Spring 2016

Voice Activated Chess Set

William Weigand
wdw10@zips.uakron.edu

Alysha Jansto
amj61@zips.uakron.edu

Kerim Bojadzija
kb69@zips.uakron.edu

Mitchell Hall
meh56@zips.uakron.edu

Voice-Activated Chess Set

Final Design Report

Design Group C

Alysha Jansto
Kerim Bojadzija
Mitchell Hall
William Weigand

Advisor: Dr. Elbuluk

April 20, 2016

Table of Contents

Team Members Contribution

Alysha Jansto [AJ]: Electrical Engineering student and the Project Leader for the design group. My design roles are the voice activation module hardware, microcontroller hardware, power supply, hardware structure of the chess set, and 3-D models of the chess set. As the project leader my roles are to organized team meetings, weekly to-do lists, project schedule, parts list, and revise documents.

Kerim Bojadzija [KB]: Computer engineering student and the software manager for the design group. Project focus will be writing and debugging the programs for the microcontroller and the voice recognition device.

Mitchell Hall [MH]: Electrical Engineering student and Archivist for the design group. Design responsibilities included; schematic layout and feedback system. As group Archivist I turned in deliverables, backed up group files, kept track of paperwork in a three ring binder and designed the poster.

William Weigand [WW]: Electrical Engineering student and Hardware Manager for the design group. Design responsibilities include making the mechanical design of the positioning subsystem, stepper motors and drivers configurations, and small circuitry.

Abstract

Chess is a game enjoyed by people of all ages and physical abilities. The Voice-Activated Chess Set was designed to allow persons who cannot use their arms, due to either a disability or if they are preoccupied with something else, to play a game of chess without assistance after initial setup. The chess game was intended to be solely controlled by voice recognition using set commands. After much testing of the voice recognition chip, it was established that the microphone could not stay always on and a dial pad was implemented to control it. A position system repositions a solenoid which is set underneath of the chessboard. The chess pieces are made of plastic and have a nail and bolt inside of them to be able to be moved by a normally closed solenoid with a magnet on the shaft. The position system moves chess pieces according to the player's voice command or input into the dial pad, provided that they follow the rules of chess. With a better microphone or voice recognition chip, this design lets individuals who are handicapped, in the sense that they cannot use their hands, to enjoy a challenging and fun game.

[KB, AJ]

1. PROBLEM STATEMENT

1.1 Need

In the U.S. there are about 21.1 million people who are not capable of performing basic physical activities [1]. Within this number are people who cannot use their arms, but the estimate, unfortunately, does not include soldiers who lost their limbs, people with terrible arthritis in their hands, or individuals with hand tremors. There are a large amount of people just in the U.S that have to depend on others to perform day to day tasks. The Voice-Activated Chess Set will let the person be able to depend on themselves to play a fun and challenging game of chess.

[AJ]

1.2 Objective

The objective of the chess board design is to be able to play a fast responding and long lasting game of chess solely using voice controlled commands. The chess board will be a two player game that has a microphone for each player. Once the pieces have been set up initially, the game will be completely hands free. This will allow the players to enjoy a game of chess without depending on their arms.

[AJ]

1.3 Background

There are many different chess board designs. There is one design in particular that is similar to the group's design which is the Magic Chess Set design. The Magic Chess Set has the same objective as the group which is a chess set with self-moving and voice activated chess pieces. The Magic Chess Set design has two modes of operations which is a player against the computer or player against a second player [2]. The group decided to only have one mode of operation which is to have the chess set be a two player game. The Magic Chess Set used a chess engine and the group decided against using a chess engine. Also, the Magic Chess Set board used a microATX motherboard to control the voice commands and chess engine, which would go over the group's budget [2].

The group wants to use more chess commands than the Magic Chess Set used. The Magic Chess Set did not incorporate chess movements like castling, pawn promotion, and en passant. The group would like to keep the chess game as close to the traditional rules, so castling, pawn promotion, and en passant will be used. The group wants to use voice commands that are more fluent like "A-3 to A-4"oppose to the Magic Chess board command of "A3A4" The Magic Chess Set uses a combination of two stepper motors to make an x-y table to relocate the chess pieces. Just like the Magic Chess Set, the group decided to use a combination of two stepper motors to make an x-y table. This design of the x-y table is the most efficient way to move chess pieces from underneath of the chess board [2].

There is another similar project to the group's which is an Arduino Powered Chess Playing Robot. The design uses an Arduino to power the chess playing robot that consists of an x-y table underneath of the table [3]. The chess playing robot moves in the x-y direction, but the x-direction has a servo attach to lift the magnet on and off of the board. The group's design is to have the electromagnet always be attached to underneath of the board but not supply current to the electromagnet when releasing the piece. The Arduino Powered Chess Playing Robot's board uses reed switches to allow the Arduino to know the location of the pieces. In the group's design the piece location will primarily be remembered by the program. The Arduino Powered Chess Playing Robot's design doesn't have a location for eliminated pieces and have to be removed by a person [3]. The group's design will have an area named the graveyard which will put the eliminated pieces off to the side of the board in a desired location.

[AJ]

1.4 Objective Tree

The group went through and evaluated key components of the design in Figure (1). In Table (1) the value of the row versus column is listed. The grading scale was 1=equal, 3=moderate, 5=strong, 7=very strong, and 9=extreme. Table (1) helped dictate the importance of each component of the design in Figure (1). When there is a reciprocal value in the cell that means the row is less than the column by the denominator value.

The first two key components in Figure (1) are the ease of use and portable. The group felt that the ease of use should be valued higher than the chess set being portable, because no one would want to play a game if it was difficult to use. Also the chess set doesn't have to move to be operating, so the chess set being portable was ranked lower.

The ease of use was divided into three sections: long operating time, standard chess rules, and voice controlled. Voice controlled was valued highest because it is one of the main design requirements. The purpose of this design is to have a chess set that is voice controlled. A sub-division of voice controlled is hands free piece movement. The design group valued standard chess rules at second. The design group wants to keep the chess rules as close as possible to the standard chess rules. Last in the ranking was long operating time.

Portable had two components of lightweight and small. The group valued the chess set being lightweight higher than the chess set being small. The group thought that it would be easier to transport if the chess set weighed less. It was decided against to have a small chess set because it can be difficult to view as a player. Also a smaller chess set would be hard to implement the design fully with the hardware chosen.

[AJ]

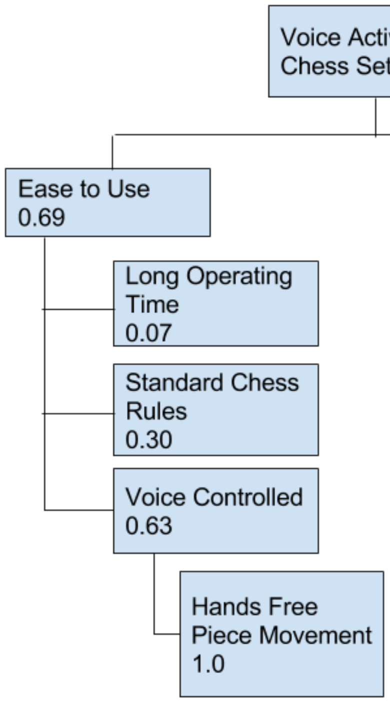


Figure 1: Objective Tree

Table 1: Objective Tree

	Easy to Use	Portable	Weight
Easy to Use	1	5	0.69
Portable	1/5	1	0.31
	Lightweight	**Small**	**Weight**
Lightweight	1	3	0.75
Small	1/3	1	0.25

	Long Operating Time	Standard Chess Rules	Voice Controlled	Weight
Long Operating Time	1	1/5	1/7	0.07
Standard Chess Rule	7	1	1/3	0.30
Voice Controlled	7	3	1	0.63

2. REQUIREMENTS SPECIFICATIONS

2.1 Marketing and Engineering Requirements

Table 2 Marketing and Engineering Requirements

Marketing Requirements	Engineering Requirements	Rationale
Has to follow the rules of chess including piece movement, piece capturing, castling, en passant, promotion, check, and checkmate.	The software should keep tracks of all the positions and movements of the pieces and notify when they are invalid.	It is a chess game.
Has to be completely hands free except for the initial setup of the pieces.	The voice activation module should accept audio input from microphones and send commands to the microcontroller so that it can move the motors.	This will allow handicapped individuals who don't have the use of their arms to play chess.
The board needs to be capable of being carried by a single individual.	Make the device out of lightweight materials where possible. The dimensions of the entire system will not exceed 75x46x27cm.	This device is intended to be played anywhere where a plug is available. It should not have to stay in one place.
The chess piece movement should respond to voice commands in a timely manner.	Microcontroller has a processing speed of 48MHz. Max motor speed of 1200rpm will be able to get the pieces around the board in an appropriate amount of time.	The game should not take longer than a normal game of chess because of piece movement.
Voice commands should be accurately interpreted.	The microphones will need a resistance gain of 680 to be used at arm's length and the voice recognition chip should be programmed with the necessary commands.	This is the main way the player interfaces with the game pieces and should be reliable.
Long and consistent operating time.	120V AC wall outlet to DC power supply to ensure long operation. Position feedback will correct any errors caused by motor slippage.	Chess games can easily become lengthy and this device should be able to keep up with player's demand.

2.2 Constraints

1. **Economic**
   The project should not cost more than $400.

2. **Sustainability**
   The chess set should be able to be moved around easily. The components inside of the chess set should be stationary, so the chess set can be relocated.

3. **Operational**
   The chess set should be able to complete chess piece movements without error.

4. **Social**
   The project should be able to accurately recognize programmed commands by any English speaking user.

2.3 Specifications

Table 3 Specifications

Proposed Specifications	Achieved Specifications	Comments
• *Microcontroller* ○ 128KB Flash memory ○ 48MHz Operating frequency ○ 55 GPIO pins	***Achieved***	
• *Voice Activation Module* ○ Operating Voltage: 5V ○ RX pin voltages (Receiving serial data with highs and lows) ▪ 5V ○ TX pin voltages (Transmitting serial data with highs and lows) ▪ 5V ○ GPIO Pins output voltages (Used for control signal to switch) ▪ 3V ○ Serial Interface: ▪ UART ▪ Baud Rate: 9600 default ▪ 8 Data Bits ▪ No parity bit ▪ 1 Stop bit ▪ No handshaking	***Marginally Achieved***	The GPIO pins were not used in the final design. There was communication difficulties between the EasyVR chip and micro - controller.
• *Position System* ○ Maximum radial load for Stepper Motors 2.7kg ○ Max speed for Stepper Motors 1200RPM ○ Stepper Motor step angle 1.8°	***Achieved***	
• *Electromagnet* ○ Alloy: Iron 1018 ○ Relative permeability: 200 ○ Core diameter: 1.5875 cm ○ Core length: 5.08 cm ○ Wire: 32 gauge enameled copper ○ Turns: 178 ○ Current: 20 mA ○ Magnetic field intensity: 0.01764 T ○ Force: 0.2205 N	***Not Achieved***	The electromagnetic core dimensions were achieved. The amount of turns had to be increased by factor of 30and the current increased drastically. The amount of turns and current had to be increased in order to attach to the chess piece. Unfortunately, the electromagnet didn't hold it's magnetic field for the required time.
• *Power Supply* ○ 120V AC outlet will be used	***Achieved***	

o Output 35V DC, distribute two voltages of 35 and 5V DC ▪ 5V DC to EasyVR device ▪ 5V DC to microcontroller ▪ 35V DC to stepper drives		
	• *Solenoid* o Current required: 0.73 A o Maximum voltage: 36 V DC o Operating voltage: 35V DC o Internal resistance: 18 Ω	The solenoid was used to take place of the electromagnet.
	• *Servo Motor* o Current required: 20 mA o Operating voltage: 5 V DC o Control signal	The servo motor is used to move the microphone back and forth to each player.

3. DESIGN

3.1 Overall

The level 0 block diagram for the Voice-Activated Chess Set is presented below in Figure 2. The major inputs and outputs of the system are represented as wide arrows and the inputs and outputs inside of the system are represented as skinny arrows. The main inputs of the system are pre-trained phrases ("2-1 to 3-1" for example) and are spoken into the voice activation module, through microphones, which are converted into sequential data. The sequential data is used by the microcontroller to determine if the player is following correct chess rules and to generate signals that will be sent to the position system. The position system actuates the movement of the pieces on the board through the use of the stepper motors and a solenoid, for coupling with pieces, underneath of the board. Feedback is sent to the microcontroller to determine if the desired position was achieved, as a second measure.

