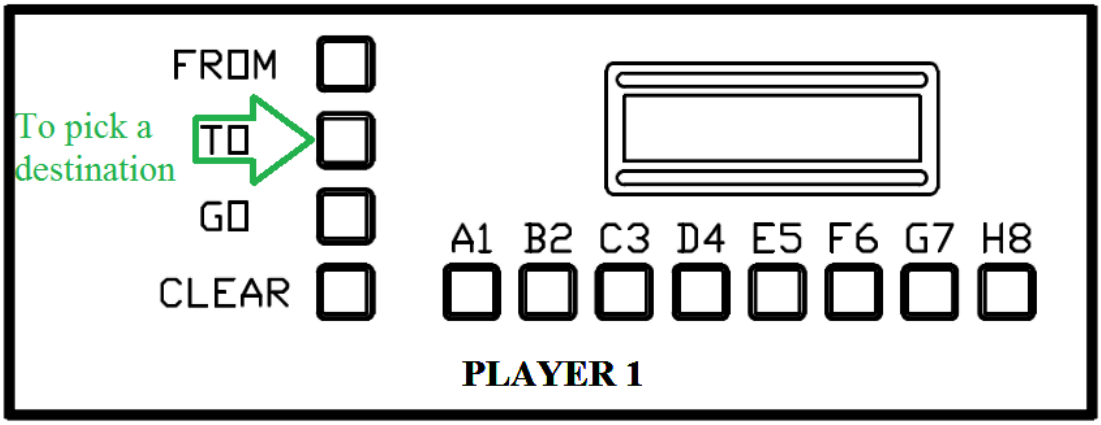

Figure 4.1.1.1-11: The HMI terminal for Player 1 highlighting the TO button so that Player 1 can pick the destination of the chess piece chosen. This picture was drawn using AutoCAD educational software.

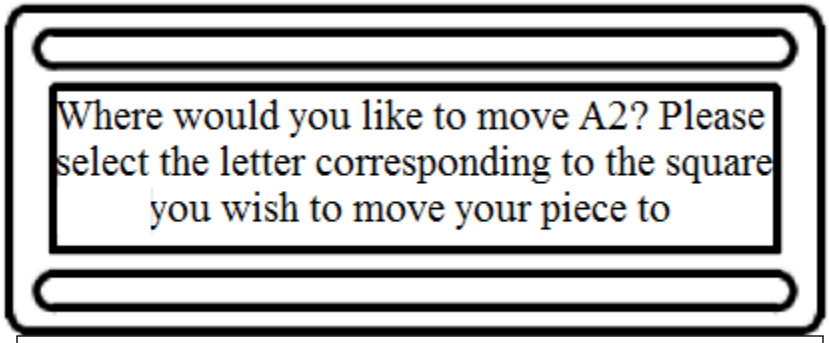

Figure 4.1.1.1-12: The LCD screen prompting the user to select the letter of the destination of their piece.

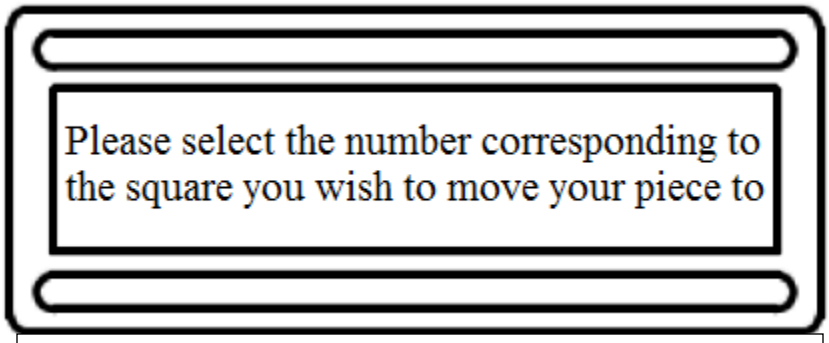

Figure 4.1.1.1-13: The LCD screen prompting the user to select the number of the destination of their piece.

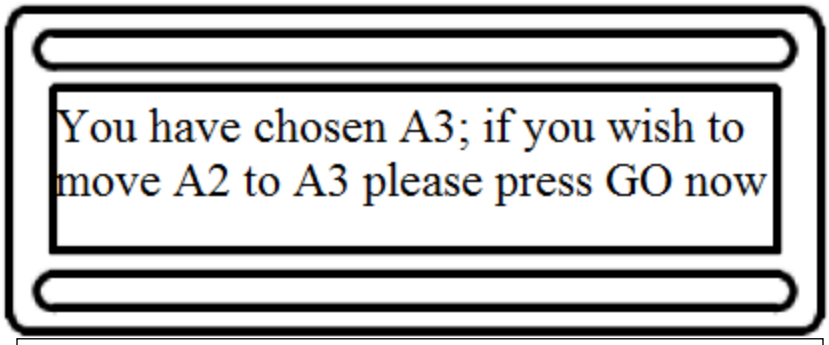

Figure 4.1.1.1-14: The LCD screen prompting the user to review the move he or she has chosen.

If the user is satisfied with the selection, then he or she will press the GO button; once the GO button has been pressed the move has been submitted and cannot be changed. If the movement causes the piece at the destination to be removed from play then the claw will go to the destination square first, pick up the piece to be removed, and put it in the corresponding player's graveyard (in this case Player 1); then the claw will go to the square containing the piece to be moved, relocate it to the destination square, and then return to the claw's default position. If the player's movement does not result in a piece being removed from play then the claw will go to the square containing the piece to be moved and relocate it to the destination square immediately before returning to its default position.

Once Player 1 has concluded his or her turn, the terminal for Player 2 will prompt the user to "Please press FROM to begin your turn" as shown below in Figure 4.1.1.1-15 and Figure 4.1.1.1-16. Just as for Player 1, Player 2's terminal should then display the "Please select the letter corresponding to the piece you wish to move" message (Figure 4.1.1.1-17) and, after the letter is selected, the terminal will prompt Player 2 to "Please select the number corresponding to the piece you wish to move" (Figure 4.1.1.1-18). After the piece to be moved has been selected, the screen will prompt the user, just as it did with Player 1, to check if the location corresponding to the piece they wish to play is correct ("You have chosen H7; Is this correct?" as shown below in Figure 4.1.1.1-19). If the input displayed is incorrect, or if Player 2 decides to change his or her selection, the user will then press CLEAR and the terminal will again prompt the user to "Please select the letter corresponding to the piece you wish to move," going from there until Player 2 is satisfied with the selection.

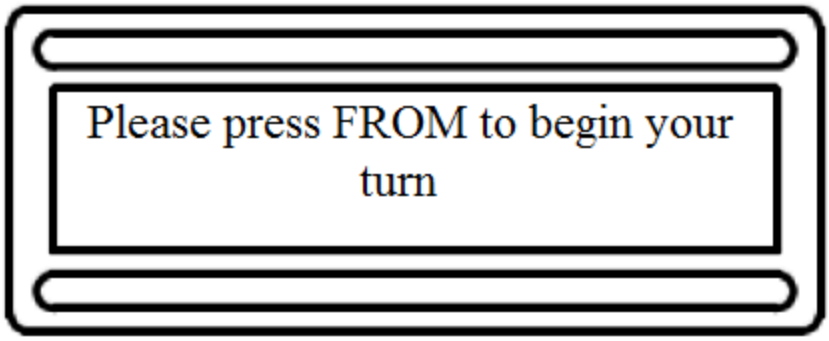

Figure 4.1.1.1-15: The LCD screen prompting the user press FROM to begin his or her turn.

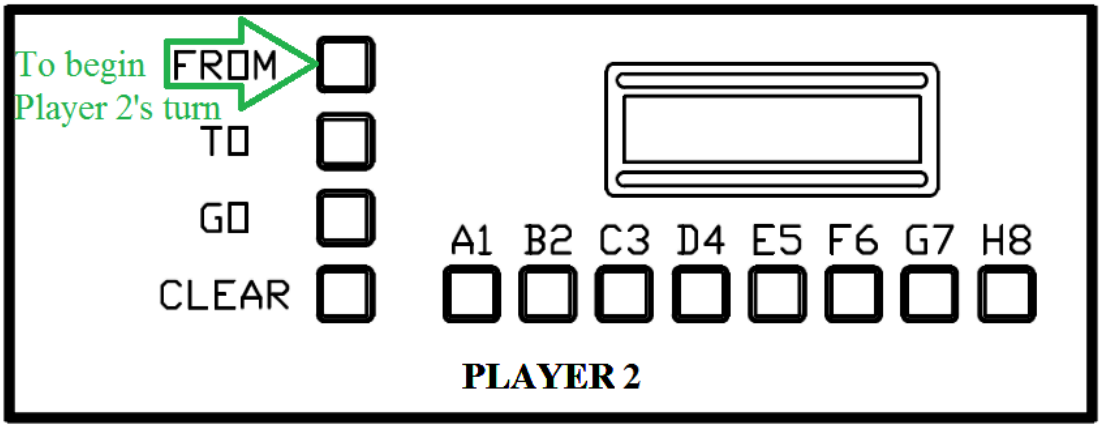

Figure 4.1.1.1-16: The HMI terminal for Player 2 highlighting the FROM button as the button to begin his or her turn. This picture was drawn using AutoCAD educational

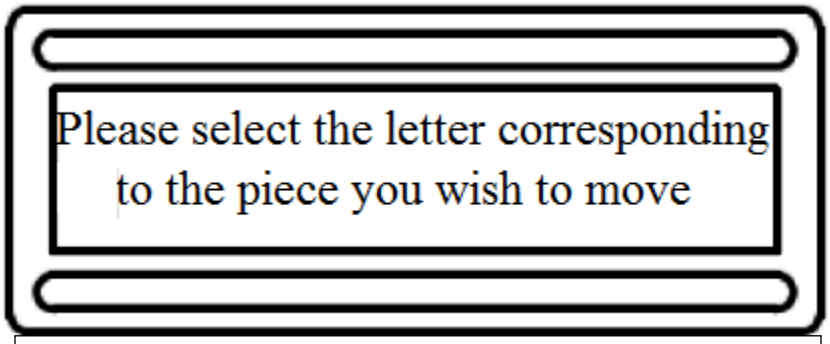

Figure 4.1.1.1-17: The LCD screen prompting the user to select the letter of the piece he or she wishes to move.

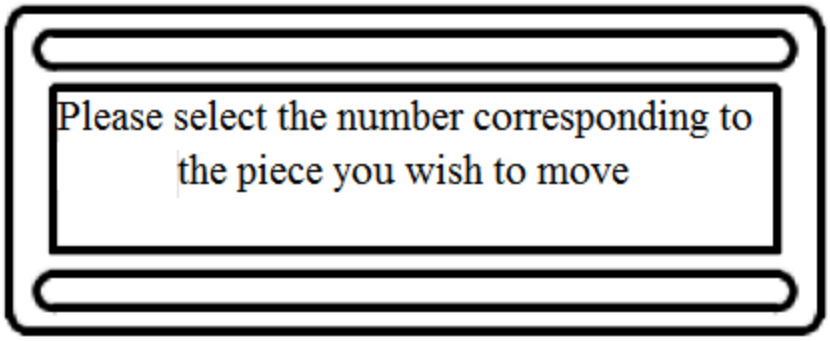

Figure 4.1.1.1-18: The LCD screen prompting the user to select the number of the piece he or she wishes to move.

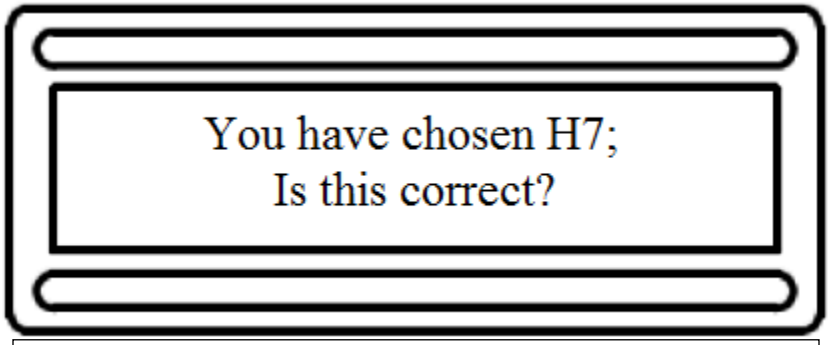

Figure 4.1.1.1-19: The LCD screen prompting the user to review the move he or she has chosen.

If the coordinates have been entered correctly then Player 2 will press the GO button as shown in Figure 4.1.1.1-20 below. The screen will then prompt Player 2 to press the TO button to select a destination (Figure 4.1.1.1-21 and Figure 4.1.1.1-22), again like during Player 1's turn, and the terminal will prompt the user for the destination ("Where would you like to move H7? Please select the letter corresponding to the square you wish to move your piece to" as shown in Figure 4.1.1.1-23 then after the letter is entered, "Please select the number corresponding to the square you wish to move your piece to" (Figure 4.1.1.1-24)). The terminal will again prompt the user to check if the location corresponding to the destination of the piece they wish to play is correct ("You have chosen H6; if you wish to move H7 to H6 please press GO now") as shown in Figure 4.1.1.1-25 below. If the input displayed is incorrect, or the user decides to change their selection, the user will then press CLEAR as shown in Figure 4.1.1.1-26 below and the terminal will again prompt Player 2 to "Please select the letter corresponding to the square you wish to move your piece to," going from there until the user is satisfied with the selection.