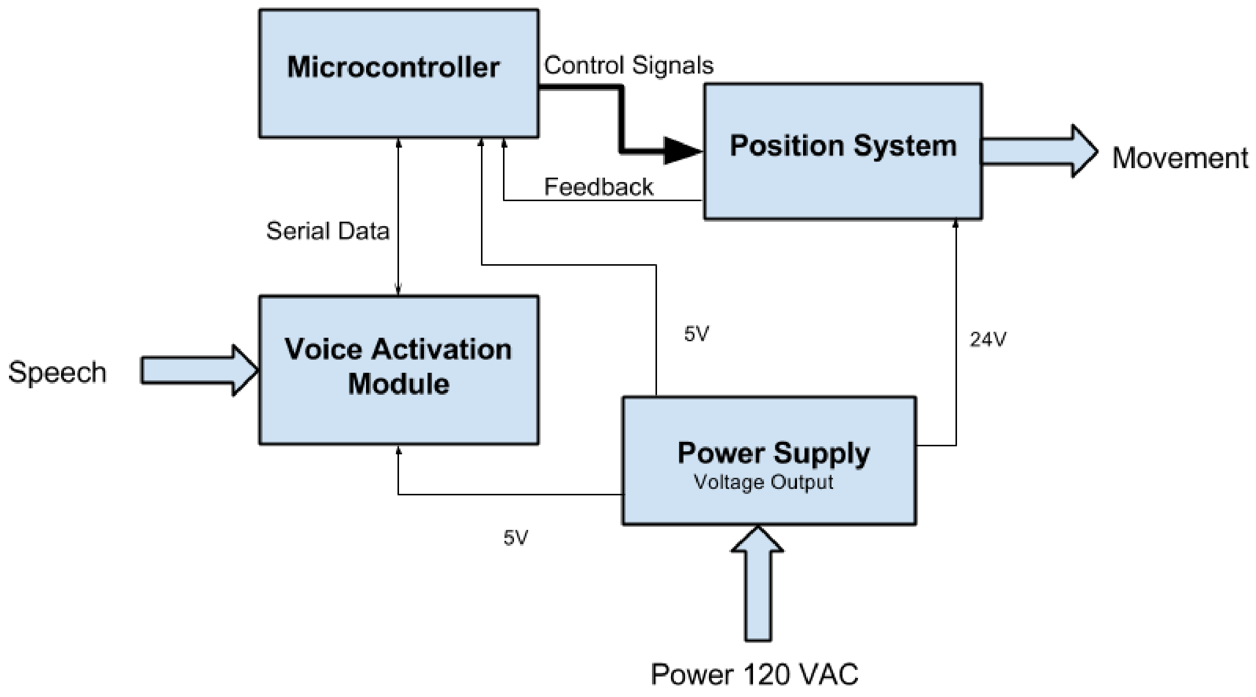
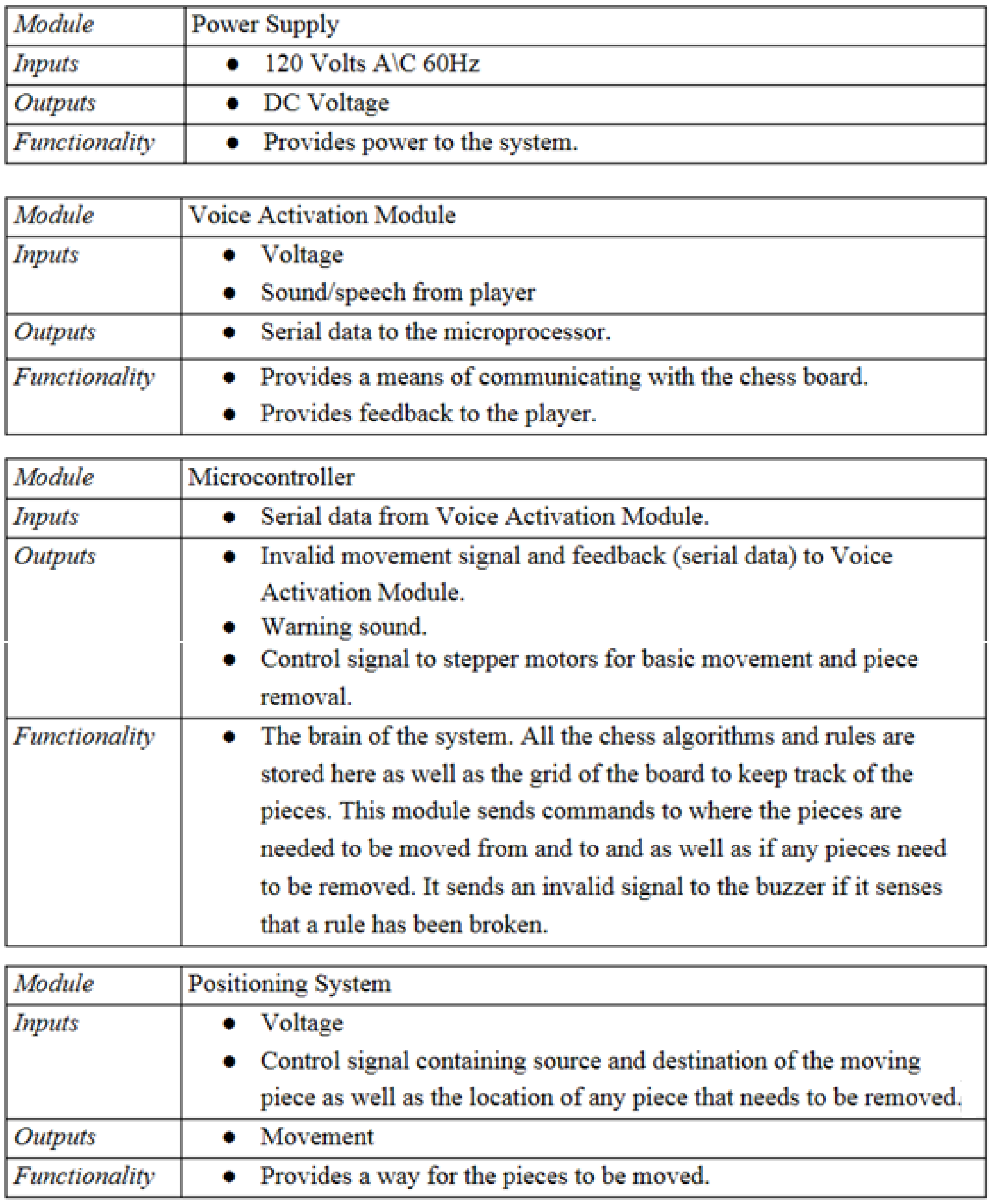


Figure 2: Level 0 Block Diagram

Table 4 Level 0 Chart



Module	Power Supply
Inputs	• 120 Volts A\C 60Hz
Outputs	• DC Voltage
Functionality	• Provides power to the system.

Module	Voice Activation Module
Inputs	• Voltage • Sound/speech from player
Outputs	• Serial data to the microprocessor.
Functionality	• Provides a means of communicating with the chess board. • Provides feedback to the player.

Module	Microcontroller
Inputs	• Serial data from Voice Activation Module.
Outputs	• Invalid movement signal and feedback (serial data) to Voice Activation Module. • Warning sound. • Control signal to stepper motors for basic movement and piece removal.
Functionality	• The brain of the system. All the chess algorithms and rules are stored here as well as the grid of the board to keep track of the pieces. This module sends commands to where the pieces are needed to be moved from and to and as well as if any pieces need to be removed. It sends an invalid signal to the buzzer if it senses that a rule has been broken.

Module	Positioning System
Inputs	• Voltage • Control signal containing source and destination of the moving piece as well as the location of any piece that needs to be removed.
Outputs	• Movement
Functionality	• Provides a way for the pieces to be moved.

[KB, MH]

3.2 Microcontroller

The microcontroller is used to run the algorithms that administrate the rules of chess. The microcontroller receives serial data from the voice activation module (EasyVR) that is used to control the position system via digital signals to stepper motor drivers. There is a dial pad that can be used as an alternative input to chess piece movement than using the microphone. The microcontroller controls three LEDs and a buzzer that give indication to the user of what is happening. The microcontroller also controls the servo motor to move the microphone to the player's direction.

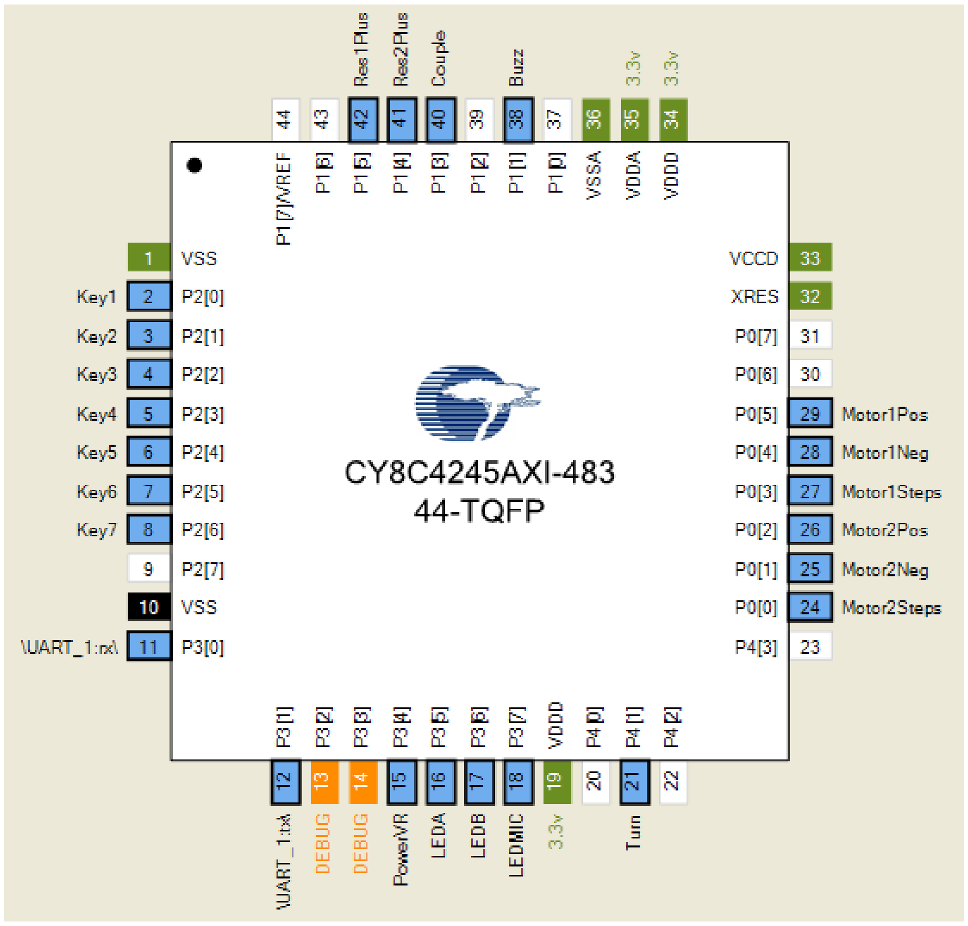


Figure 3: Microcontroller Level 1 Block Diagram

In Figure (3), the pin locations on the Cypress microcontroller can be seen. Pin P4 Vdd is used to receive 5V from the power supply that first went through the 40V to 5V voltage regulator. The ground pin is connected to ground on the power supply. The next pins are all pins that connect to the EasyVR. The pin TX to send to RX from the EasyVR is at 3.1. The pin RX to receive TX from the EasyVR is at 3.0. Pin 3.4 is used to send 5V to the EasyVR and ground is shared between the EasyVR and microcontroller.

The stepper drives need three pins each from the microcontroller to control the movements. For the x direction stepper drive the pins 0.5, 0.4, 0.3 are used. For the y direction stepper drive the pins 0.2, 0.1, 0.0 are used. The three pins control positive steps, positive direction, and negative direction. To control the servo motor to move the microphone the I/O pin on the microcontroller is 4.1.

Other I/O pins are used to control the LEDs and buzzer. One LED was used for player 1 to indicate their turn, that pin is 3.6. Another LED was used for player 2 and that pin was 3.5 on the microcontroller. A third LED is used to indicate to the player that the microphone is listening for a command and the pin is used 3.7. A buzzer is used to indicate the command is invalid and the pin used it 1.1.