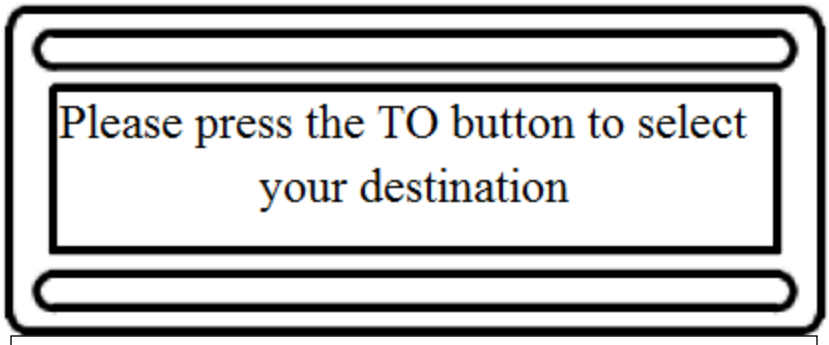

Figure 4.1.1.1-20: The LCD screen prompting the user to press the TO button to begin selecting the destination for the chess

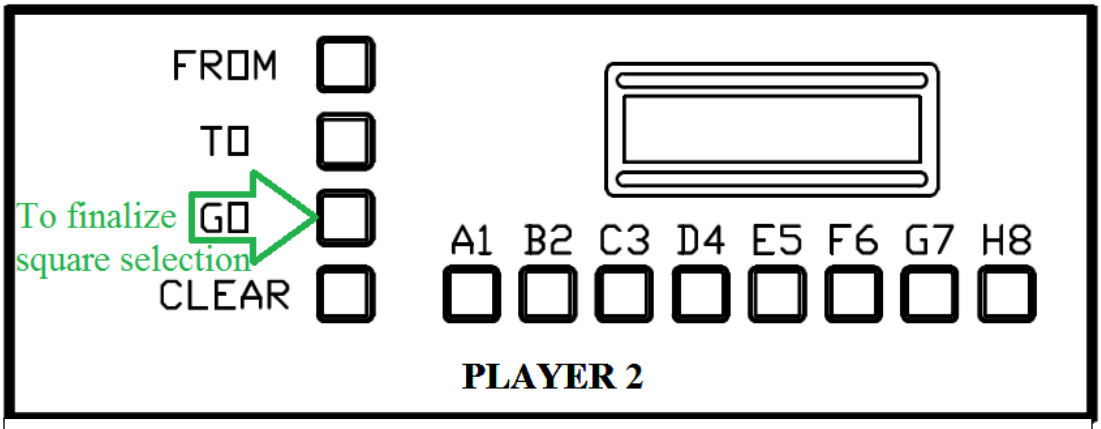

Figure 4.1.1.1-21: The HMI terminal for Player 2 highlighting the GO button for the purpose of finalizing the square selection. This picture was drawn using AutoCAD

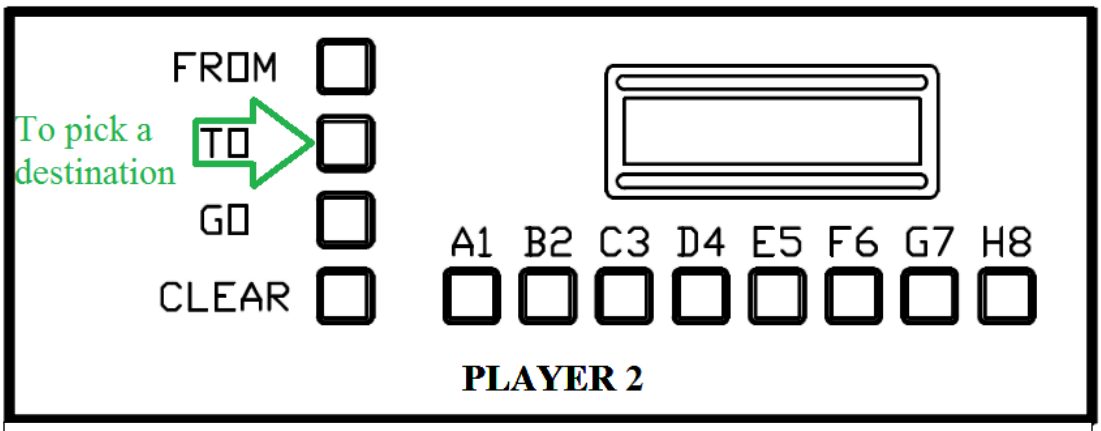

Figure 4.1.1.1-22: The HMI terminal for Player 1 highlighting the TO button so that Player 2 can pick the destination of the chess piece chosen. This picture was drawn using AutoCAD educational software.

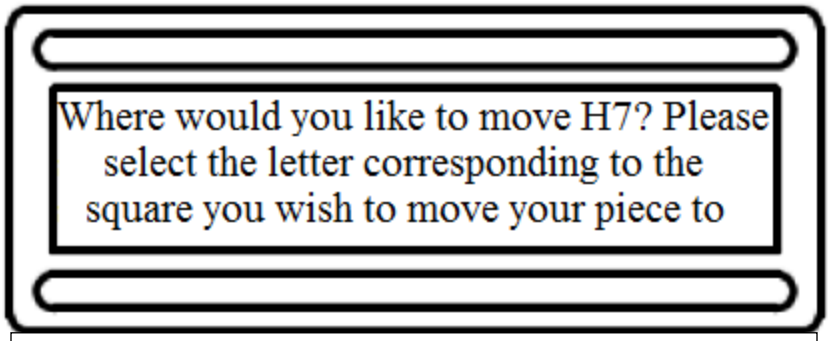

Figure 4.1.1.1-23: The LCD screen prompting the user to select the letter of the destination of the piece he or she wishes to

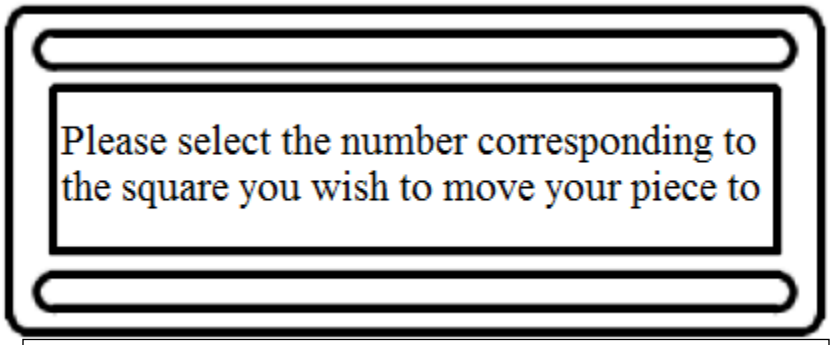

Figure 4.1.1.1-24: The LCD screen prompting the user to select the number the destination of the piece he or she wishes to

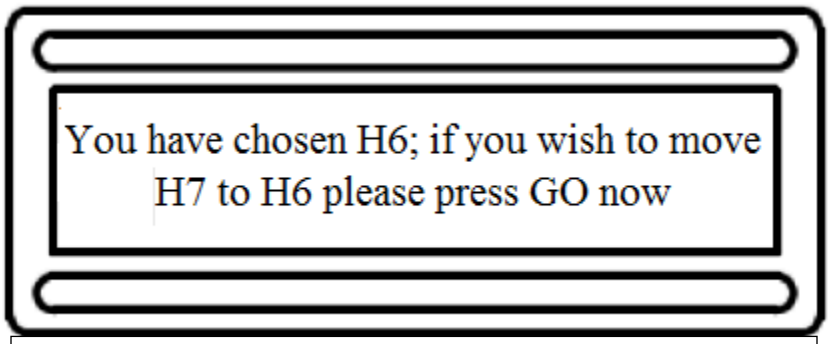

Figure 4.1.1.1-25: The LCD screen prompting the user to review the move he or she has chosen.

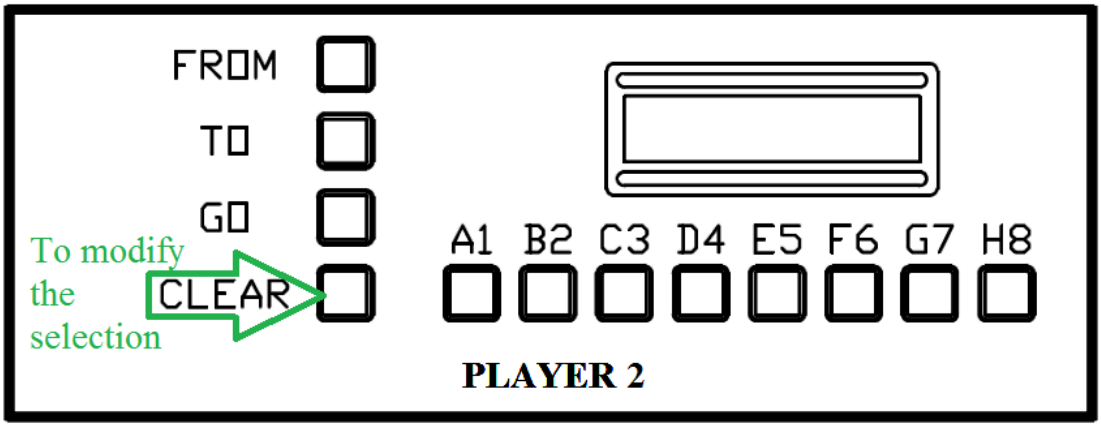

Figure 4.1.1.1-26: The HMI terminal for Player 2 highlighting the CLEAR button for the purpose of modifying the original selection. This picture was drawn using AutoCAD

If Player 2 is satisfied with the selection, then he or she will press the GO button just like during Player 1's turn. If the movement causes the piece at the destination to be removed from play then the claw will go to the destination square first, pick up the piece to be removed, and put it in the corresponding player's graveyard (in this case Player 2); then the claw will go to the square containing the piece to be moved, relocate it to the destination square, and then return to the claw's default position. If the player's movement does not result in a piece being removed from play then the claw will go to the square containing the piece to be moved and relocate it to the destination square immediately before returning to its default position.

4.1.1.1.2 Player versus Computer

At the start of the game, and for each successive turn, the terminal for Player 1 prompts the user to make their first move just as in the Player versus Player mode, (shown below in Figure 4.1.1.2-1 and 4.1.1.2-3) and the user will press any of the buttons as shown below in Figure 4.1.1.2-2; the terminal then prompts the user to select the letter of his or her choice (Figure 4.1.1.2-4) and, after the letter is selected, the terminal will prompt the user to select the number of his or her choice (Figure 4.1.1.2-5). After the piece to be moved has been selected, the screen will prompt the user to check if the location corresponding to the piece they wish to play is correct (shown below in Figure 4.1.1.2-6). If the input displayed is incorrect, or the user decides to change their selection, the user will then press CLEAR the terminal will again prompt the user to select the letter of his or her choice and going from there until the user is satisfied with the selection.

Figure 4.1.1.2-1: The LCD screen prompting the user to press the FROM button to begin a game.

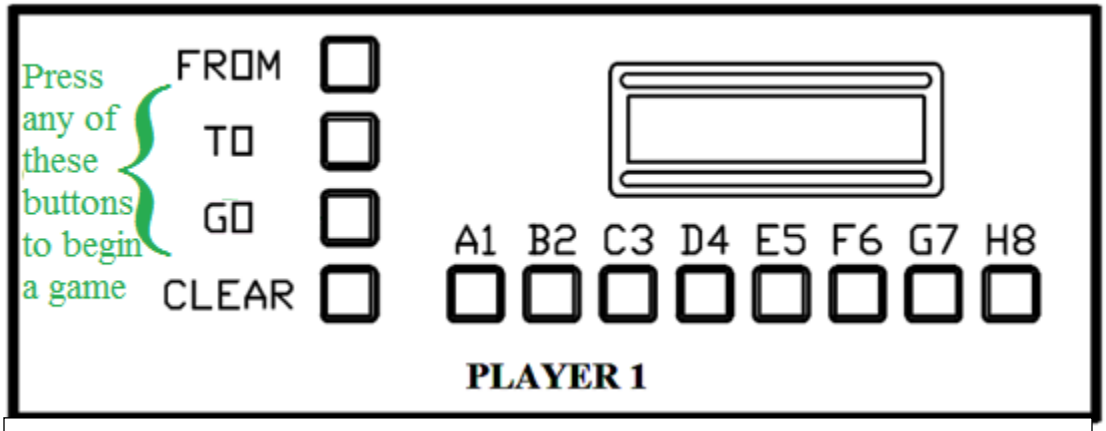

Figure 4.1.1.2-2: The HMI terminal for Player 1 highlighting the four main buttons. This picture was drawn using AutoCAD educational software.

Figure 4.1.1.2-3: The LCD screen prompting the user to press FROM to being his or her move

Figure 4.1.1.2-4: The LCD screen prompting the user to select the letter corresponding to the piece he or she wishes to move

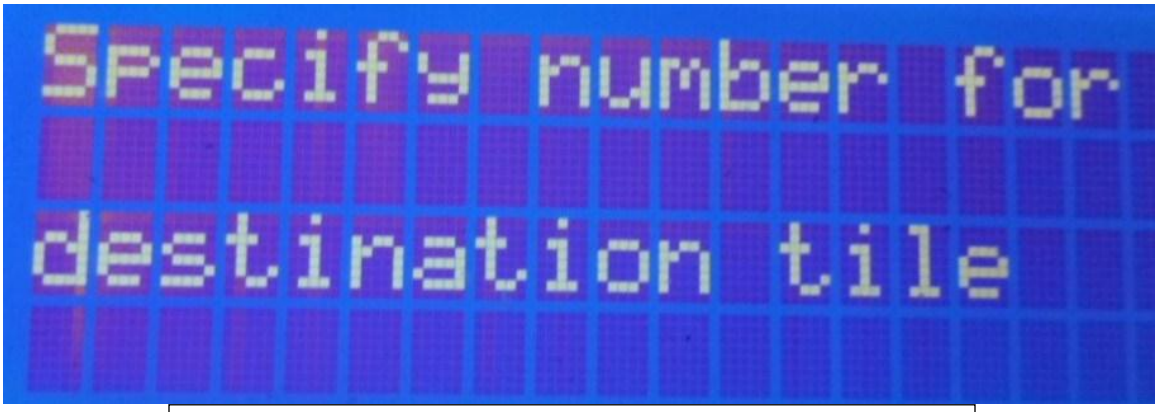

Figure 4.1.1.2-5: The LCD screen prompting the user to select the number corresponding to the piece he or she wishes to

Figure 4.1.1.2-6: The LCD screen prompting the user to review the move he or she has chosen and to hit the GO button if correct.

Figure 4.1.1.2-7: The LCD screen displaying that the Interactive Automated Chess Set is moving the Player's piece

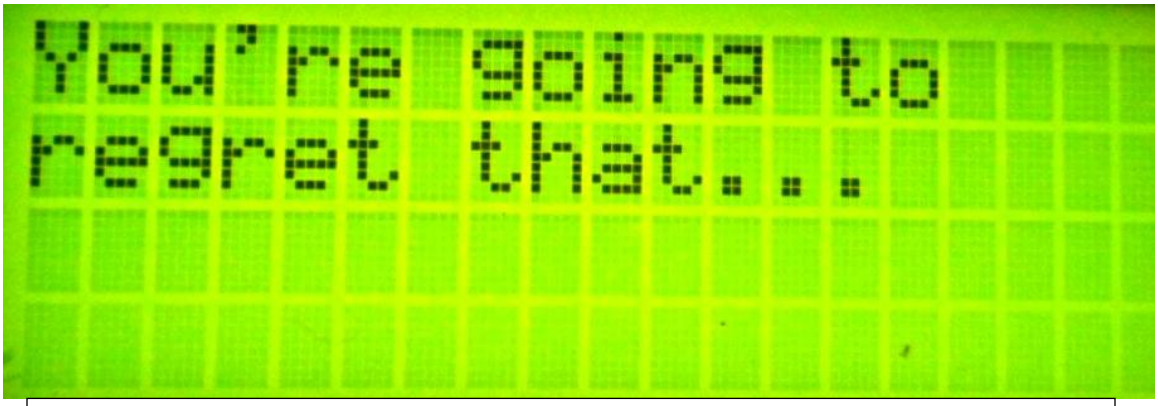

Figure 4.1.1.2-8: The LCD screen displaying the AI's opinion on the Player's move on the HMI 2 screen

If the coordinates have been entered correctly then Player 1 will press the GO button and the LCD screen will display that the piece is being moved by the Interactive Automated Chess set as shown in Figure 4.1.1.2-7. The AI's opinion on the Player's move on is displayed on the HMI 2 screen as shown in Figure 4.1.1.2-8 above. If the movement caused the piece at the destination to be removed from play then the claw will go to the destination square first, pick up the piece to be removed, and put it in the graveyard then the claw will go to the square containing the piece to be moved, relocate it to the destination square, and then return to the claw's default position. If the player's movement does not result in a piece being removed from play then the claw will go to the square containing the piece to be moved and relocate it to the destination square immediately before returning to its default position.

After Player 1 has completed his or her turn, and a short pause indicating that the Interactive Automated Chess Set is "thinking" about its move, HMI 1 will display that the AI's piece is being moved (Figure 4.1.1.2-9) and the claw will descend as needed, first going to the destination square if one of Player 1's pieces are to be removed from play and placed into the grave yard, and grabbing the piece the

computer's program has chosen and relocating it to the chosen destination square. The LCD terminal for Player 1 will then prompt the user to press any button as shown below in Figure 4.1.1.2-10 and the game will continue back and forth until its conclusion.

Figure 4.1.1.2-9: The LCD screen displaying that the Interactive Automated Chess Set is moving the AI's piece

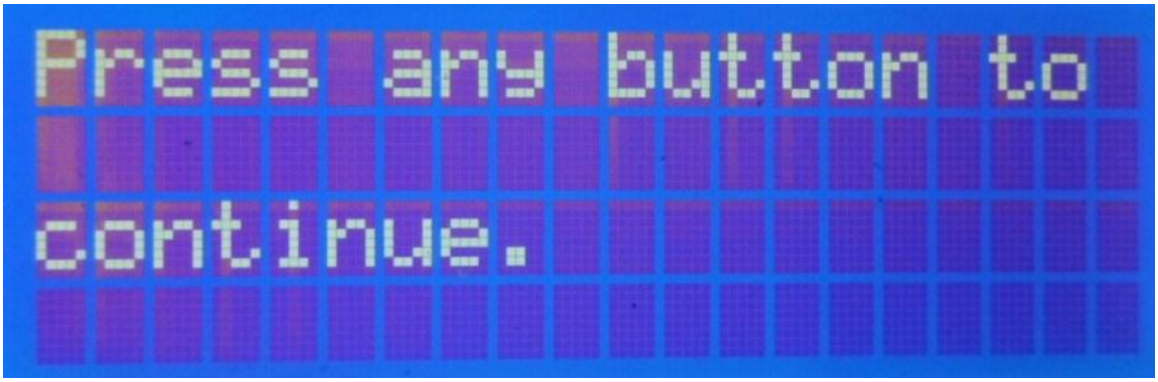

Figure 4.1.1.2-10: The LCD screen prompting the user press any button to begin his or her turn.

4.1.1.1.3 Computer versus Computer

Although all of the mechanical infrastructure is set and ready for this mode of game play it is not yet programmed and as such is not being implemented at this time. It was intended for demonstration purposes and since that did not add anything new to our project it was removed from the final version

4.1.1.2 From the Machine's Perspective

The world, from the machine's perspective, begins when the switches are flipped and the Interactive Automated Chess Set begins to boot up. After displaying the welcome message the computer will eagerly await the user's choice (once the Player versus Player mode is up and running) in the type of game; for both the Player versus Player and the Player versus computer mode, the game will start off almost identically. After the machine reads in the FROM and TO choices (and is confirmed with a GO selection by Player 1), the choice will be sent to the microcontroller where it will be evaluated for validity. If the move is in fact valid then the microcontroller's AI will log the move in its memory and send the appropriate signals to the motors that control the claw as well as to the LEDs residing under the appropriate square or squares and the piece or pieces (in the event that a piece is being removed from play) will be moved as necessary. If the movement is not, in fact, valid then the computer will search through the appropriate tables in its programming to find the exact error, display the error and how to correct it to the user, and await the user's correction and acknowledgment of said error.

Once Player 1's turn has been completed then Player 2's turn will continue in the same fashion, if a Player versus Player mode has been selected, or the computer will then take it's turn following the same steps as Player 1's turn with the exception of the input into the Player 2 terminal, skipping straight to the cataloging and motion portion of the program before returning control to the Player 1 terminal.

4.2 Acceptance Testing

We developed a written test plan, called the Acceptance Test Plan (ATP), in the design phase where the Acceptance Test Results (ATR) are the final test results. We printed out the ATP document, complete all of the tests, and made the test result available at our final presentation. Each sub-system has its own section in the ATP as well. By following good testing procedures potential problems were discovered and corrected in the test and adjust phase rather than during game play on final presentation day. The process of writing the test plan forces us to think of critical functionality that must be validated. Industry uses acceptance testing to confirm that their designs function as designed where typical acceptance tests start with the I/O check out. The tester goes through every controller I/O point and verifies, with a meter and with the circuit de-energized, that the circuit is in fact wired as per the wiring diagram drawing or the wire list. It is common for the tester to find problems with the wiring of custom systems where just one or a few systems are produced. Wiring problems can be some of the hardest problems to trouble shoot. Suppose that you are expecting a specific input to turn on when a limit switch has been contacted, but the input was actually wired to a user input switch. Now your machine is operating in an

unpredictable and possibly unsafe state. It is always the best practice to take the time to properly validate the wiring system as the first phase of testing.

The next phase of testing is normally to conduct similar I/O tests with the circuit energized. The tester now verifies when a switch is pushed the controller input goes high and low accordingly. If the system has an emergency stop circuit, that system will be closely checked before any motion functionality is done; now basic functionality testing can begin. In the functionality testing phases the tester will exercise a motion axis from point A to point B. Again if the system has an E-Stop circuit it will be tested first and then again while the system is actually in motion where all of the normal modes of operation are tested.

The final phase of acceptance testing is conducting tests on all of the failure modes. This is how the designer can prove that the control system can detect the fault, respond accordingly, and most importantly in a safe manor; if the design requirements call for user fault messaging it will be verified here. If a single point failure mode results in a condition that will not be automatically detected at the time of the original fault, it is said to be a latent failure mode; in the industry, engineering analysis typically identifies such conditions. Preventative maintenance inspections and testing of such systems must be done to insure the machine operates in a safe state and therefore the possible failure modes must be tested. This is where things can get interesting for the programmer. Consider when an over travel condition is detected on a motion axis and the DRS calls for the axis to E-Stop automatically. In failure modes, the testing condition are simulated by forcing the over travel sensor low resulting in a number of messages generated. Only the original fault condition should be displayed. The programmer now must mask out all other fault messages that occur as a result of the contingency response. Also the message should be latched until it has been acknowledge be the user. A failure condition where a fault quickly pops up on the screen and goes away will leave the user wondering what happened. Good testing identifies a number of problems and allows the team ample opportunity to correct them during the testing phases. It also helps identify the things you never thought about.

Some systems such as production lines use redundant sensors to detect a fault condition and continue to run until maintenance can be done. Other systems, such as ride control systems, are designed to detect the fault and shut down. In this case the safety of the rider is paramount, and the remaining good working sensor has now been reduced to a single point failure mode. Failure Modes and Effects Analysis (FMEA) is done to validate that a given mitigation will detect the fault and if the mitigation fails that the failure can also can be detected. If a FMEA has been done all of the failure modes identified in the analysis will need to be included in the ATP.

The software prototyping and test strategy is also an integral part to the development of our system. We will prototype each piece of software as a unit

and write tests for it, ensuring those tests pass before moving on to different pieces. We begin by setting our debugger to launch the AVR simulator. Atmel Studio allows us to view the program counter, stack pointer, status registers, and more contents of our virtual processor in debug mode. We can view the contents of memory as well as watch and step through variables. In this way we will begin testing the AVR ported Micro-Max chess engine to ensure it is ready for deployment. We can test the chess engine in this way because it doesn't require any peripherals to run, and we can simply active its methods from a temporary main file.

4.2.1 I/O Checkout

The purpose of the I/O Checkout Acceptance Test is to validate the microcontroller wiring system has been correctly wired as per the wiring diagram. Testing will be done with an Ohm Meter and every point will Ohmed out from the field device to the appropriate microcontroller pin. The I/O List will be used as a check list for completing the task. All test steps are clearly defined in Table 4.2.1-1 below.

Item #	Passed	Test Description
4.2.1.1		Connect an Ohm meter across the 12 VDC power and the 12 VMC terminals. Push down and pull up on the E-Stop switch, and activate by hand each of the over travel limit switches one at a time. Each time the circuit is interrupted verify the results match what is shown on the meter display.
4.2.1.2		Use the I/O List in Section 6.4 of the Appendix as a check list to complete this portion of the I/O Check out. This test MUST be conducted with the machine fully de-energized. Ohm out every circuit from the field device to the microcontroller pin. All circuits must have a resistance reading of less than 3 Ohms.
4.2.1.3		Use the I/O List in Section 6.4 of the Appendix as a check list to complete this portion of the I/O Check out. This test will be conducted with the circuit energized. Care must be taken with meter lead placement. Failure to correctly place the meter lead on the correct terminal without contacting other energized circuits could result in equipment damage. Reference the Type and Configuration columns on the I/O List. If the type is output the measurements will be made on the field device. If the type is input the measurement will be made at the microcontroller board input pin. If the configuration is sinking the positive voltmeter lead will be connected to 5 VCC. If the configuration is scouring the negative voltmeter lead will be connected to DC Common. If the signal is an output force the output on and off and measure the voltage. If the signal is an input toggle the field device while measuring the voltage. Verify each signal is working correctly.

Table 4.2.1-1: I/O checkout test plan procedures

4.2.2 Mechanical Assembly

The scope of the Mechanical Assembly Acceptance Testing is to validate proper operation and design of our machine. We will check machine clearances, gear alignment, verify that our slide rails are free of binding. Our wire management system will be checked for binding, pinching, and proper flexing in this section. All test steps are clearly defined in Table 4.2.2-1 below.