[AJ]

3.2.1 Microcontroller Software

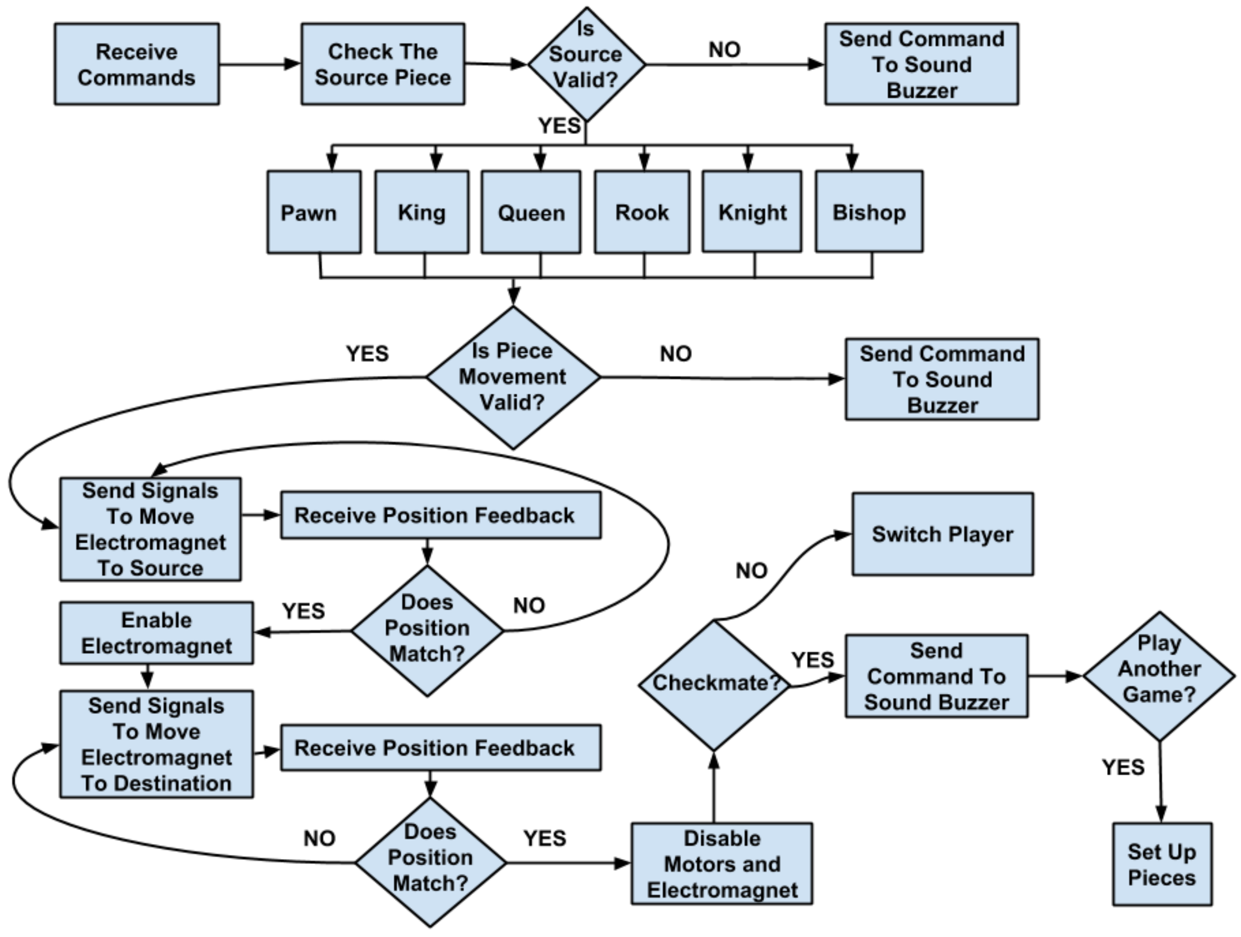

Receive Commands
Check The Source Piece
Is Source Valid?
NO
Send Command To Sound Buzzer
YES
Pawn
King
Queen
Rook
Knight
Bishop
YES
Is Piece Movement Valid?
NO
Send Command To Sound Buzzer
Send Signals To Move Electromagnet To Source
Receive Position Feedback
Switch Player
NO
YES
Does Position Match?
NO
Enable Electromagnet
YES
Checkmate?
Send Command To Sound Buzzer
Play Another Game?
Send Signals To Move Electromagnet To Destination
Receive Position Feedback
YES
NO
Does Position Match?
YES
Disable Motors and Electromagnet
Set Up Pieces

Figure 4: Software Flow Chart for Microcontroller

To make the Voice-Activated Chess work properly, a program was loaded into the microcontroller through a USB connector already on the module and is written in C programming language. Figure 4 above shows the main functions of the program and how it will flow. The most important part of the program is to detect if the movement received from the voice activation module is valid and to send the correct control signals to the position system so that the chess piece can be moved. It starts by setting up the board in memory, initializing the UART and ADC functionality, and constantly sending signals to the EasyVR device using the TX pin and waiting for serial data to be received on the RX pin with the UART interface. The voice recognition device chosen works by constantly communicating with the microcontroller, so the program has to send a predetermined command to the device, making it listen to the microphone, wait for a success byte, and then ask for the index of the command received. All the commands have an index associated with them. The program uses many if statements to determine what to do with each byte received. For example, if 0x00001001 is received, the program will know that the player wants to access the first row on the board and then if 0x00001001 is received next, it knows that the player wants to look at the 11 position on the board. It then determines what piece it is and if it belongs to the current player. The program has fluid code, giving it room for human error so if a player makes a mistake, it won't move ahead.

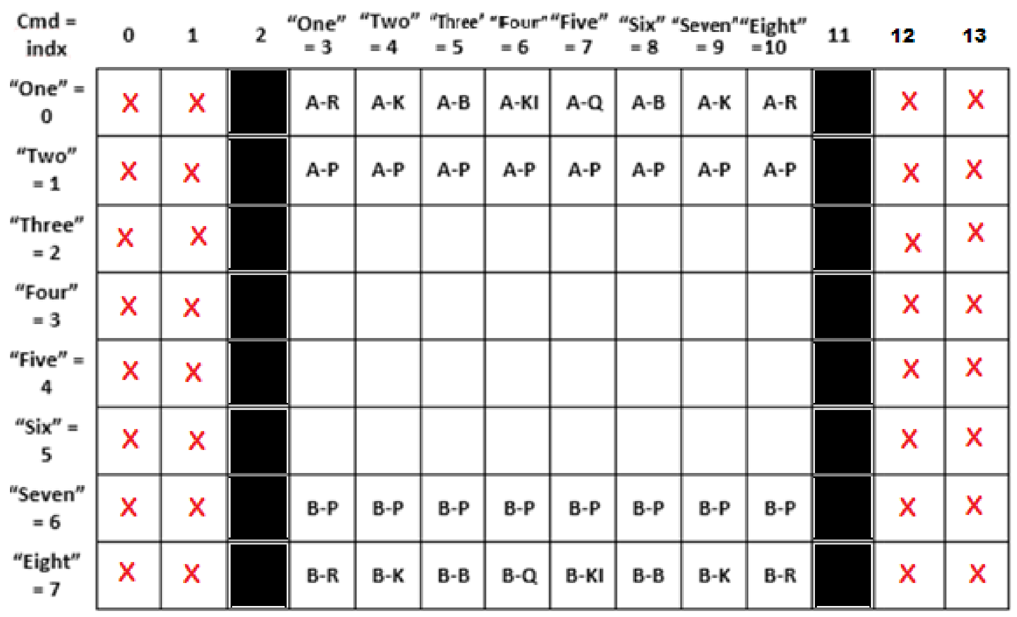
To determine what type of piece is at the source, an 8x14 matrix is implemented in memory, coded as a global, two-dimensional array of pointers to character arrays, mimicking the chess board plus the graveyard slots and extra spacers as shown in Figure 5. The extra spacers are colored black and the squares with the red Xs are the graveyard slots. The first number command corresponds to the first index of the array, the row index, and the second number command corresponds to the second index of the array, the column index. So "one" is the zeroth index of the first array and "three" is the fifth index of the second array. The program then checks what is stored at this place in the array. The chess pieces are coded as strings as shown in Figure 5. If there is no chess piece at the source or if it is the wrong player's, a command is sent to the buzzer, giving it voltage, and notifying the player. If there is a piece there, the program determines what piece it is and goes into a series of branches. Here, the validity of the movement is determined. For example, a bishop can only move to the slots in the matrix diagonal to it. The program makes sure there are no pieces from the same player in the way of the destination. Many of this is done by using nested conditional loops and checking conditions for each piece.

The individual sections for each of the pieces are needed because of special circumstances such as "en passant" for pawns and castling for kings. This part of the code also makes sure the player's king is not in check and therefore invalid movements include leaving him in check. If any movement received is invalid, it sends the command for the buzzer again. If everything is correct, before sending the control signals to the position system module, it also checks if there is an opponent piece at the destination by using the matrix.

If there is no opponent piece, it calculates the required number of steps and direction needed to be sent to the motors to move the solenoid to the source. Initially, the solenoid is in a set position and every time it moves, feedback is read from two of the input pins and the new position is stored in memory. This way, the program knows how many steps to calculate for the stepper motors. Once the movement is calculated, the two enable signals are sent to the motors and the movement control signals.

When feedback is received, the program checks if the position of the solenoid matches the correct spot in the matrix. This is done by using a switch-case clause where each square of the board has a set voltage boundary it should be within. If it does not match, the movement signals for the source are calculated once again, using the new position it was moved to, as shown in the flow chart above labeled Figure 4. Once it matches, the enable for the solenoid is sent from an output pin so that the piece is coupled with the magnet, and then the movement signals for the destination are calculated and sent using the same feedback matching tactic as before. If the feedback is correct, a signal is sent to the motors and solenoid to be disabled.

If there is a piece at the destination that is needed to be taken out, this is done first, using the same tactics above, except the destination being a graveyard slot, and then the source piece will be moved. Each piece has a designated graveyard slot on the side of the board. This makes it easier when a pawn reaches the other end and promotion is needed to be done and also, the board can set itself up to play more games. Initially the program was created so that it checks the board for checkmate after each movement, but there was not enough ROM on the microcontroller and this function has to be disposed of. The program instead waits for a command from a player who cannot make any moves to signify checkmate. It also waits for a specific command to see if the players want to play again. If this command is received from the voice activation module, the pieces are moved from the graveyard and board to their initial places by scanning each square on the board. The board below in Figure 5 will then be set to the beginning values.



Cmd = indx	0	1	2	"One" = 3	"Two" = 4	"Three" = 5	"Four" = 6	"Five" = 7	"Six" = 8	"Seven" = 9	"Eight" = 10	11	12	13
"One" = 0	X	X	■	A-R	A-K	A-B	A-KI	A-Q	A-B	A-K	A-R	■	X	X
"Two" = 1	X	X	■	A-P	A-P	A-P	A-P	A-P	A-P	A-P	A-P	■	X	X
"Three" = 2	X	X	■									■	X	X
"Four" = 3	X	X	■									■	X	X
"Five" = 4	X	X	■									■	X	X
"Six" = 5	X	X	■									■	X	X
"Seven" = 6	X	X	■	B-P	B-P	B-P	B-P	B-P	B-P	B-P	B-P	■	X	X
"Eight" = 7	X	X	■	B-R	B-K	B-B	B-Q	B-KI	B-B	B-K	B-R	■	X	X

Figure 5: Positions of Pieces As Stored In Memory

[KB]

3.3 Voice-Activation Module

In Figure 6, the level 1 block diagram for the Voice-Activation Module can be seen. There are two inputs, speech from player 1 and speech from player 2. There is only one microphone that is used. There is two LEDs on top of the board to indicate which player is permitted to use the microphone. The LEDs are controlled via the microcontroller. The voice recognition chip will take the audio from the microphone and convert it to serial data to send to the microcontroller.