Item #	Passed	Test Description
4.2.2.1		With the motor controls powered down; push the A frame structure back and forth along the Y axis to and from the mechanical hard stops.
4.2.2.2		Verify that there is no noticeable binding.
4.2.2.3		Verify that the Y axis wire management system moves freely with no pinching, rubbing or extreme pulling.
4.2.2.4		Push the X axis carriage back and forth along the Tee rail. Verify that the carriage moves freely with no binding.
4.2.2.5		Verify that the eccentric gear assembly has no issues with operational clearances.
4.2.2.6		Verify that the X axis wire management system moves freely with no pinching, rubbing or extreme pulling.
4.2.2.7		Rotate the eccentric gear stopping in the fully up and fully down position. Verify that the Z axis can hold position in the up position.
4.2.2.8		Verify that the Z axis wire management system moves freely with no pinching, rubbing or extreme pulling.
4.2.2.9		Attach a plumb bob to the center point of the pincher, and move machine to the center of each square. Verify that the machine can complete all possible moves with no mechanical issues noted.
4.2.2.10		Verify that the Y axis spur gear can move across the Y axis rack gear with proper alignment and maintain uniform pressure.
4.2.2.11		Verify that the X axis spur gear can move across the Y axis rack gear with proper alignment and maintain uniform pressure.

Table 4.2.2-1: Mechanical assembly test plan procedures

4.2.3 Human Machine Interface

The scope of our Human Machine Interface acceptance testing shall verify the correct functionality of both HMI #1 and HMI #2. All user input switches and all multiplexing components will be verified for proper operation. Both LCD displays will be tested for proper operation as well. Reference the wiring diagram in section 6.3 of the appendix for a better idea of how each HMI will be wired. Due to the extensive amount of testing required in this section the test procedures will be split into three separate Tables starting with Table 4.2.3-1 below.

Item #	Passed	Test Description
4.2.3.1		With the machine energized measure voltage with respect to DC common across switch S1A1. Verify the reading is 5.5 to 4.5 VDC. Push the button in and verify the voltage is less than .03 VDC
4.2.3.2		Repeat test 4.2.3.1 for each HMI user input switch.
4.2.3.3		With all HMI user input switches released, force output PE0 on. Push and hold switch S1A1, and measure voltage with respect to DC Common on each of the three inputs PD5, PJ0, & PJ1. Verify that PD5, PJ0 and PJ1 reflect the BCD 7 (1,1,1). Remove force on PE0.
4.2.3.4		With all HMI user input switches released, force output PE0 on. Push and hold switch S1B2, and measure voltage with respect to DC Common on each of the three inputs PD5, PJ0, & PJ1. Verify that PD5, PJ0 and PJ1 reflect the BCD 6 (1,1,0). Remove force on PE0.
4.2.3.5		With all HMI user input switches released, force output PE0 on. Push and hold switch S1C3, and measure voltage with respect to DC Common on each of the three inputs PD5, PJ0, & PJ1. Verify that PD5, PJ0 and PJ1 reflect the BCD 5 (1,0,1). Remove force on PE0.
4.2.3.6		With all HMI user input switches released, force output PE0 on. Push and hold switch S1D4, and measure voltage with respect to DC Common on each of the three inputs PD5, PJ0, & PJ1. Verify that PD5, PJ0 and PJ1 reflect the BCD 4 (1,0,0). Remove force on PE0.
4.2.3.7		With all HMI user input switches released, force output PE0 on. Push and hold switch S1E5, and measure voltage with respect to DC Common on each of the three inputs PD5, PJ0, & PJ1. Verify that PD5, PJ0 and PJ1 reflect the BCD 3 (0,1,1). Remove force on PE0.
4.2.3.8		With all HMI user input switches released, force output PE0 on. Push and hold switch S1F6, and measure voltage with respect to DC Common on each of the three inputs PD5, PJ0, & PJ1. Verify that PD5, PJ0 and PJ1 reflect the BCD 2 (0,1,0). Remove force on PE0.
4.2.3.9		With all HMI user input switches released, force output PE0 on. Push and hold switch S1G7, and measure voltage with respect to DC Common on each of the three inputs PD5, PJ0, & PJ1. Verify that PD5, PJ0 and PJ1 reflect the BCD 1 (0,0,1). Remove force on PE0.
4.2.3.10		With all HMI user input switches released, force output PE0 on. Push and hold switch S1H8, and measure voltage with respect to DC Common on each of the three inputs PD5, PJ0, & PJ1. Verify that PD5, PJ0 and PJ1 reflect the BCD 0 (0,0,0). Remove force on PE0.

Table 4.2.3-1: HMI #1 user designation zone inputs test plan procedures

Once the tests contained in 4.2.3-1 have been successfully completed, the same tests will need to be conducted for the HMI # 2 user destination zone inputs. The acceptance test procedures for the HMI # 2 user destination zone inputs can be referenced in Table 4.2.3-2 below.

Item #	Passed	Test Description
4.2.3.11		With all HMI user input switches released, force output PF0 on. Push and hold switch S2A1, and measure voltage with respect to DC Common on each of the three inputs PE5, PE6, & PE7. Verify that PE5, PE6 and PE7 reflect the BCD 7 (1,1,1). Remove force on PF0.
4.2.3.12		With all HMI user input switches released, force output PF0 on. Push and hold switch S2B2, and measure voltage with respect to DC Common on each of the three inputs PE5, PE6, & PE7. Verify that PE5, PE6 and PE7 reflect the BCD 6 (1,1,0). Remove force on PF0.
4.2.3.13		With all HMI user input switches released, force output PF0 on. Push and hold switch S2C3, and measure voltage with respect to DC Common on each of the three inputs PE5, PE6, & PE7. Verify that PE5, PE6 and PE7 reflect the BCD 5 (1,0,1). Remove force on PF0.
4.2.3.14		With all HMI user input switches released, force output PF0 on. Push and hold switch S2D4, and measure voltage with respect to DC Common on each of the three inputs PE5, PE6, & PE7. Verify that PE5, PE6 and PE7 reflect the BCD 4 (1,0,0). Remove force on PF0.
4.2.3.15		With all HMI user input switches released, force output PF0 on. Push and hold switch S2E5, and measure voltage with respect to DC Common on each of the three inputs PE5, PE6, & PE7. Verify that PE5, PE6 and PE7 reflect the BCD 3 (0,1,1). Remove force on PF0.
4.2.3.16		With all HMI user input switches released, force output PF0 on. Push and hold switch S2F6, and measure voltage with respect to DC Common on each of the three inputs PE5, PE6, & PE7. Verify that PE5, PE6 and PE7 reflect the BCD 2 (0,1,0). Remove force on PF0.
4.2.3.17		With all HMI user input switches released, force output PF0 on. Push and hold switch S2G7, and measure voltage with respect to DC Common on each of the three inputs PE5, PE6, & PE7. Verify that PE5, PE6 and PE7 reflect the BCD 1 (0,0,1). Remove force on PF0.
4.2.3.18		With all HMI user input switches released, force output PF0 on. Push and hold switch S2A1, and measure voltage with respect to DC Common on each of the three inputs PE5, PE6, & PE7. Verify that PE5, PE6 and PE7 reflect the BCD 0 (0,0,0). Remove force on PF0.

Table 4.2.3-2: HMI #2 user designation zone inputs test plan procedures

Once the tests contained in 4.2.3-2 have been successfully completed, the same tests will need to be conducted for the HMI #1 and HMI #2 user selection inputs. The acceptance test procedures for the HMI #1 and HMI #2 user selection inputs can be referenced in Table 4.2.3-3 below.

Item #	Passed	Test Description
4.2.3.19		With all HMI user input switches released, force output PE4 on. Push and hold switch S1A, and measure voltage with respect to DC Common on each of the three inputs PE1, PE2, & PE3. Verify that PE1, PE2 and PE3 reflect the BCD 7 (1,1,1). Remove force on PE4.
4.2.3.20		With all HMI user input switches released, force output PE4 on. Push and hold switch S1B, and measure voltage with respect to DC Common on each of the three inputs PE1, PE2, & PE3. Verify that PE1, PE2 and PE3 reflect the BCD 6 (1,1,0). Remove force on PE4.
4.2.3.21		With all HMI user input switches released, force output PE4 on. Push and hold switch S1C, and measure voltage with respect to DC Common on each of the three inputs PE1, PE2, & PE3. Verify that PE1, PE2 and PE3 reflect the BCD 5 (1,0,1). Remove force on PE4.
4.2.3.22		With all HMI user input switches released, force output PE4 on. Push and hold switch S1D, and measure voltage with respect to DC Common on each of the three inputs PE1, PE2, & PE3. Verify that PE1, PE2 and PE3 reflect the BCD 4 (1,0,0). Remove force on PE4.
4.2.3.23		With all HMI user input switches released, force output PE4 on. Push and hold switch S2A, and measure voltage with respect to DC Common on each of the three inputs PE1, PE2, & PE3. Verify that PE1, PE2 and PE3 reflect the BCD 3 (0,1,1). Remove force on PE4.
4.2.3.24		With all HMI user input switches released, force output PE4 on. Push and hold switch S2B, and measure voltage with respect to DC Common on each of the three inputs PE1, PE2, & PE3. Verify that PE1, PE2 and PE3 reflect the BCD 2 (0,1,0). Remove force on PE4.
4.2.3.25		With all HMI user input switches released, force output PE4 on. Push and hold switch S2C, and measure voltage with respect to DC Common on each of the three inputs PE1, PE2, & PE3. Verify that PE1, PE2 and PE3 reflect the BCD 1 (0,0,1). Remove force on PE4.
4.2.3.26		With all HMI user input switches released, force output PE4 on. Push and hold switch S2D, and measure voltage with respect to DC Common on each of the three inputs PE1, PE2, & PE3. Verify that PE1, PE2 and PE3 reflect the BCD 0 (0,0,0). Remove force on PE4.
4.2.3.27		Slowly adjust the HMI # 1 10K Ohm potentiometer to obtain the best light intensity adjustment. Repeat the process for HMI # 2.
4.2.3.28		Reference the ASCII table and print out every combination of characters to verify that all for bits of the upper and lower nibbles are working. The correct ASCII character should be displayed on the appropriate LCD screen. Conduct this test for both LCD screens.
4.2.3.29		Print a character on every cell on every line, resulting in 40 characters populating the screen. Conduct this test for both LCD screens.

Table 4.2.3-3: HMI #1 and HMI #2 user designation zone inputs test plan procedures

4.2.4 Motor and Axes Control

The scope of our motor and motor control acceptance testing shall verify the positioning and placement functionality. This phase of testing will begin with validating the over travel limit switches and the axis home sensors and finish with continuous run time to verify that error building is not a problem with our system. All test steps are clearly defined in Table 4.2.4-1 below.

Item #	Passed	Test Description
4.2.4.1		With the E-Stop switch pushed down verify with a volt meter that 12 VDC is NOT present on all three motor control drive terminals 12VMC.
4.2.4.2		Pull up the E-Stop and verify that 12 VDC is now present on all three motor control drive terminals 12VMC.
4.2.4.3		With the machine powered down move the carriage to a random position near the center point of the table.
4.2.4.4		Power up the machine and verify that the grabber is positioned in the center of cell A1 plus or minus 5 mm with respect to the X axis. If the test fails adjust the home sensor and repeat the test until the proper results can be achieved.
4.2.4.5		Power up the machine and verify that the grabber is positioned in the center of cell A1 plus or minus 5 mm with respect to the Y axis. If the test fails adjust the home sensor and repeat the test until the proper results can be achieved.
4.2.4.6		Verify that the Z axis completes the calibration sequence and parks in the up position on the up sensor plus or minus 5.4 degrees. If the test fails adjust the up sensor and repeat the test until the proper results can be achieved.
4.2.4.7		Hold down the X axis over travel limit switch # 1 and then command the machine to make a move. Verify that no motion occurs.
4.2.4.8		Hold down the X axis over travel limit switch # 2 and then command the machine to make a move. Verify that no motion occurs.
4.2.4.9		Hold down the Y axis over travel limit switch # 1 and then command the machine to make a move. Verify that no motion occurs.
4.2.4.10		Hold down the Y axis over travel limit switch # 2 and then command the machine to make a move. Verify that no motion occurs.
4.2.4.11		Command the machine to move to cell H8 and pick up a chess piece. When the Z axis is in the down position; kill power to the machine and verify that the Z axis has parked in the lowest position plus or minus 5.4 degrees.
4.2.4.12		Power up the machine and verify that it returns back to the A1 home location.
4.2.4.13		Command the machine to go to cell H7 pick up a piece and move it to cell G6. Verify that the machine successfully completes the task.
4.2.4.13		Command the machine to go to cell A2 pick up a piece and move it to cell B2. Verify that the machine successfully completes the task.
4.2.4.14		Command the machine to go to a cell and pick up a piece. While the machine is in motion push the E-Stop switch. Verify that all machine motion stops.

Table 4.2.4-1: Motor and axes control test plan procedures

4.2.5 LED Lighting Matrix

For testing the LED matrix the LED driver, a MAX7219, will be placed into a bread board and the chip will be fitted into the center split (located in the center of the board) to prevent the horizontally connected pin holes in the bread board from bridging the MAX7219's pins. A LED, red in color, will be connected with its anode attached to the SEGG (Segment G) pin and its cathode connected to DIG0 (Digit 0) pin. The GND (Ground) pins will be connected to the rail on the side of the bread board making a grounding bus. A 9.53kΩ resistor (the resister value is specified in the data sheet) will bridge the ISET (Peak segment current) and V+ (+5VDC positive supply) pins. A wire will be put in the same line as the V+ pin and the other end of the wire will connect to two capacitors in parallel; the capacitors are for voltage filtering, with the other end of the capacitors connected to the grounding bus. A wire will be connected to the Din, Load, and CLK pins. The V+ bus will have a wire going to a DC power supply and the grounding bus will have a wire going to a common ground. For a visual of the layout see Figure 4.2.5-1a below. This will be the basic setup for testing the LED matrix.