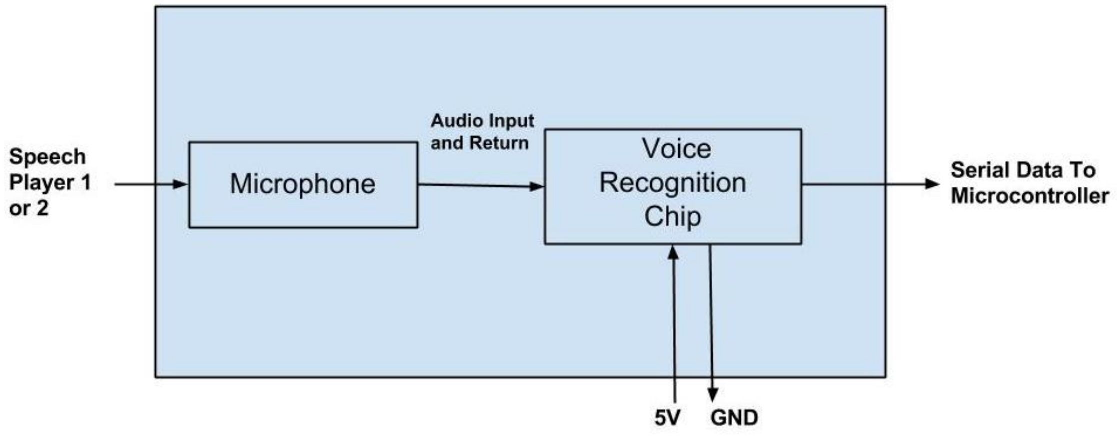


Figure 6: Voice-Activation Module Level 1 Block Diagram

Table 5 VR Block Diagram Chart

Module	Microphone
Inputs	- *Speech Player 1 (or 2)*
Outputs	- *Audio signal*
Functionality	- *Capture speech command*

Module	Voice Recognition Chip
Inputs	- *Audio signal* - *5V*
Outputs	- *Audio output*
Functionality	- *Convert audio signal to serial data* - *Interrupt serial data to create high/low control signal*

In the original level 1 block diagram the voice recognition block had more inputs and outputs. There was supposed to be two microphone connections to this block diagram opposed to only one microphone. Another input that is deleted is serial data coming back to the voice recognition chip from the microcontroller. There was supposed to be a warning buzzer as an output to warn the player of invalid chess movement. Now there is only one microphone but the other functions and connections are on the microcontroller. The complications of the EasyVR chip will be discussed in Section 3.31. [AJ]

3.3.1 Voice-Activation Module Hardware

A few different voice modules were considered but the EasyVR Speech Recognition Module 3.0 was chosen. The EasyVR device was the most cost efficient choice. The EasyVR device can use any host with a UART interface powered at 3.3V-5V. It can have up to 32 user defined speaker verification commands which is a sufficient amount of commands needed to move the chess pieces. In ***Error! Reference source not found.*** the inputs and outputs of the EasyVR evice can be seen.

The EasyVR device was not so easy to use as the name suggests. The first test that was conducted with the EasyVR was displaying an LED using one of the GPIO pins. The second test was to connect the 8 ohm speaker to produce a buzzer sound. Both of the tests were achieved. The third test was to connect the microphone to the EasyVR device. All three components worked for a few days, but then the EasyVR completely stop functioning. There was a concern that electrostatic discharge could have possibly damaged the device, but EasyVR device power indication LED was still on.

The problem delayed the project for almost two weeks. It was decided to make one final test and connect the EasyVR to a computer via UART USB cable. The EasyVR was then connected to the Commander software (software that is used with the EasyVR device) and was rebooted. The EasyVR device became functional in the sense that the microphone could be used but when a GPIO pin was connected to an LED would not work. It was then decided by the design team to only use the EasyVR for player command input. The only reasoning that can be made from this conflict is that the pins draw too much current. This does not allow the microphone to have enough current when other connections are made.

Then few more weeks of working with the EasyVR went by. When the device was connected to the microcontroller it was not realized that the EasyVR was sending random status bytes to the microcontroller. This took a few weeks to debug because it was not understood why the EasyVR would be sending random error, success, or other status bytes. It was also then that it was decided that a dial pad would have to be used as a second command input. It was realized that the microphone couldn't be idling or it would pick up any status. That is why the '*' key on the dial pad has to be pushed to allow the microphone to listen.

Since there were a lot of complications with the EasyVR device only the microphone is connected to the device as well as communication pins to the microcontroller. There is only one microphone that will be used for both players. The microphone is capable to pick up sound an arm length away from the player (roughly 0.6-0.8m). The EasyVR has a cable connector (J6) that directly connects the microphone to the EasyVR. The microphone will rotate by a servo motor which is controlled via the microcontroller. The servo motor will move the microphone to the direction of the player that needs to input a command.

The EasyVR was very difficult to configure. For some reason other pins couldn't be used without disturbing the microphone, so the microphone is the only functionality of the EasyVR that was used. The four pins used were TX and RX to the microcontroller. Also 5V and ground pins were connected to the microcontroller on the EasyVR, and of course the microphone was connected but it wasn't a pin, it was a cable connector (J6). The connections can be seen in Figure 7.

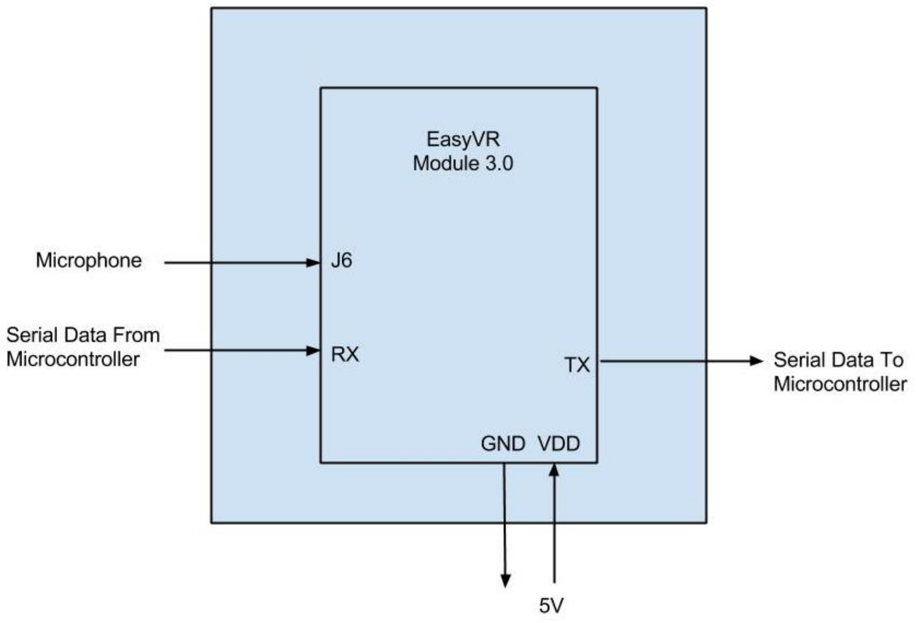

EasyVR
Module 3.0
Microphone
J6
Serial Data From Microcontroller
RX
TX
Serial Data To Microcontroller
GND VDD
5V

Figure 7 Voice Activation Level 2 Block Diagram

When the player says his or her instruction, the EasyVR device will be looking for keywords. The keywords are in Table 8. The EasyVR device will convert the sound into electric pulses and the software will take these as binary bits. The algorithms programmed into the EasyVR device will analyze these signals and create text with them. The text will then be sent to the microcontroller where it will need to be analyzed so that it can send the appropriate commands to the respective module.

[AJ]

3.3.2 Voice-Activation Module Software

The commands that acts as inputs to the system are programmed into the EasyVR device with software that comes with the purchase of it. Each command is stored in either a speaker independent or speaker dependent group of trained vocal phrases. For this system, speaker independent is necessary for any person to be able to use it. However, the software for training custom speaker independent commands is way too expensive for the set budget so no training was actually done to the EasyVR chip. Although, there are enough speaker independent commands programmed into the chip already so substitutions such as letters to numbers and “promotion” to “look” had to be made. To be able to function, the microcontroller needed to talk the chip via a UART protocol. For C programming, which is what this system is using, the EasyVR comes with a header file named “Protocol.h” and contains a list of predefined variables that the MCU can send to the device. The microcontroller program first needs to set up the EasyVR device software, such as telling it what language the commands are going to be said in and at what distance the players will be from the microphones so that it can pick up their voices. Once that is done, each time the VR device receives an audio input, it will try to match it with the list of programmed commands. When a match is made and the microcontroller is asking for one, the device first sends a success byte, waits for the microcontroller to ask it for the command, and sends it back the index of the command in the group. Then, after each command sent, it waits for the microcontroller to send serial data for the next command.

[KB]

3.4 Position System

The position system moves the chess pieces without human motion. This system is made up of two linear motion tracks and a solenoid. The linear motion tracks will move the solenoid underneath of the chess piece from underneath of the playing board. When the solenoid is placed it is energized providing magnetic coupling the iron based chess piece. This will allow movement of the solenoid to translate into movement of the chess piece.

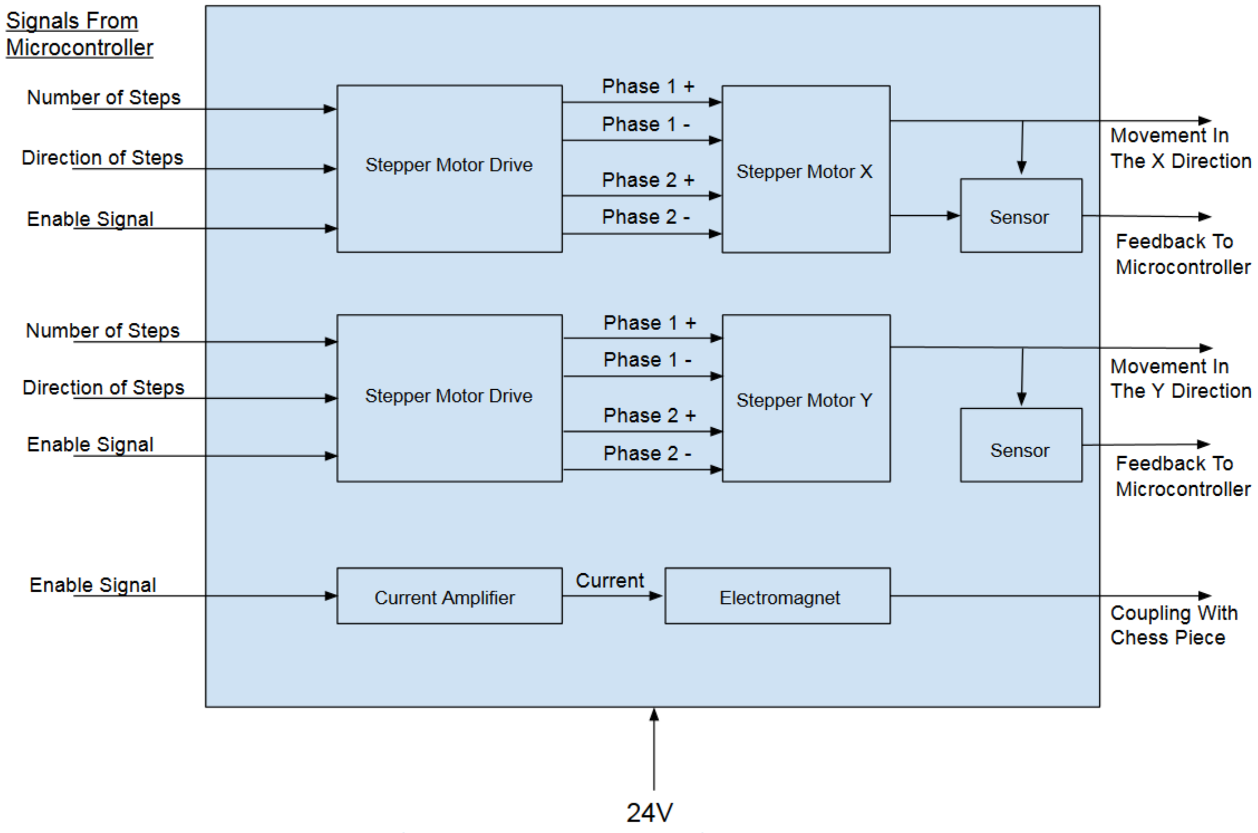


Figure 8: Level One Block Diagram for Position System

Table 6 Position System Block Diagram Chart

Module	Stepper Motor Drive
Inputs	- *Number of steps* - *Direction of steps* - *Enable signal*
Outputs	- *Phase 1 +* - *Phase 1 –* - *Phase 2 +* - *Phase 2 -*
Functionality	- *Control stepper motor*

Module	High Voltage switch
Inputs	- *Enable signal*
Outputs	- *Voltage applied to the solenoid*
Functionality	- *Allows the microcontroller to operate the solenoid*

Module	Stepper Motor X (or Y)
Inputs	- *Phase 1 +* - *Phase 1 –* - *Phase 2 +* - *Phase 2 -*
Outputs	- *Movement in the X (or Y) direction* - *Sensor*
Functionality	- *To relocate solenoid*

Module	Sensor
Inputs	- *Movement in the X or Y direction*
Outputs	- *Feedback to microcontroller*
Functionality	- *Locate solenoid in case it is not position correctly*

Module	Solenoid
Inputs	- *Current*
Outputs	- *Coupling with chess piece*
Functionality	- *To hold onto chess piece to relocate*

The two linear motion tracks is made using two stepper motor drives. The stepper motor drive will take three input signals from the microcontroller; a pulse train which will set the time interval between steps in the motor; a direction signal which is a logic high or low that sets whether the motor steps clockwise or counterclockwise; and an enable command which will also be a logic high or low that allow the stepper motor to respond to other commands.