For testing the chip operations a microcontroller development board will be used. The development board will have the same microcontroller as the one that will be used for the overall project: the Atmel Xmega 128A1U microcontroller. The wires used for the serial connection will be connected to the development board for serial output to simulate the signals that would switch the LED off and on. After the LEDs light up as desired, the cathode of each individual LED would continue to be connected to the DIG0 pin, the anode of each individual LED will be connected from the SEGG pin to the SEGF (Segment F) pin, and then the development board will be set to turn the LEDs on. This process will be repeated so the anode of each LED would connect to each of the SEG pins; once the anode of the LED has been tested on each of the SEG pins the cathode of the LED will be connected to the DIG1(Digit 1) pin. The testing process overall basically becomes testing the first Digit pin with each of the Segment pins and once all the Segment pins have been tested with the first Digit pin, the next Digit pin will be used and the process of going through the Segment pins and will repeat again; this process will continue to iterate until all the Segment and Digit pins have been tested exhaustively.

The next step in testing the serial connections and chip operations will be to set up several other bread boards with the same configuration as the simulation in the paragraph above so that, eventually, there will be a setup for each LED color: red, blue, and green. The chips will be serially connected as shown below in Figure 4.2.5-1b; for a closer look at the schematic view of each chip as well as how the serial connections attach to each chip see Figure 4.2.5-3, Figure 4.2.5-4, and Figure 4.2.5-5 below. The procedure of testing the chips will be repeated from the previous paragraph with the exception that all three chips will be working together via the serial connection; the serial connection will be simulated again by the development board used in the previous paragraph, testing the

LED's anodes in each SEG pin slot and the cathodes in each DIG slot. Once every LED combination works the prototyping can commence.

The prototype board with all three LED drivers, shown in the schematic view below in Figure 4.2.5-2, will be attached with male connectors to connect the sources and sinks of each chip to the LEDs. Each LED will be on a separate individual board; the schematic layout of those individual boards can be seen below in Figure 4.2.5-6. The LED boards will have wires connecting to a female plug that can, in turn, plug into the male end located on the LED controller board; this will lead to multiple boards finally connecting together each color LED (red, blue and green) and each LED will have its own bus for its personal source and sink connections. This will lead to approximately 64 LED boards total, one for each square of the chessboard.

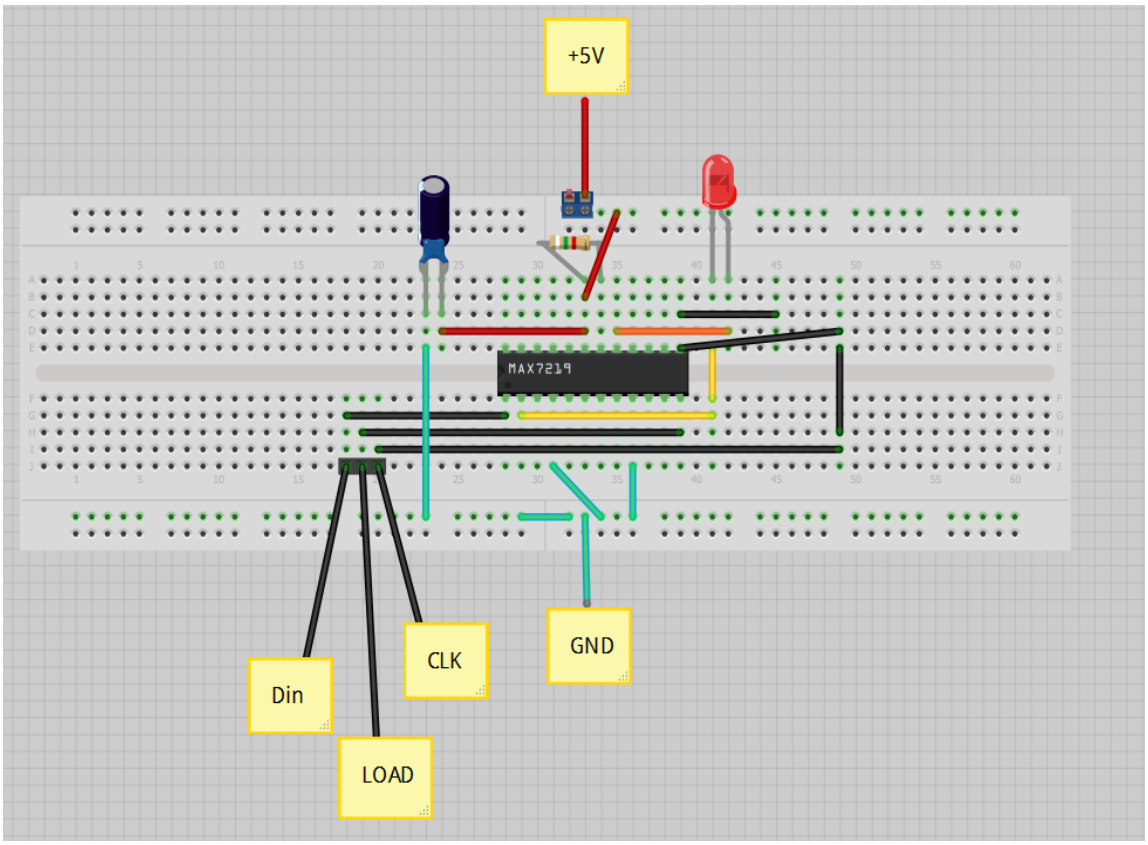

Figure 4.2.5-1a: The layout for the bread board incorporating the MAX7219 chip into a single LED design. This figure was created using Fritzing.

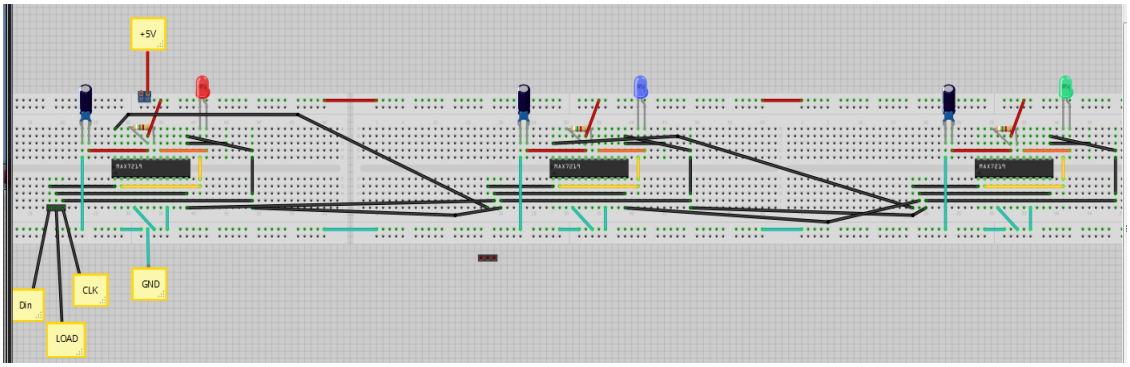

Figure 4.2.5-1b: The layout for the bread board incorporating the MAX7219 chip into all three LED designs; each board is connected serially to the one next to it. This figure was created using Fritzing.

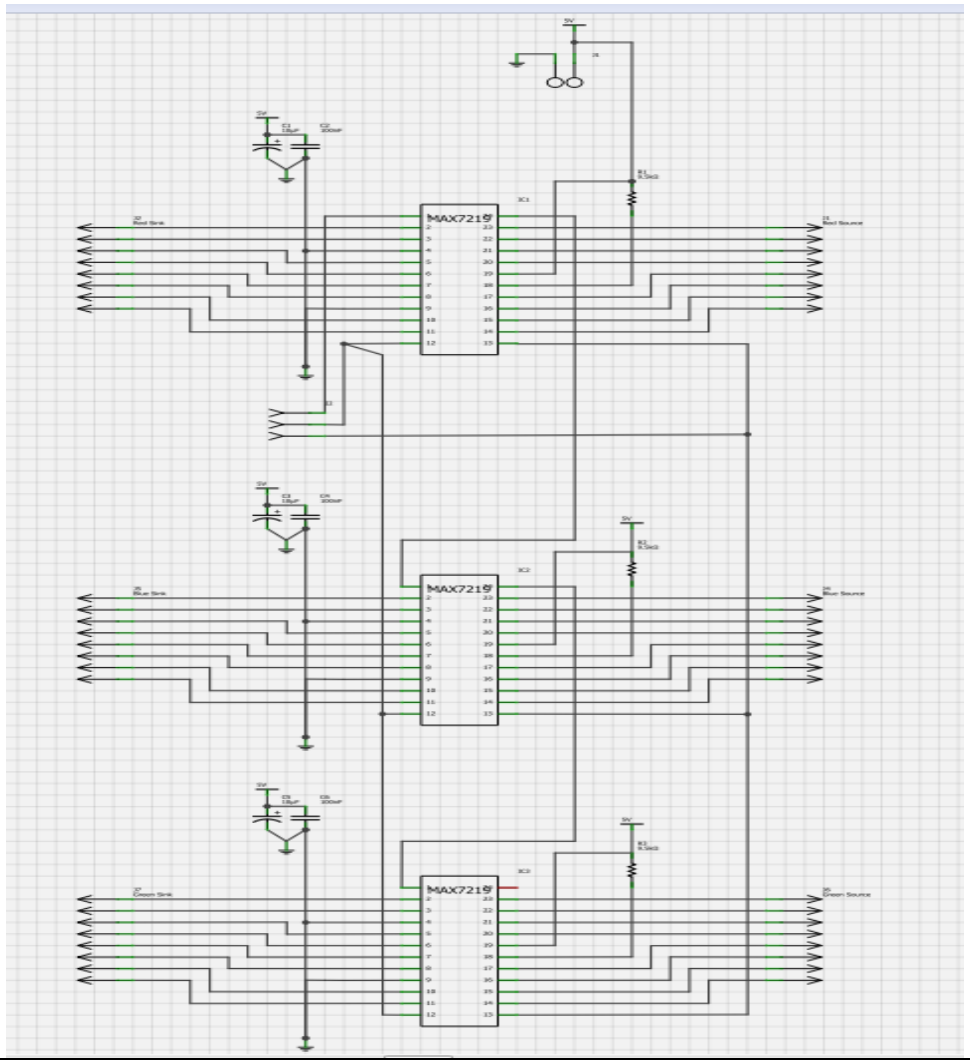

Figure 4.2.5-2: The prototype board with all three LED drivers connected as described above. This figure was created using Fritzing.

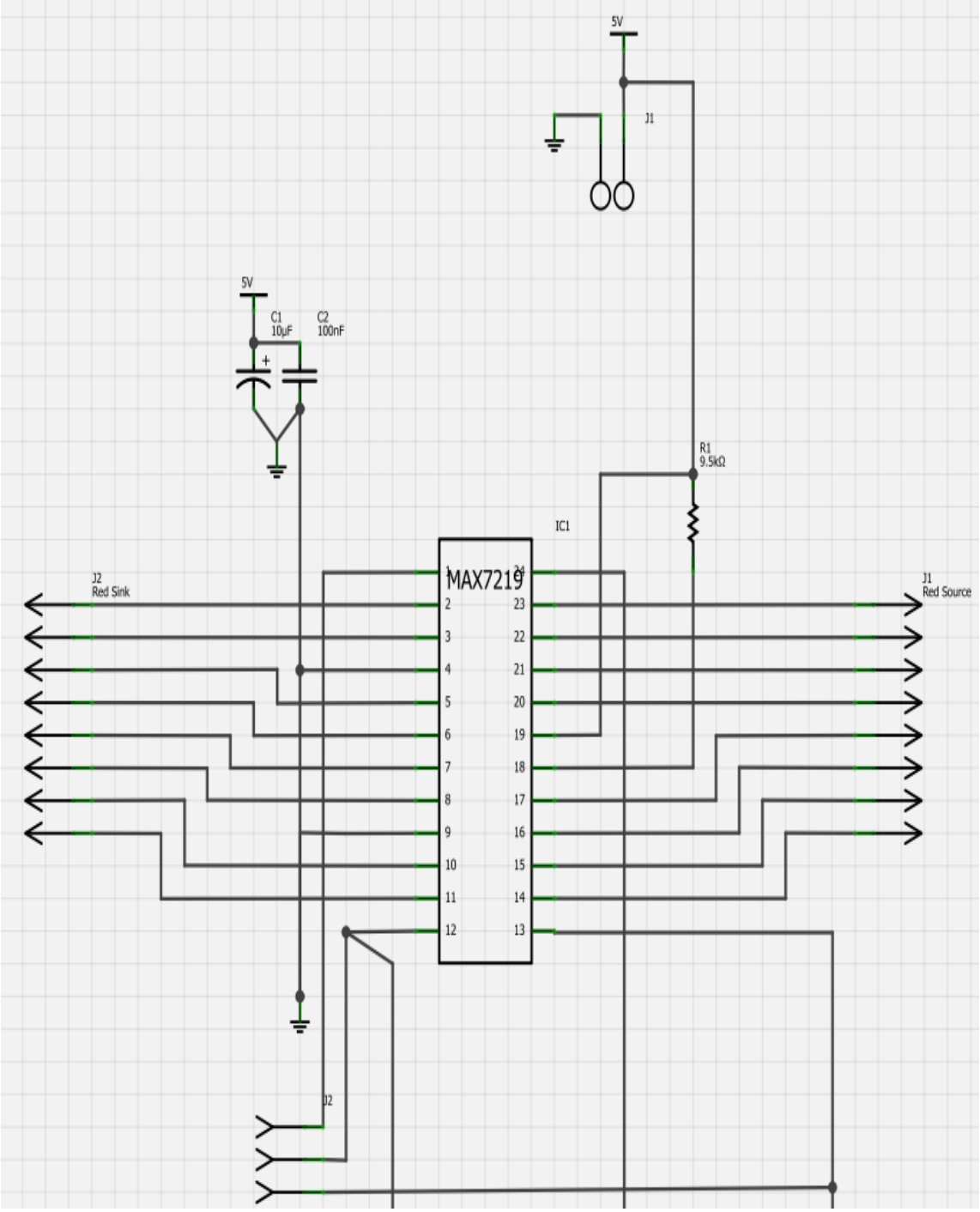

Figure 4.2.5-3: The prototype board showing a zoomed in version of the first LED driver. This figure was created using Fritzing.

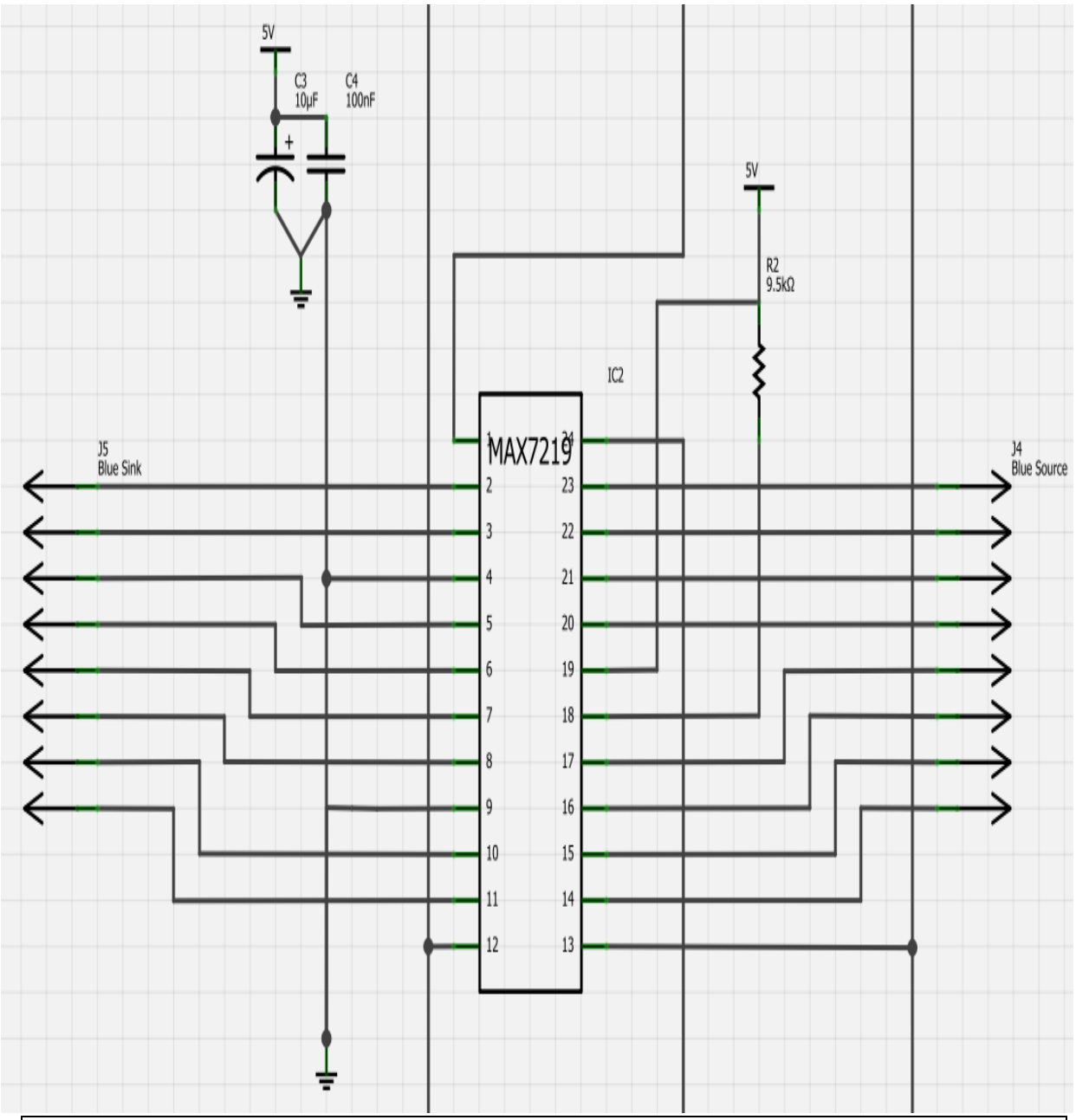

Figure 4.2.5-4: The prototype board showing a zoomed in version of the second LED driver. This figure was created using Fritzing.

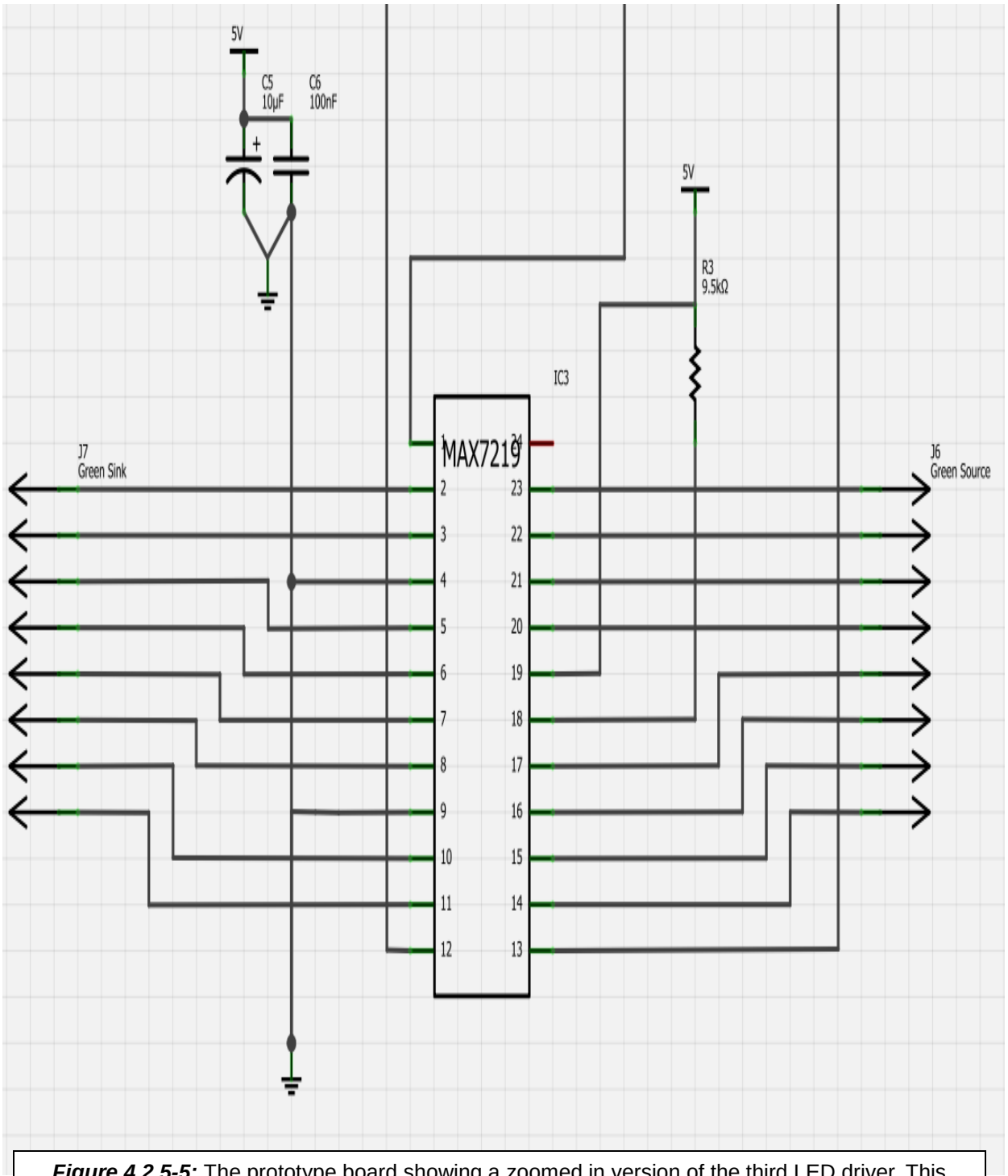

Figure 4.2.5-5: The prototype board showing a zoomed in version of the third LED driver. This figure was created using Fritzing.

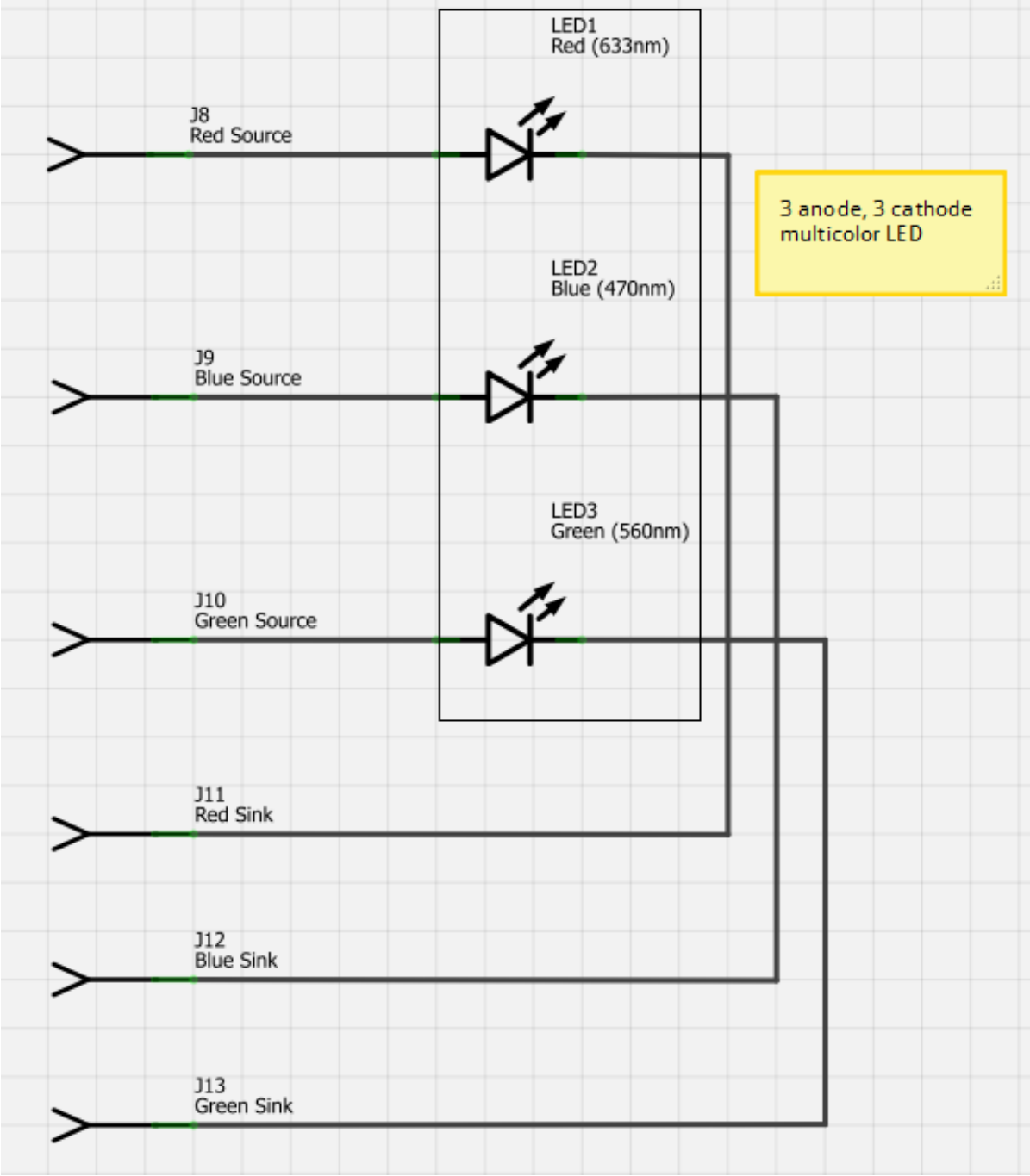

Figure 4.2.5-6: The schematic view of each LED on a separate individual board depicting each source and sink. This figure was created using Fritzing.

4.2.6 Hall Effect Sensor Grid

Since the Hall Effect Sensor Grid was not implemented an acceptance test was not required for the system.