The linear tracks will have the stepper motor moving a belt and pulley with a support rail. The stepper motor turning will move a belt. The belt is attached to a linear bearing that is moving back and forth along a support rail. This rail is used so that the belt will not have to support weight on its own. A platform is attached to the linear bearing which will move along the rail for the length of the chess board.

There is a lower and upper linear track. The lower linear track is parallel with the chess pieces rows as the horizontal path. A support rail will be needed on the opposite side of the board in order to support both ends of the upper linear track and rail. The opposite rail will not have a pulley system attached to it. The upper linear track will be supported by the linear bearings on the lower linear track.

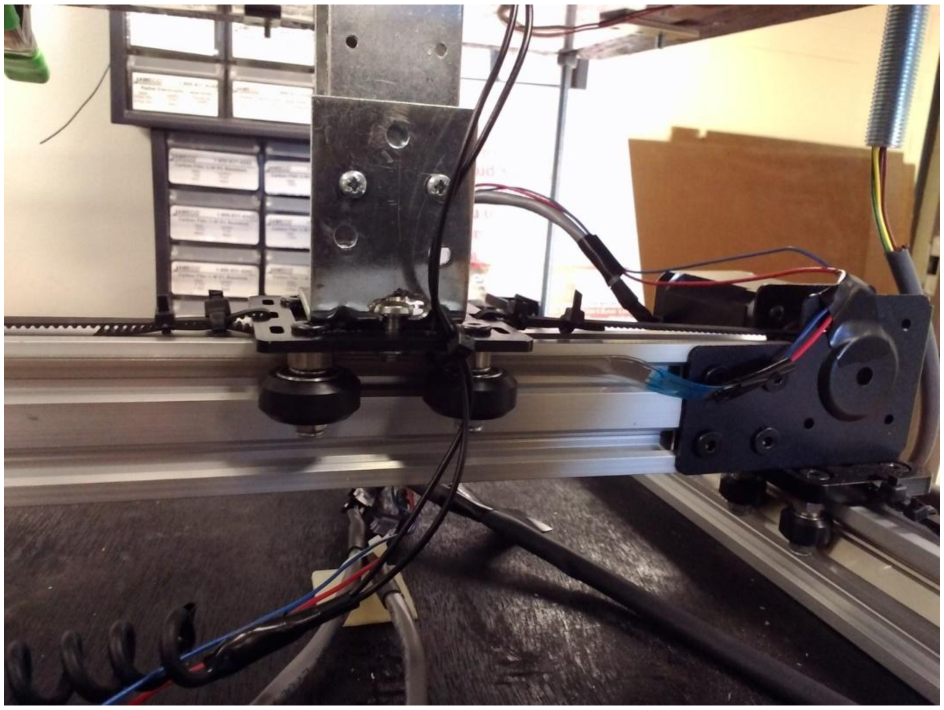


Figure 9 Example of Linear Track

A major advantage of the stepper motors is the ability to be controlled with an open loop. However, for this project some amount of positioning feedback will be required. At start up the microcontroller will need to know the position the solenoid is in both directions in order to provide commands to the stepper motors for their next action. The type of feedback that will be used is two resistive feedback strips. The microcontroller will have map the chess board and graveyards grid locations with the amount of steps the stepper motors will be taking. The microcontroller will compare the current location using the feedback system to the grid map to make sure the solenoid of the position system is in the right chess board square. This will check the position system’s movements.

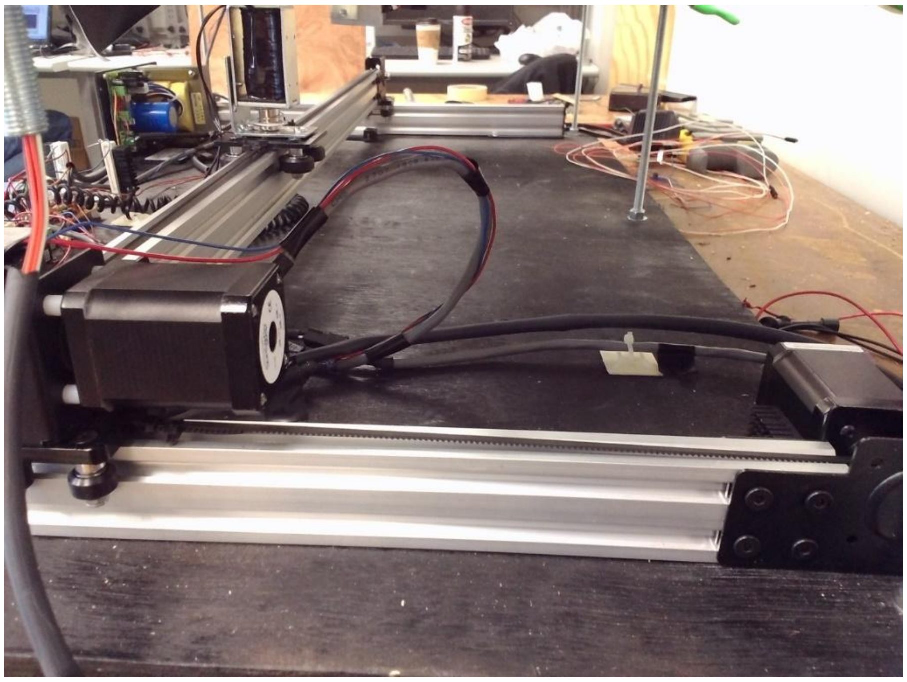


Figure 10: X-Y Table Configuration

The last part of the position system is the solenoid, pictured below in Figure 11. When the solenoid has been moved under the desired chess piece the microcontroller will turn on the solenoid to magnetically couple it with the position system. The microcontroller will then send commands to move the solenoid to a new location determined by the player. Once the position system has reached its destination the solenoid will be disabled to release the chess piece and the position system will await its next command.

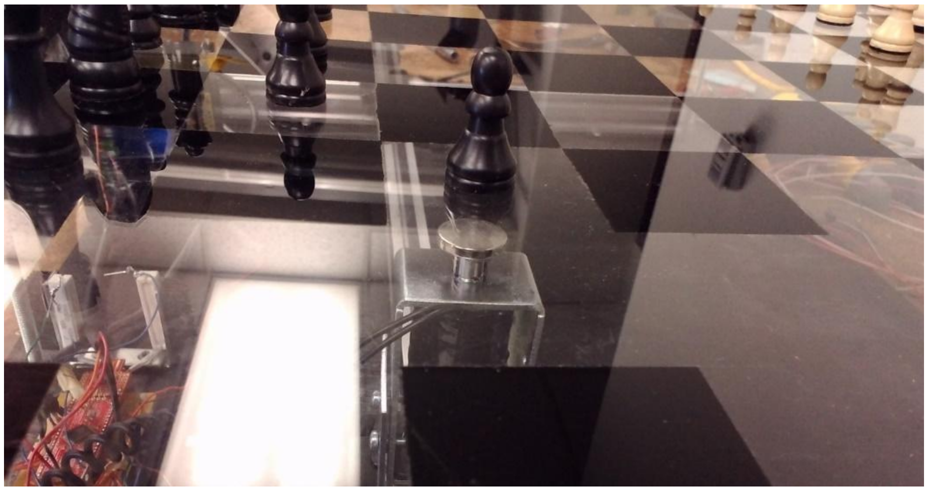


Figure 11: X-Y Table configuration

The position system went through two designs. The first had the platforms attached to linear bearings which the belt would move. This design had to be abandoned because of issues the University had with ordering parts from the vendor that was specified. The vendor used for the redesign, Open Builds, was suggested by Eric Rinaldo. Their parts had a very modular design to them allowing us to easily stack the linear tracks in order to create the X-Y table. A minor issue was tensioning that the belts had in this system. The belt that was bought was open ended and had to be secured to each end of the platform by zip ties, this can be seen in Figure 9. We did have two occasions where the belt came loose and reattaching and tensioning was troublesome.

We did intend to have limit switches at the edges of the boards to be in a startup routine where the system will move the platform to a corner. As the system works currently it assumes that the position at start up is the square 5-2 regardless of the actual position of the platform. The reason that the limit switches was twofold, for out testing they were not needed, but they would be essential for a commercial product.

[WW]

3.5 Feedback

The feedback used for this project is a linear position sensor integrated into the position system. The linear position sensor will monitor the location solenoid on both the x and y axis. Position dependent resistances are used as part of a voltage divider configuration as seen schematically below.

One resistive strip is placed along the x track and the second is placed along the y track. The strip in the x direction is 400mm long and the y direction strip is 750mm long. The total resistance of each strip is 20K Ohm with a tolerance of 20%. The resistive strips is place in-between the wheels of the position systems and the frame as seen in the following Figure 12.

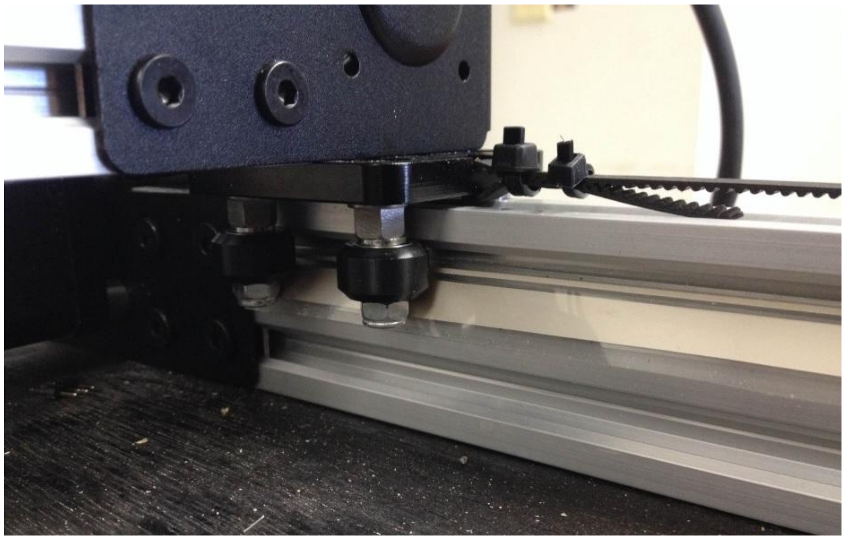


Figure 12 Resistive Feedback Strips Position

As the wheel presses down on the top layer it forces a silver shunt to make contact with a carbon resistor. In Figure 13 an actuator pressed down on the silver shunt is shown.

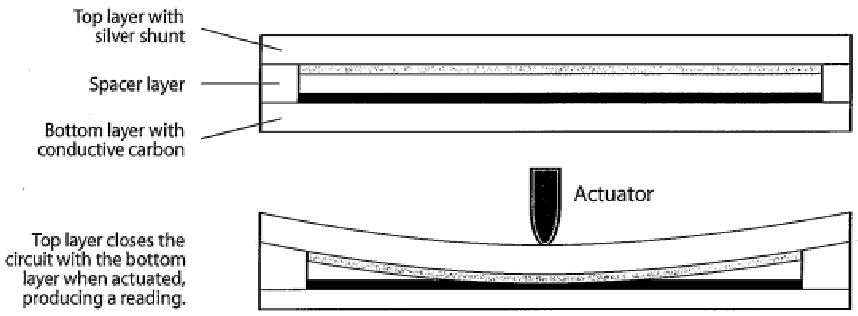

Top layer with silver shunt
Spacer layer
Bottom layer with conductive carbon
Actuator
Top layer closes the circuit with the bottom layer when actuated, producing a reading.

Figure 13 SoftPot Actuation Diagram

The resistance of the potentiometer will change based on where the wheel is pressed down along the strip. When the wheel is at the far side of the track the strip's resistance is 0 kΩ and as the wheel travels along the length of the strip the resistances goes up to 20 kΩ. A 10 kΩ resistor was chosen as the first resistor in the voltage divider so that when the feedback strip is at 0 kΩ the feedback system will not act as a short circuit draining excess current from the power supply.

After the feedback system was installed measurement were taken to test for any defects. In order to obtain to following data set the solenoid was moved manually to the center of each square and on the line in-between squares for both the rows and columns. Figure 14 was generated by entering these resistance values into Matlab and applying a voltage divider formula. It can be seen in Figure 14 that the characteristic curves of the resistance strips have varying consistency. This is unlikely to cause a problem because the strips still have a unique value at each position.

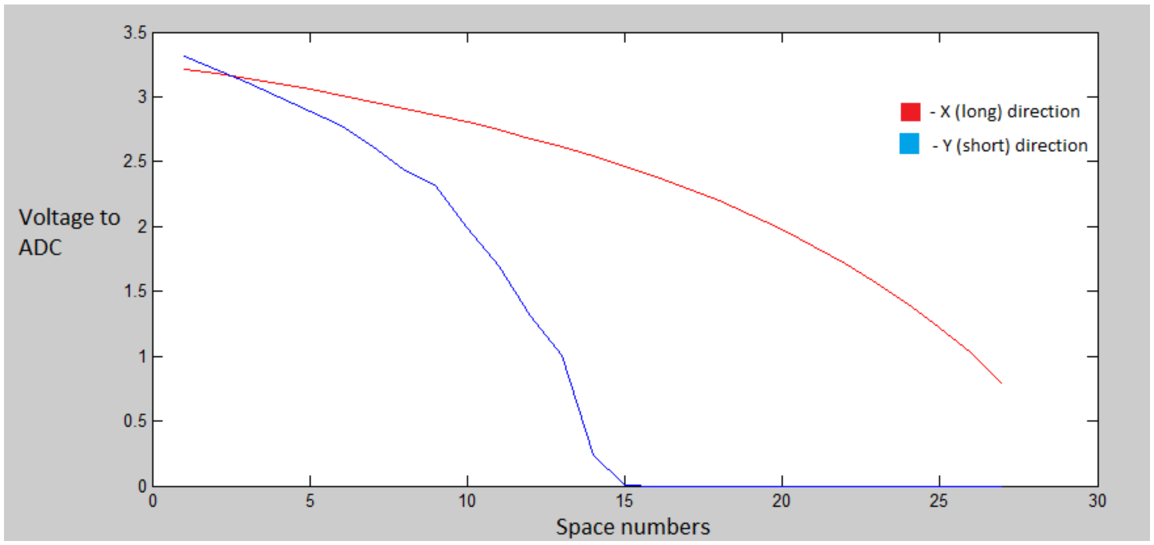


Figure 14 Resistive Strip ADC Voltages per Space

During the debugging phase it was found that when the solenoid moved the chess piece, the piece would lag the solenoid. When the solenoid was moved to the center of a square the chess piece would be approximately 1 cm behind the solenoid. The solution to the problem was to add more steps to the relocation command making the solenoid overshoot its destination and leaving the piece in the center of the square. As a result of this the feedback systems correct position values needed to be recorded to reflect the movement commands of the position system. The microcontroller would need to be programmed with correct voltage ranges in memory that is can use to compare with the actual measurements.

The process to obtain these voltage ranges was to power up the solenoid and the feedback circuit and manually move the solenoid to move pieces while simultaneously recording the voltage values of the feedback circuit through the use of a digital mutli-meter. A piece would be moved one square forward until the piece was centered on the square and the voltage drop across the potentiometer was recorded. The piece was then moved forward by a small amount so that the piece was still in the acceptable center of the square and the voltage value was again recorded. This created an acceptable voltage range that the microcontroller could interpreted as the solenoid being in the correct position. This process of finding voltage ranges was done for moving in each direction, +X, -X, +Y, and -Y to account for lag in each direction.

The feedback system will have a unique voltage reading for each position that the solenoid can be in. The microcontroller will compare this voltage readings with a set of known correct voltage ranges. If the voltage readings does not match what the microcontroller was expecting then the microcontroller will add the necessary number of steps to correct the problem. This ensures that any stepper motor slippage is accounted for.

It is important that the resistive strips stay consistent after long term use because any variation in their characteristic resistance curves changes their *correct voltage ranges* which could result the position system thinking a piece is in the correct position when it is not. The following calibration procedure should be followed in order to correct for any variations that might occur over time.

- Make sure that the playing board is fix in place. Any variation in it position will misalign the feedback sensors.
- Remove all but one chess piece from the board. This will be the test piece.
- After checking to make sure that the entire device is powered off, disconnect the power wires from the stepper motor drive.
- Using a digital multi-meter measure the voltage drop across the potentiometer that needs to be calibrated.
- Turn on power to the device.
- Gathering *correct voltage ranges* for the rows (Y direction)
    - Move the test piece and the magnet to the D1 square manually
    - Make sure the magnet is centered under the square
        - Record voltage value.

        - Take your voltage value and add 0.01V to get the upper voltage limit then take your voltage and subtract 0.01V to obtain the lower voltage limit. (Vn ± 0.01V)
    - Move the solenoid to the D2 square and repeat previous step recording *correct voltage ranges* for all rows.
- Gathering *correct voltage ranges* for the columns (X direction)
    - Repeat the steps of gathering *correct voltage ranges* in the Y direction .
        - No values needed when the piece is on a designated spacer square. The
- Once collecting *correct voltage ranges* is finished count how many ranges have been collected. There should be 20.
- These voltages should be compared with the values that are currently in the system. If the values are different they should be updated.

[MH]

3.6 Electromagnet/Solenoid

The original design called for an electromagnet to couple with the chess pieces. The electromagnet was designed by using fundamental electromagnetic equations to determine the current and number of turns needed to generate the appropriate force on the chess pieces. The electromagnet was desirable because of the easy idea of turning it on/off. To turn it on/off all that needed to be done was turn on/off current to the windings. The design team thought that the electromagnet would have the right amount of magnetic force to attach to the pieces, but unfortunately the magnetic force was not consistent. Since the electromagnet was not consistent, a solenoid with a magnet on the shaft was used.

The first electromagnet prototype was made of a galvanized steel bolt with magnet wire wrapped into the threads of the bolts. The first prototype was built to the design specification and was extremely weak. It was later discovered that galvanized steel has very poor magnetic properties compared to most iron based materials. The size of the bolt was still too large so it was assumed that the core was not in the saturation reign, because of this more turns were added to the core to try to increase the magnetic flux density. This created a new problem because only so many turns could be added to the core before loops of wire started to fall of the ends. It became apparent that there needed to be a way of contain the large number of turns around the core that would be needed in order to achieve the desired field strength. An empty wire spool from the lab was finally used to hold the magnetic wire turns. This worked out perfectly because it allowed us to test out different magnetic cores.

The first 3 layers of magnet wire were hand wound to ensure that the crucial inner turns of the winding were as tightly wound as possible. After the first three layers were wound the spool was place on an electric drill the large spool containing the magnetic wire was placed on a spinning shaft. The drill was used to wind the remainder of the magnet wire onto the small spool unto it was full. In the previous prototype more turn needed to be added which lead to a time wasting process of disconnecting the electromagnet, soldering more magnet wire on, and rewinding. It is for this reason that the spool was wound to its maximum capacity with the idea being that if the electromagnet was too strong the current could be reduced. This process took approximately one and a half hours. The electromagnet can be seen in Figure 15.

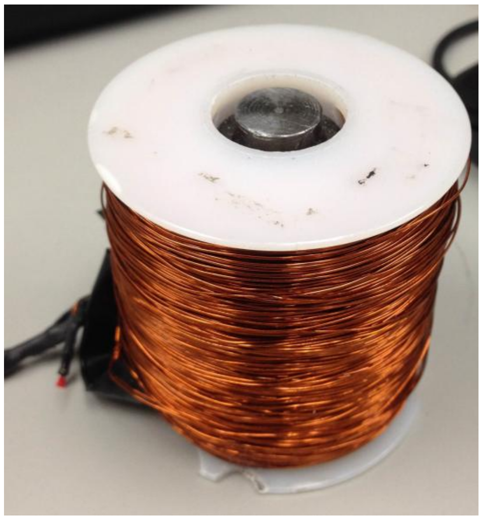


Figure 15 Electromagnet

In total 360 meters of 30 gauge magnet wire was used to construct the solenoid of the electromagnet. The length of magnet wire was found by measuring the internal resistance of the magnet wire and dividing it by the Ω/m value of 30 gauge wire. The number of turns was roughly of 5,000. The electromagnet was still too weak to drag chess pieces and low core quality core material is the suspected cause.

The galvanized steel core of the electromagnet was replaced by 1018 steel which has a theoretical permeability of around 100.The steel 1018 core also did not fit into the wire spool correctly. The spool had an inner diameter of 1 inch and the steel core had a diameter of ½ inches meaning that the air gap in-between the coil and the core likely causes a significant amount of reluctance that impeded magnetic flux. The electromagnet was eliminated because it was not reliable to move the pieces around.

The next design that was used to attach to the chess pieces magnetically underneath of the board was a solenoid. The solenoid had a strong 1 inch diameter disk magnet attached to the shaft of the solenoid. When the solenoid was energized the shaft would lift up making the magnet closer to the board where the chess pieces were resting. The magnet was not directly touching the underneath of the board. It was capable of magnetically coupling with the iron nail inside the chess piece to relocate it. Since the magnet was always going to have a consistent magnetic field, this design was optimal for the overall design of the project.

The solenoid was controlled by a low side transistor switch connected to the 35 volt rail of the power supply. The microcontroller turned the transistor on and off to raise and lower the solenoid respectively. The microcontroller would send a high signal to activate the switch to supply current to the solenoid to raise the shaft up. The solenoid would stay energized until the chess piece relocation was finished. The solenoid with the magnet attached can be seen in Figure 16.

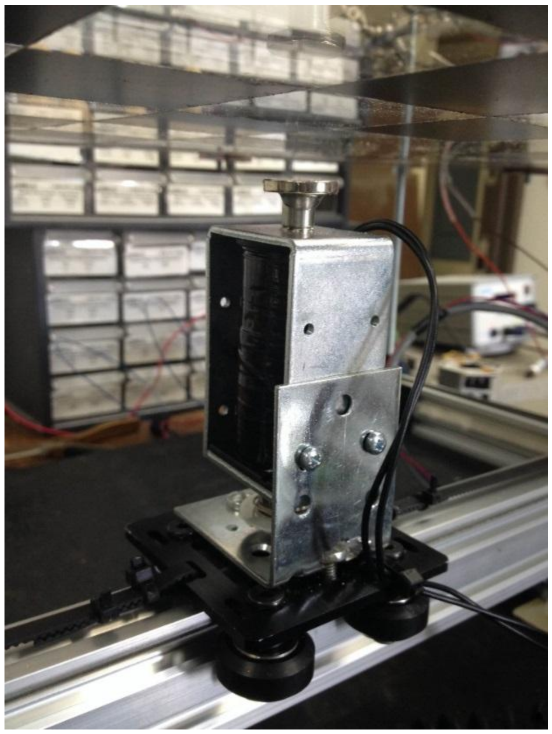


Figure 16: Solenoid

[MH, AJ]

3.7 Power Supply

The power supply needs to be long lasting in case there is a long and competitive game of chess. To ensure that the game can be played for a long time, the power supply is plugged into a standard wall outlet of 120V AC 60 Hz. The power supply chosen for the project is Stepping System Power Supply, model STP-PWR-3024, from Automation Direct. Automation Direct is donating this power supply that goes along with the stepper motors that they are also donating to the project.

The power supply will output 35V DC at 4A. There is an additional output of 5V DC at 1A. The additional output of 5V DC at 1A will supply the Voice Activation Module. There is a need of 5V DC to the microcontroller, so a DC to DC converter will be added to the output 35V DC at 4 A. The DC to DC converter will convert 35V DC to 5V DC at maximum of 2A. Lastly, 35V DC will supply the stepper drives.

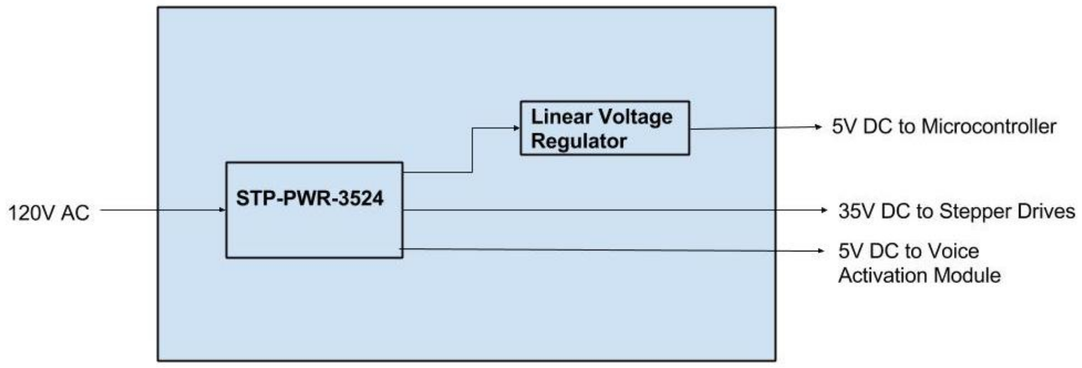


Figure 17 Power Supply Block Diagram 2

Table 7 Power Supply Block Diagram Chart

Module	STP-PWR-3504
Inputs	- *120V AC*
Outputs	- *Linear Voltage Regulator* - *35V DC* - *5V DC*
Functionality	- *Convert 120V AC to 35V and 5V DC.*

Module	Linear Voltage Regulator
Inputs	- *35V DC*
Outputs	- *5V DC with up to 2A*
Functionality	- *Convert the voltage from 35V DC to 5V DC.*

[AJ]

3.8 Circuit Schematic

The schematic version of the overall design can be seen on the next page. This schematic is grouped into 3 section; position system, microcontroller, and voice activation module. The position system is outlined in yellow, the microcontroller in blue, and the voice activation module is outlined in purple.