4.2.7 Picker/Claw

The scope of our grabber/claw acceptance testing shall verify the opening, closing, and holding functionality. In this section the claw will be moved downward while open so clearance with adjoining cells can be verified to make sure no interference or displacement of adjacent pieces occur. The clamping force will be checked to make sure that the grabber can hold the piece without worrying about slippage or breakage. If the servomotor stalls the motor current will increase substantially to around the stall current value listed in the spec sheet. Testing must insure that the motor never reaches the stall condition. The necessary test steps are shown in Table 4.2.7-1 below.

Item #	Passed	Test Description
4.2.7.1		Command the machine to go to cell A2 pick up a piece and move it to cell B2. While the X axis is in motion kill power to the 12 volt buss. Verify that the grabber continues to hold the chess piece without dropping it. Reset the fault and verify the grabber moves to the home position.
4.2.7.2		When the machine deposits any given piece, verify that the piece does not slip or slide downward resulting in an un-controlled placement.
4.2.7.3		Pick any cell surrounded by four adjacent cells; populate the cell and the four adjacent cells. Command the grabber to pick up the piece, and verify that the machine successfully completes the move without contacting the other four pieces.
4.2.7.4		With the Grabber holding any given chess piece in the closed position measure the servo motor current, and verify that the motor is NOT over clamping the piece. Repeat the test for each type of chess piece.

Table 4.2.7-1: Grabber/claw test plan procedures

4.2.8 Chess Module

The chess module is very tightly coupled, so we can't test the functions individually. We will create a virtual chess game driven by the virtual main that inputs moves and calls the appropriate methods to simulate a very short chess opener. This opener is designed to make sure that when the player attempts to take an AI piece, the chessboard is updated properly. The test, described in Figure 4.2.8-1 below, involves a set of three player moves: moving the knight three times directly down the board to the AI player's pawn row. In this case, we know that a piece will be taken. We can test for the result in which the AI responded to player turn #2 by taking the knight with the pawn, or for the case in which the player gets to move turn #3 and take the enemy's pawn. In this case, we will make sure that the AI retaliates by taking the player's knight. In this way, we will make sure that the functions Think, DoMove, and ReadMove work as we

think they will. We will know that the chessboard is able to be manipulated by player moves, pieces can be taken, and that the AI is performing according to the heuristics we have set for it. We will save the test suite and refer to it later when we are implementing player versus player and AI versus AI modes. In those cases we may need to design separate test suites. The main thing to note is that the test results will be “by inspection”, such that the programmer must be watching the memory at each “step” of the game in the AVR simulator.

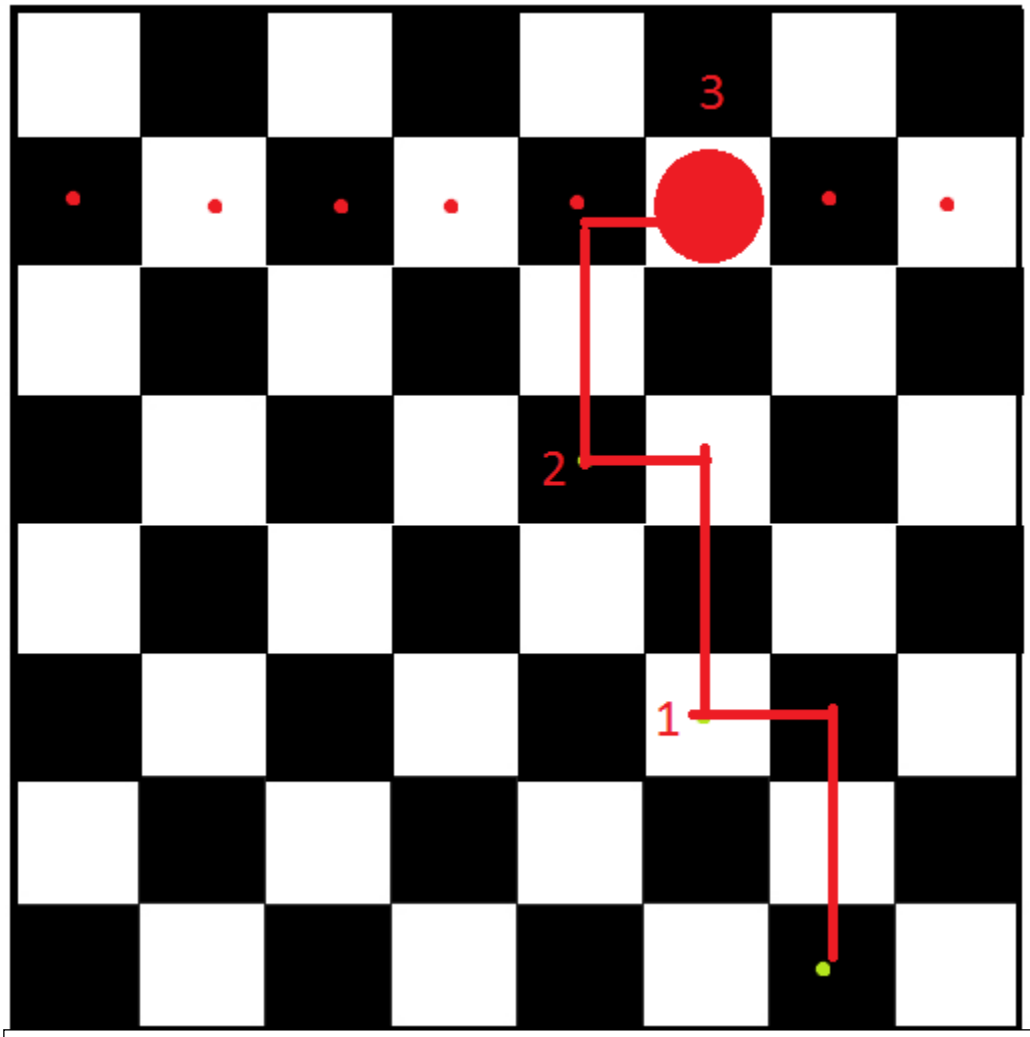

Figure 4.2.8-1: “Knight rush” test suite for chess engine

4.2.9 Main Module for AI

We will test our main module in a simulated environment as well because it is not dependent on I/O functionality. We will create a test suite simulating human input from the I/O interface as well as feedback from the motor controller and chess engine. We will ensure that our main module processes the input correctly from stub methods that represent the I/O controller. We will test that our main process

flows correctly when receiving game-over signals from the chess engine that we correctly send messages to the audio controller. Then we will test integration with the chess engine by passing real messages between the main module and the chess engine module. The main module will generate the same play messages that were generated for our test suite. The tests for the main module are detailed in Table 4.2.9-1 below.

Test	Input Specification	Required Output
Begin Game	On button message from I/O controller	1. I/O system initialization call
		2. LCD splash text call
		3. Process blocked, waiting for button press from I/O
Valid player turn	Valid move from I/O module given current state of game board	1. Positive move result from chess engine
		2. Activation of pickUpPiece and movePieceToLocation functions in motor controller
Invalid player turn	Invalid move from I/O module given current state of game board	1. Negative move result from chess engine
		2. Message to LCD driver to redo turn
AI turn	AI move from chess engine	1. Message to motor controller to move piece to correct location
Game over	Game over message from chess engine	1. Game over text to LCD driver
		2. Game over message to audio driver
		3. Display splash text to LCD
		4. Process blocked, waiting for button press from I/O
Motor failure	Motor failure message after attempting to move a piece	1. Send message to LCD driver regarding error
		2. Send message to audio driver regarding error
		3. Retrace to correct location in main process
		4. Process blocked, waiting for button press from I/O

Table 4.2.9-1: Test plan for main module

5 Section 5 Budget and Time Management

5.1 Pricing Breakdown

The Bill of Materials (BOM) is our official record of parts used on our project. In industry the Bill of Materials provided by Engineers is often referenced by maintenance teams and their purchasing partners. Sometimes the technicians just need more information on a given part so they will reference the assembly drawing, locate the part, cross it back to the BOM, and now they can pull up the correct spec sheet on-line. Other times purchasers are asked to buy a replacement part and the BOM is their verification document that they have ordered the correct part. Normally only Engineering Departments can make changes to drawings and BOMs. This policy helps to keep systems compliant with good engineering practices. Often times parts look the same but the manufacturer offers multiple configurations all with different part numbers. If a technician installs the wrong part unsafe and unpredictable equipment behavior can result. The worst case is when the action results in a latent failure mode being introduced into the system. The wrong part may go undetected for years until it has been found as a root cause by an accident investigation team.

Shown below in Table 5.1-1 are all of the parts, where a piece was used (if used for a specific place), how much each piece cost, and the number of units ordered at each time. Some parts had to be ordered several times due to infant mortality and some redesign as needed before the final presentation of the Interactive Automated Chess Set. The entire project, not including donations and supplies the group already had lying around, cost a grand total of $544.80.

Idem #	Where is it used?	# Bought	Price Per Unit	Shipping/Tax	Total Price
SF-ROB-09238	Claw	2	$ 14.95	$ 7.99	$ 37.89
HT-33322S	Claw	1	$ 9.99	$ -	$ 9.99
A1M12MYHF102000	Claw	1	$ 41.36	$ 9.52	$ 50.88
A1A12MHR101000	Claw	1	$ 15.23	$ 9.52	$ 24.75
S10Z10M020T0505	Claw	2	$ 16.75	$ 9.52	$ 43.02
HEF4514B	Hall Effect Sensors	2	$ 0.48		$ 0.97
74HC151	Hall Effect Sensors	2	$ 0.16		$ 0.33
OH090U	Hall Effect Sensors	3	$ 1.99		$ 5.97
AVR-ISP-MK2		1	$ 38.04	$ -	$ 38.04
	HMI	1	$ 24.95	$ 5.21	$ 30.16
UVR1H100MDD		12	$ 0.04	$ 2.25	$ 2.72
NCD103M100Z5UF		12	$ 0.08	$ 2.25	$ 3.15
MAX7219CNG+		6	$ 9.62	$ 7.86	$ 65.58
GP50-9531-FTW		10	$ 0.005	$ 7.86	$ 7.91
VAOS-5050RGB-W1		1	$ 2.03	$ 7.86	$ 9.89
TL1105AF250Q		12	$ 0.2251	$ 2.25	$ 4.95
CSC10A014K70GEK		1	$ 0.17	$ 2.25	$ 2.42
M74HC148B1R		4	$ 0.57	5.99	$ 8.27
		1	$ 12.30	$ -	$ 12.30
	Chess set	1	$ 19.99	$ 8.00	$ 27.99
	Port A-I	10	$ 1.35	$ 3.64	$ 17.14
445-4730-ND	C24	2	$ 0.29	$ 2.84	$ 3.42
	JP1	2	$ 0.70	$ 1.95	$ 3.35
	HMI	1	$ 16.24	$ 1.99	$ 18.23
	Under board	64	$ 0.45	$ 1.49	$ 30.29
	Under board	3	$ 2.95	$ 1.49	$ 10.34
	Under board	3	$ 2.95	$ 1.49	$ 10.34
763-0420D3ZFLGBW-V3 NHD-0420D3Z-FL-GBW-V3 4 x 20 STN-GRAY	HMI	1	$ 24.90	$ 37.99	$ 62.89
PST 5273	general	1	$ 1.50	$ 0.10	$ 1.60

5.2 Sponsorship and Guidance

Our team secured sponsorship from Igus Inc. Igus donated around $200 worth of product though their YES program. They donated all of our linear slide rails and carriages for all three axes, and donated wire management safety chain

products. We were able to obtain electronics from Allied Electronics. SPD gave us a 15% discount on our gear order. We learned to ask vendors for their support, and we were pleasantly surprised. We received Mechanical Engineering advice and design review from Alan Skaggs MME from Walt Disney World Co. Our design documents were also being reviewed by Carl Stover MEE GE Drive Systems retired.

5.3 Time Allocation

For the entirety of Senior Design I each member kept track of the time they devoted to the project including the date, time, whether the main focus of that time was the project or if it was worked on intermittently during that time (Solid or Int), and any group members they worked with. Those logs can be found in Tables 5.3-1 through 5.3-4 below.

Name	***Date***	***Work Done***	***Time Range for Work***	***Total Time (Hours)***	***Solid or Intermittant***	***Group Member Worked With***
Sam	5/19/2012	Setting up contact info and group calendar	10:30-11:30	1.00	Solid	None
All	5/21/2012	Brainstorming Ideas for project	17:30-19:45	2.25	Solid	All
All	5/22/2012	Brainstorming Ideas for project	11:15-12:00	0.75	Solid	All
All	5/24/2012	Mini Metting for Group ID Doc	10:50-11:40	0.83	Solid	All
Sam	5/24/2012	Initial Project and Group ID Document	11:50-12:40	0.83	Solid	All
Sam	5/24/2012	Initial Project and Group ID Document	14:15-16:20	2.08	Solid	None
Nick	5/24/2012	Research	14:00-16:00	2.00	Solid	None
Nick	5/24/2012	Research / Initial Project Doc	17:00-19:00	2.00	Solid	None
Paul	5/26/2012	AutoCAD Drawings Block Diagram	8:00-13:00	5.00	Solid	None
Paul	5/26/2012	Writing Goals and Desing Conciderations	20:00-22:00	2.00	Solid	None
All	5/29/2012	Assigning Tasks in Block Diagram	9:00-10:00	1.00	Solid	All
Brett	5/29/2012	Research Parts (Trip to Skycraft)	13:00-14:00	1.00	Solid	Paul, Sam
Brett	5/30/2012	LED Research and Block Diagram	14:00-18:00	4.00	Solid	None
All	5/30/2012	Finishing Touches on Initial and Group ID Doc	18:00-19:20	1.33	Solid	All
Paul	6/3/2012	Research Igus, Jamesco, Motor Cont & Mech	11:00-18:00	7.00	Int	None
Brett	6/3/2012	Research hall effect sensores/magnets	17:30-19:30	2.00	Int	None

Table 5.3-1: Time allocation 5/19/2012 – 6/3/2012

Name	***Date***	***Work Done***	***Time Range for Work***	***Total Time (Hours)***	***Solid or Intermittant***	***Group Member Worked With***
Brett	6/5/2012	Reserching LED Grid	14:10-16:20	2.17	Int	None
Paul	6/5/2012	Call Igus, Install AutoCAD	14:00-21:00	4.00	Int	None
Paul	6/6/2012	Igus and Vex	18:00-21:00	2.00	Int	None
Brett	6/6/2012	LED Grid Driver	21:08-21.37	0.48	Solid	None
Sam	6/7/2012	Table of Contents	13:00-2:15	1.25	Solid	None
Brett	6/8/2012	Meeting	18:00-18:20	0.33	Solid	Nick
Brett	6/7/2012	Meeting	18:20-19:15	0.92	Solid	Nick, Sam
All	6/8/2012	Meeting	19:15-20:30	1.25	Solid	All
Brett	6/9/2012	Hall effect sensors reserch	11:00-14:00	3.00	Int	None
Nick	6/10/2012	Microcontroller research	11:00-15:00	4.00	Int	None
All	6/18/2012	Meeting	18:00-19:30	1.50	Solid	All
Paul	6/18/2012	Drawings	20:00-21:00	1.00	Solid	None
Paul	6/19/2012	Drawings and Research	13:00-21:00	7.00	Solid	None
Brett	6/20/2012	Research	13:00-15:30	2.50	Int	None
Nick	6/24/2012	MCU Research/Software sections of Paper	11:00-14:30	2.50	Solid	None
Paul	6/24/2012	Drawings, Research, Writing	7:00-21:00	12.00	Int	None
Sam	6/25/2012	Writing (Section 1)/Formatting Paper	10:00-12:00	2.00	Solid	None
All	6/26/2012	Meeting	10:00-11:45	1.75	Solid	All
Brett	6/26/2012	Writing	12:15-13:45	1.50	Solid	None
Brett	6/27/2012	Writing	16:45-19:00	2.25	Int	None
Paul	6/27/2012	Writing	17:00-21:00	3.00	Int	None
Paul	6/29/2012	Writing	17:00-21:00	4.00	Solid	None
Brett	7/1/2012	Writing	13:00-16:10	3.17	Solid	None
Paul	7/1/2012	Writing	9:00-21:00	10.00	Int	None
Nick	7/2/2012	Writing	7:30-9:30	2.00	Solid	None
Brett	7/2/2012	Writing	12:30-16:30	4.00	Solid	None
Brett	7/2/2012	Writing	17:10-18:07	0.95	Solid	None
Paul	7/2/2012	Writing	17:00-21:30	2.00	Int	None
Brett	7/4/2012	Writing	14:45-16:00	1.25	Solid	None
Nick	7/4/2012	Writing, Researching Audio	12:00-4:00	4.00	Solid	None
Paul	7/4/2012	Writing Building Motor Cont Test Unit	10:00-19:0	6.00	Int	None
Brett	7/6/2012	Reserching USB/Scematic setup for report	12:00-15:18	1.30	Int	None
Brett	7/8/2012	Writing	13:07-15:05	1.97	Solid	None
Paul	7/8/2012	Writing	8:00-21:00	11.00	Int	None
Nick	7/8/2012	Writing and Designing	17:00-19:30	2.50	Solid	None

Table 5.3-2: Time allocation 6/5/2012 – 7/8/2012

Name	***Date***	***Work Done***	***Time Range for Work***	***Total Time (Hours)***	***Solid or Intermittant***	***Group Member Worked With***
Sam	7/9/2012	Research and Writing	14:40-16:00	1.33	Solid	None
Paul	7/9/2012	Writing and Drawings	17:00-21:00	4.00	Int	None
Brett	7/10/2012	Meeting	9:00-9:40	0.67	Solid	Sam, Paul
All	7/10/2012	Meeting	10:00-10:30	0.50	Solid	All
Sam	7/10/2012	Research and Writing	10:30-11:00	0.50	Solid	None
Brett	7/10/2012	Research and Writing	13:00-14:00	1.00	Solid	None
Paul	7/10/2012	Research and Writing	12:00-21:00	9.00	Int	None
Brett	7/11/2012	Research and Writing	16:00-18:30	2.50	Solid	None
Paul	7/11/2012	Writing and Drawaings	16:00-21:00	5.00	Int	None
Sam	7/11/2012	Writing and First Submission for Dr. Richie	7:00-8:15	1.25	Solid	None
Brett	7/12/2012	Power Supply R&D	16:00-20:00	4.00	Solid	None
Sam	7/12/2012	Writing, Streamlining, Combining and Formatting	20:30-22:30	2.00	Solid	None
Brett	7/12/2012	Power Supply R&D	22:38-00:55	2.28	Solid	None
Sam	7/13/2012	Writing, Streamlining, Combining and Formatting	7:30-8:00	0.50	Solid	None
Sam	7/13/2012	Writing, Streamlining, Combining and Formatting	8:15-10:30	2.25	Solid	None
Brett	7/13/2012	Power Supply R&D, Writing	12:00-16:00	4.00	Int	None
Paul	7/13/2012	Writing	20:00-21:00	1.00	Solid	None
Brett	7/15/2012	Research and Writing	13:20-17:00	3.67	Solid	None
Paul	7/15/2012	BOM	10:00-18:00	8.00	Int	None
Brett	7/15/2012	Research and Writing	18:50-20:22	1.53	Solid	None
Sam	7/15/2012	Fresh Set of Eyes	22:00-23:00	1.00	Int	None
Sam	7/15/2012	Fresh Set of Eyes	23:00-23:50	0.83	Solid	None
Sam	7/16/2012	Fresh Set of Eyes	9:30-10:40	1.17	Solid	None
Sam	7/16/2012	Fresh Set of Eyes	12:00-12:30	0.50	Solid	None
Sam	7/16/2012	Fresh Set of Eyes	13:15-15:50	2.58	Solid	None
Brett	7/16/2012	Reserch	13:00-15:00	2.00	Int	None
Brett	7/16/2012	Writing	16:00-16:55	0.92	Solid	None
All	7/16/2012	Meeting	18:00-19:00	1.00	Solid	All
Sam	7/17/2012	Writning	10:10-11:30	1.33	Solid	None
Brett	7/17/2012	Research and Writing	15:59-17:16	1.28	Solid	None
Paul	7/17/2012	Build MC, Ordering parts, Shopping for Materials	10:00-18:00	5.00	Int	None
Brett	7/18/2012	Research and Writing	12:50-14:50	2.00	Solid	None
Sam	7/18/2012	Writing	15:00-16:30	1.50	Solid	None
Brett	7/18/2012	Research and Writing	20:25-21:00	0.83	Solid	None
Paul	7/18/2012	Testing Motor Control, Igus Request	17:00-21:00	3.00	Int	None
Sam	7/19/2012	Writing	13:15-14:30	1.50	Solid	None
Brett	7/19/2012	Design, Prototype for paper	13:30-15:20	1.83	Solid	None
Sam	7/19/2012	Writing	16:00-16:45	0.75	Solid	None

Table 5.3-3: Time allocation 7/9/2012 – 7/19/2012

Name	***Date***	***Work Done***	***Time Range for Work***	***Total Time (Hours)***	***Solid or Intermittant***	***Group Member Worked With***
Nick	7/20/2012	Writing	13:40-14:40	1.00	Solid	None
Brett	7/20/2012	Design, Prototype for paper	16:25-18:15	1.83	Solid	None
Brett	7/21/2012	Writing, Design, Prototype for paper	15:26-19:00	3.57	Solid	None
Brett	7/22/2012	Writing, Design, Prototype for paper	13:44-16:13	2.48	Solid	None
Brett	7/23/2012	Writing, Design, Prototype for paper	14:50-16:06	1.27	Solid	None
Paul	7/24/2012	Motor Testing Shopping Cutting Wood	10:00-21:00	11.00	Int	None
Sam	7/25/2012	Brett's Writing Document	15:00-16:15	1.25	Solid	None
Sam	7/26/2012	Brett's Writing Document	0:00-0:45	0.75	Int	None
Sam	7/26/2012	Writing	13:00-14:50	1.83	Int	None
Brett	7/26/2012	Writing, a bit of Research	14:48-15:48	1.00	Solid	None
Sam	7/26/2012	Writing	21:45-22:45	1.00	Solid	None
Sam	7/27/2012	Formatting and filling to final document	11:30-12:00	0.50	Solid	None
Sam	7/27/2012	Formatting and filling to final document	12:30-14:15	1.75	Solid	None
Sam	7/27/2012	Formatting and filling to final document	16:00-17:50	1.83	Solid	None
Sam	7/27/2012	Formatting and filling to final document	21:50-0:20	2.50	Solid	None
Sam	7/28/2012	Formatting and filling to final document	12:15-14:10	1.92	Solid	None
Sam	7/28/2012	Formatting and filling to final document	14:30-15:00	0.50	Solid	None
Brett	7/28/2012	Editing doc for final doc	15:20-16:40	1.33	Solid	None
Sam	7/28/2012	Formatting and filling to final document	17:20-18:00	0.67	Solid	None
Paul	7/28/2012	Writing	20:00-22:30	2.00	Int	None
Sam	7/28/2012	Formatting and filling to final document	23:20-1:00	1.67	Solid	None

Table 5.3-4: Time allocation 7/20/2012 – 7/28/2012

If all of the hours are tallied above then the total number of hours devoted to the Senior Design class, not including lecture time, can be found in Table 5.3-5 below where Time (.xx) refers to the weight the intermittent time spent was given. This spread shows that a bare minimum of 173.19 hours were focused solely on the research and design of the Interactive Automated Chess Set with a maximum equal to 271.29 hours.

Total (.25)	***Total (.50)***	***Total (.75)***	***Total Hours***
173.19	205.89	238.59	271.29

Table 5.3-5: Total hours devoted to the design of the Interactive Automated Chess Set thus far

Due to the nature of Senior Design II and how much time, including spare moments in between classes and other obligations, was needed in order to get the Interactive Automated Chess Set up and running, it was almost impossible to keep track of exactly how much time was spent working on the project this semester but we estimate that it is the same or greater than the time spent during Senior Design I.

6 Section 6 Appendix

6.1 Acronyms

- A/R: amount used as required
- ATP: Acceptance Test Plan
- ATR: Acceptance Test Results
- BGA: Ball Gate Array
- BOM: bill of materials
- CLK: clock
- CNC: Computer Numerical Control
- DeMux: Demultiplexer
- DFF: D Flip Flop
- DIG0: Digit 0
- Din: Data input
- Dout: Data output
- E': Enable' (not)
- EL: Enable latch
- FMEA: Failure modes & effect analysis
- GUI: General user interface
- HMI: Human Machine Interface
- I/O: Input/Output
- ISET: Peak segment current
- LCD: Liquid Crystal Display
- LED: Light-Emitting Diode
- MC: Microcontroller
- MES: Manufacturing Execution Systems
- Mux: Multiplexer
- MVC: model view controller
- OEM: Original Equipment Manufacturers

- PLC: Programming Logic Controllers
- PPR: Pulses per Revolution
- PWM: Pulse Width Modulation
- RGB: Red Green Blue
- SCADA: Supervisory Control & Data Acquisition
- SDP/SI: Sprocket Drive Products & Sterling Instruments Company
- SEGG: Segment G
- TI: Texas Instruments
- TSCO: Tom Kerrigan's simple chess program
- UCI: Universal chess interface
- VFD: Variable Frequency Drive control
- YES: Young Engineers Program

6.2 References

1. ("Anaheim Automation")
   "HMI Guide." *Anaheim Automation*. Anaheim Automation, Inc, 2011. Web. 09 Jul 2012. <http://www.anaheimautomation.com/manuals/forms/hmi-guide.php>.
2. ("A small HMI with a big attitude" 1-4)
   "Magelis STO Touch screen graphic terminal, configured by Vijeo Designer."*A small HMI with a big attitude*. Jul 2011: 1-4. Web. 12 Jul. 2012. <http://static.schneider-electric.us/docs/Automation Products/HMI and IPC Hardware/Magelis Operator Terminals--Type STU and STO/8000BR1159_MagelisSTO.pdf>.
3. ("Series FT" 1)
   "Digital Touch Screen." *Series FT*. n.d. 1. Web. 12 Jul. 2012.
4. ("igus: plastics for longer life")
   "Plastic Bushings, Linear Guides, Cable Carriers, Continuous-Flex Cables." *igus: plastics for longer life*. igus, n.d. Web. 12 Jul 2012. <http://www.igus.com/default.asp?c=us&L=en>.
5. (patrickmccabe)
   patrickmccabe, . "Chess Robot & MRL." *Let's Make Robots*. Chillout Zone, n.d. Web. <http://letsmakerobots.com/node/26979>.
6. [1]
   http://www.chesshouse.com/3_3_4_Quality_Club_Special_Chess_Pieces_p/e104.htm
7. Fritzing website: http://fritzing.org/

6.3 Diagrams and Other Supplemental Material