The position system consists of four different components; the stepper motor drives, stepper motors, feedback potentiometers, and solenoid. The stepper motor drives actuate the stepper motors and the stepper motors will move the solenoid into position. The solenoid will couple with the ferromagnetic material in the chess pieces which will relocate the chess piece. The potentiometer feedback will communicate to the microcontroller the positions system's exact location.

The next main outlined section is the microcontroller. The microcontroller will receive sequential data from the voice activation module regarding current piece location and desired piece location. If the algorithm determines that the piece movement is valid it will send digital control signal to the stepper motor drive. The microcontroller also lights up the 3 LEDs. Two of the LEDs are player indication turn and the third LED indicates that the microphone is listening for a command.

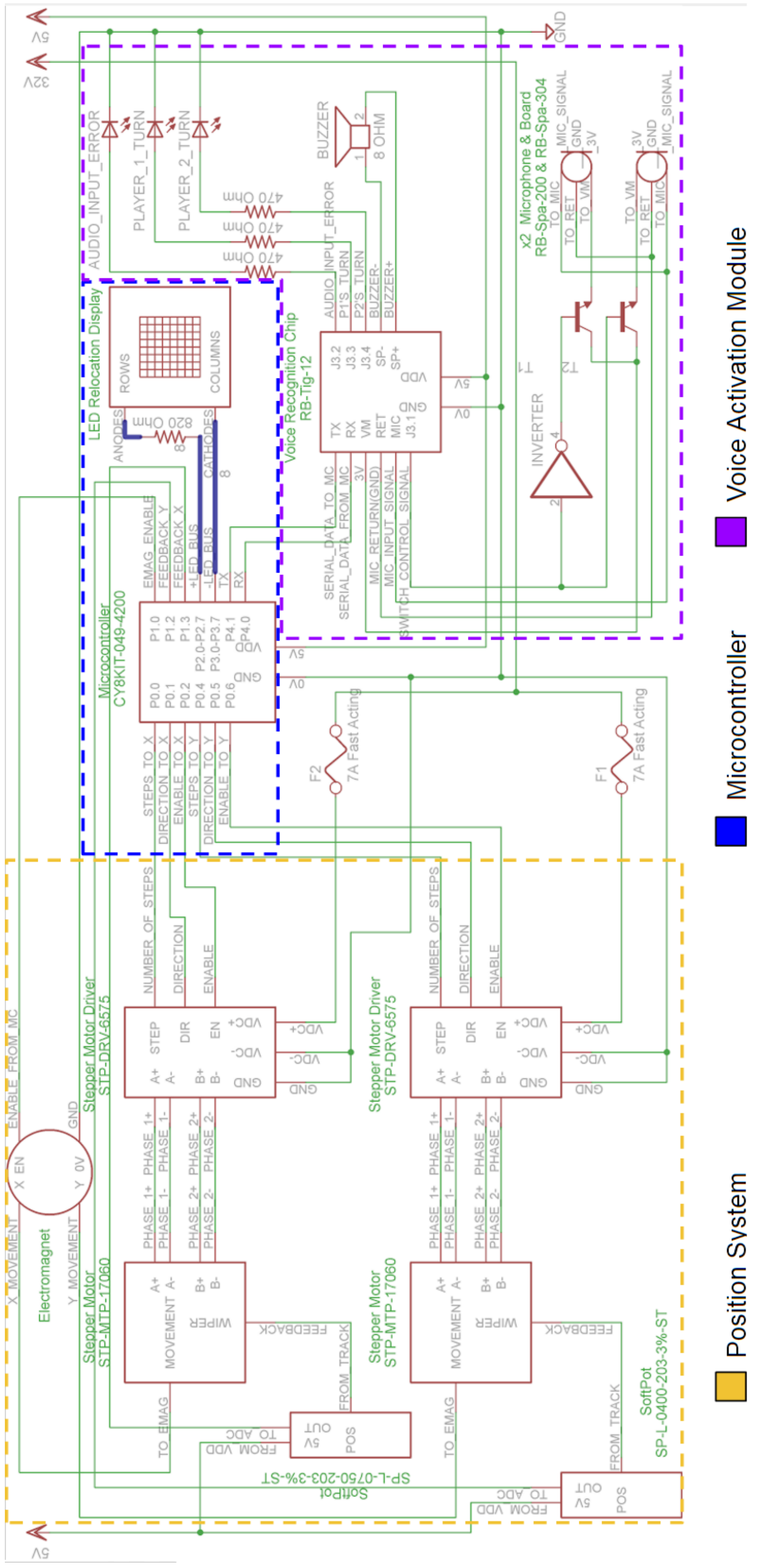
The third outlined component is the voice activation module. This module takes in a voice input via the microphone. It then converts the data to serial data to communicate to the microcontroller. The voice recognition chip is powered by 5 VDC by the power supply.

The following changes have been made since the end of the design stage;

- Two 10 k Ω resistors where added to the feedback system.
- Servo was added to the Voice Activation module.
- Keypad added to Microcontroller module.
- Fly back diode and 150 k Ω resistor added to position system.
- Solenoid replaced the electromagnet.
- Fast acting fuses removed from position system.
- Switching circuit removed from Voice activation module.
- LED display matrix removed.
- Buzzer relocated to microcontroller module.
- LEDs relocated to microcontroller module.
- Two voltage regulators added.
- 470 Ω resistors for LEDs replaced with 150 Ω resistors.
- Low side switch for solenoid added to microcontroller module.
- Two 15 Ω 10 W wire wound power resistors added to solenoid circuit.

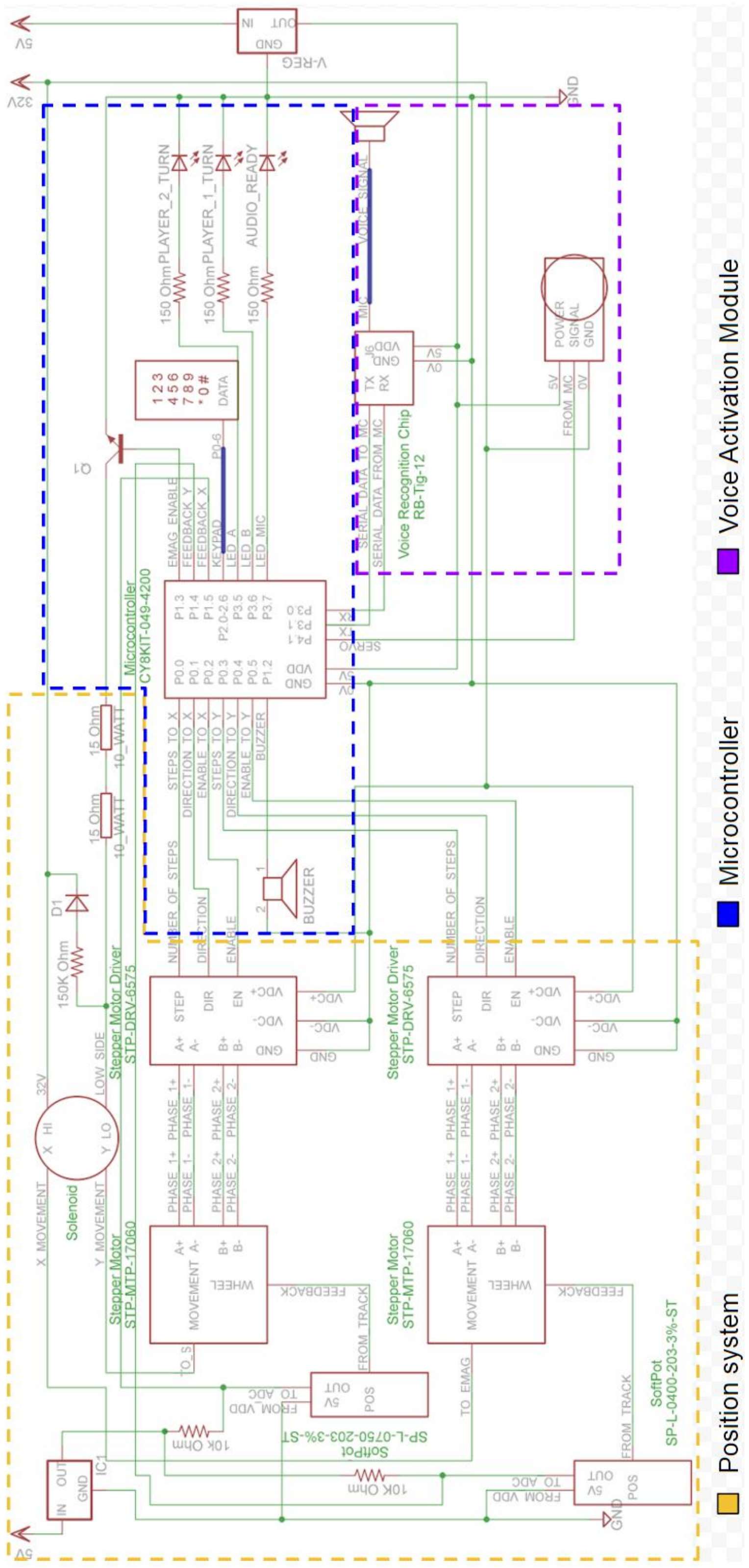
The differences can be seen in Figure 18 and Figure 19.

[MH]



Voice Activation Module
Microcontroller
Position System
LED Relocation Display
Voice Recognition Chip RB-Tig-12
Microcontroller CY8KIT-049-4200
x2 Microphone & Board RB-Spa-200 & RB-Spa-304
BUZZER
8 OHM
INVERTER
T1
T2
F1
F2
7A Fast Acting
Stepper Motor Driver STP-DRV-6575
Stepper Motor STP-MTP-17060
Electromagnet
SoftPot SP-L-0750-203-3%-ST
SoftPot SP-L-0400-203-3%-ST
5V
32V
GND
470 Ohm
820 Ohm

Figure 18 Proposed Overall Schematic



Voice Activation Module
Microcontroller
Position system
Microcontroller CY8KIT-049-4200
Voice Recognition Chip RB-Tig-12
Stepper Motor Driver STP-DRV-6575
Stepper Motor STP-MTP-17060
SoftPot SP-L-0750-203-3%-ST
SoftPot SP-L-0400-203-3%-ST
Solenoid
V-REG
BUZZER
PLAYER_2_TURN
PLAYER_1_TURN
AUDIO_READY
150 Ohm
15 Ohm 10_WATT
150K Ohm
10K Ohm
D1
Q1
IC1
32V
5V
GND

Figure 19 Achieved Overall Schematic

3.9 Chess Set

3.9.1 Overall Chess Set

The overall chess set was not supposed to exceed the dimensions of 75 x 46 x 27 cm but unfortunately it did. The dimensions of the chess set are 101.6 x 50.8 x 25.4 cm. The tracks of the position system were larger than expected. Since the electromagnet was replaced by a solenoid the height of the board also increased. The top of the board is made of Plexiglas and wood. The playing board is made out of Plexiglas that is held in by a sheet wood. Inside of the chess set the location of the power supply, microcontroller, voice activation module, speaker, and circuits can be found. The ideal chess set construction can be seen in *Figure 20* and the actual chess set can be seen in *Figure 21*.

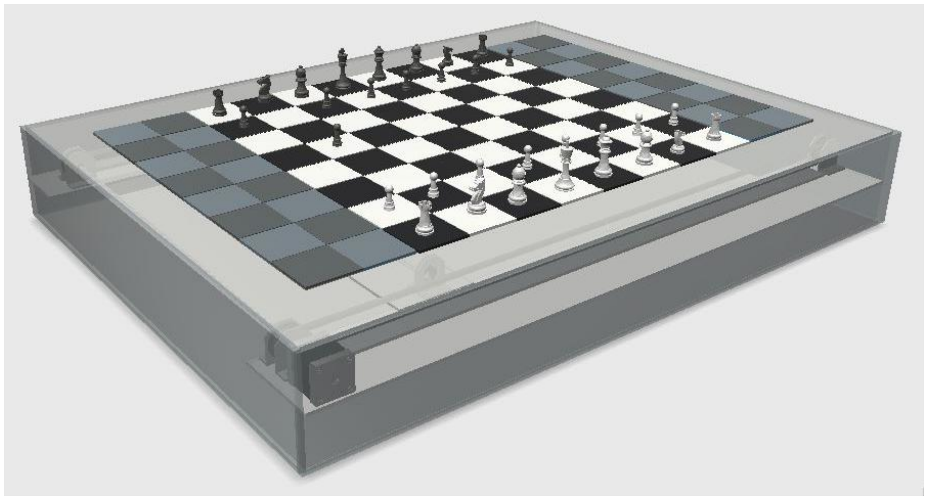


Figure 20 3D Diagram Chess Set

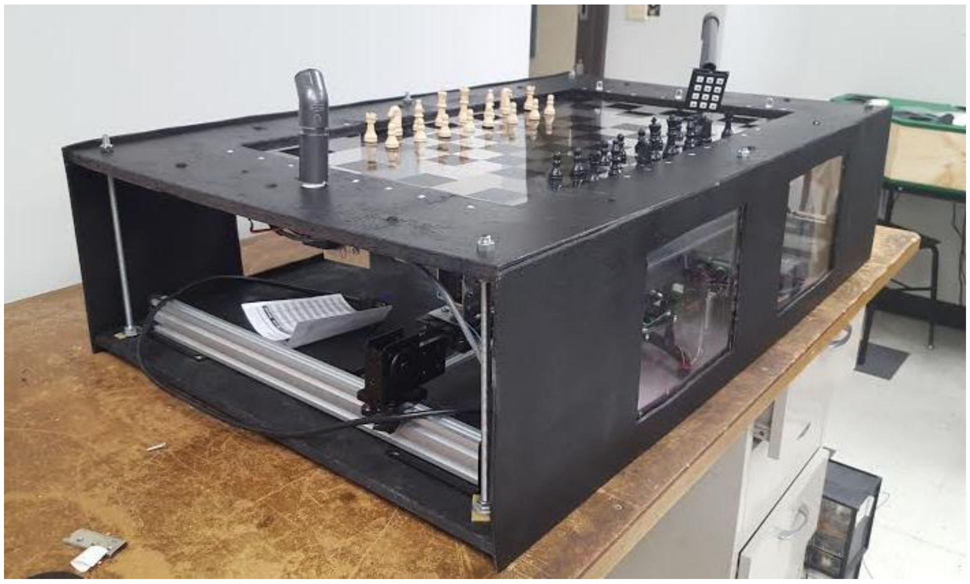


Figure 21 Actual Chess Set

[AJ]

3.9.2 Layout of Board

The area where the chess game is played on what is called the playing board. The playing board is an 8 by 8 grid with 5 cm x 5 cm squares (total area of 40 cm x 40 cm). The chess pieces are organized in a 2 x 8 matrix by chess piece type. The left and right sides of the playing board have areas known as the graveyard. The graveyard is used to place the eliminated pieces off of the playing board. The graveyard is a 2 by 8 grid with squares of 5 cm x 5 cm. The spacing in between the playing board and graveyards is a 5 cm space. There is a 5 cm space form the edge of the board to the edge of the playing board. The overall board was made out of Plexiglas and wood, so there is no magnetic interference. The overall top of the board can be seen in Figure 22.

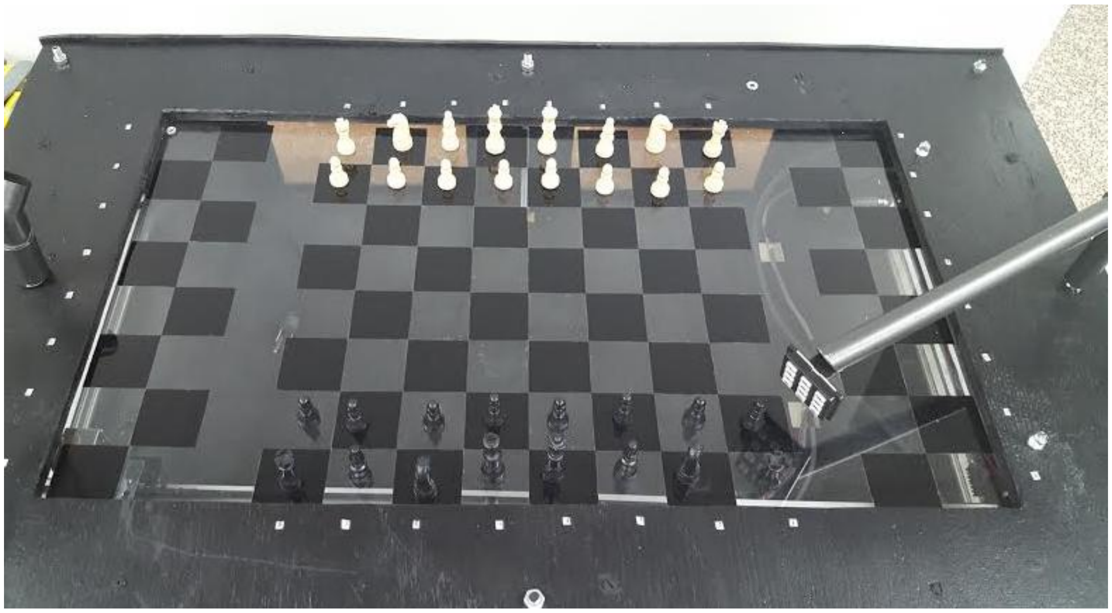


Figure 22 Top Layout of Board

[AJ]

3.9.3 Chess Pieces

The chess pieces are hollow plastic with a base of 1.5 cm diameter. The diameter of the pieces are much smaller than the squares on the playing board (5 cm x 5 cm). In each piece there is an iron nail in the center of a galvanized nut. The iron nail is for the magnet to couple with and the galvanized nut is to add weight which makes the pieces more stable as they move. The size of the pieces makes sure that the magnet on the solenoid won't interfere with other pieces when one piece is attached while being relocated. For example when a knight jumps over a row of pawns, the board and pieces are sized such that the knight is able to slide in between the pawns to the desired location. The first initial set up of the board, the pieces will need to be physically placed on the board. After the first initial set up the pieces will not need physical assistance again as the position system will reposition the pieces for a rematch. A game that would be replayed is hands free because the position system is capable of moving the pieces from the sides of the board, the graveyard slots, to the starting positions.

[AJ]

3.10 Playing the Game

Table 8: Input Commands

Command	Speech	Dial-pad
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
Restart	0	0
Castle	9	9
With rook in column 1	1	1
With rook in column 2	8	8
Promotion	look	#
Rook	1	1
Knight	2	2
Bishop	3	3
Queen	4	4
Notify checkmate	10	#

Controlling the mechanism is done by giving it a series of commands either by voice or by dial pad. To do a voice command, the player must press down on the * button on the keypad. This tells the voice recognition chip to listen for a command. To move a chess piece, the player needs to input a source square and a destination square. The commands are just the numbers of the squares on the grid, done by row first and then column. For example, two commands inputted as "1" and "1" would make the white rook next to the graveyard the source piece. Once four numbers are inputted, the position system will move the solenoid underneath the source square, if there is a piece there, and move it to the destination square, considering it is a legal chess move. This is how the game is played even when capturing. For example, a player may input the destination square for a pawn to be diagonal if there is a piece there. Even "en passant" works this way. Of course, there are other commands programmed in to be used.

For a player to do a castle, they need to either tap "9" on the keypad or say it to the microphone. This puts the player in a castle waiting position and the program waits for input to know what rook to castle with. The rooks are just inputted using the number of the column they are on, either "1" or "8." When a player reaches the other side with one of their pawns, they are put into a promotion waiting position. Here, the player needs to first say "look" or simple tap "#" on the keypad. This puts the mechanism into yet another waiting the position and the player needs to tell the program what piece to look for in the graveyard. As shown in Table 8 above, the numbers 1 through 4 correspond to different pieces. Lastly, once the player cannot do any moves because they are all leaving the king in check, they may signify checkmate by saying "10" to the microphone or by pressing down the # button on the keypad. If the players would like to play another game, they may press “0” on the keypad or say it to the microphone. This will move all the pieces back to their initial positions and readjust the solenoid.

[KB]

4. MAINTENANCE

There is not as much maintenance that needs to be kept up with the chess board. Simple checks need to be performed of as checking for loose wires or overheated components. All of the wiring is either solder or held in with a bolt and housing combination. The stepper motors can become hot with constant use of relocating pieces. There are heat sinks on the stepper motors but overheating is something to be aware of. The bearings and tracks of the position system may need lubrication to keep the position system operating smoothly. It is imported to check from time to time that the mounting screws on the platform of the position system do not shake loose as the actuation of the solenoid can cause a jarring motion that can loosen screws.

[AJ, MH]

5. BUDGET

One of the requirements of the design was to stay inside our $400.00 budget. It can be seen in Table 9 that the budget went over $270.00. The prices are rounded up to the nearest dollar but this wouldn’t have improved the fact that the design went over budget. The components that are listed in Table 9 are the components that were actually used in the achieved design. There have been additional components that were ordered and not used. Unfortunately the design group used all $400.00 that the school provided to us.

Our design group was very fortunate to have many people and companies donated to us. We had Automated Direct donate the stepper motors and drives to us. The microcontroller and stepper drives were donated to us by our group member, William’s father. The EasyVR device was donated from Mitchell’s intern company. The rest of the donations came from the University or other group member’s family members.

Table 9 Actual Cost

Purchased Components	Donated Components	Price
Position System		210.00
Feedback		40.00
Servo		15.00
LEDs		8.00
Buzzer		6.00
Plexiglass		43.00
Chess Pieces		5.00
	Stepper Motors	36.00
	Stepper Drives	140.00
	Power Supply	40.00
	Small Electronics	15.00
	Wood	10.00

	Mechanical Hardware	25.00
	Dail Pad	13.00
	Selenoid	14.00
	Microcontroller	10.00
	EasyVR	40.00
Total		**670.00**

The budget could have been improved in a few ways. One way would have been not purchasing excess Plexiglass. Another would have been having the capability to purchase the position system from Amazon than from Open Build. Amazon would have cut the cost in half. A third way would have been to create a circuit to convert audio signal from the microphone to serial data for the microcontroller to read.

[AJ]

6. CONCLUSION

A lot of time and effort was used to create the design of the chess board in the fall semester. Spring semester was the design teams chance to implement the design that was so thought carefully about. We had three major components of our design altered. The altercations didn't cause our design to dramatically change but it did push back our schedule.

The first component of the overall design that was changed was replacing the electromagnet with a solenoid with a magnet attached to the shaft. The electromagnet was not strong enough to move the largest chess pieces (the queen) and it also would lose its magnetic field after 20 minutes of being energized. This was not going to be suitable to play a game of hands free chess. An immediate solution needed to be put to place and the solenoid was a great replacement.

The second component that changed the design work that was done in the fall semester was the EasyVR device. If a LED, 8 ohm speaker, a second microphone would be attached to the other pins of the device the primary microphone would not function. The additional pins on the device would draw too much current to them and not supply enough current to the clip location of the primary microphone. The 8 ohm speaker had to be replaced with a buzzer that was an output of the microcontroller. The indication LEDs were moved from the EasyVR device to the microcontroller. Also the design had to be altered to have one microphone instead of two microphones. An additional problem with the EasyVR device is when the microphone would idle random status bytes would be sent to the microcontroller. To avoid this dail pad had to be incorporated in the design. The dail pad was used as a secondary input and a way to turn on/off the microphone.

The third altercation to the design was the overall chess set structure. The position system covered a larger area than proposed and it also added height clearance. The solenoid also forced the top layout of the board to be raised. Another component that was not thought about was the weight of the power supply. It was a lot heavier then imagined. The original design requirement was to have the chest set be capable for one person to carry. Since there was a lot of mechanical components that was not thought about, the chess set have to be carried by two people.

[AJ]

REFERNCES

[1] Disability Status: 2000, "Census 2000 brief", United States department of commerce, Available: https://www.census.gov/prod/2003pubs/c2kbr-17.pdf

[2] H. Amason, J. Burbridge, B. Nottingham, T. Tran, “Magic Chess,” University of Central Florida. (Spring 2014) Available: http://eecs.ucf.edu/seniordesign/fa2012sp2013/g03/documents/SDCP2.pdf

[3] “How to Build an Arduino Powered Chess Playing Robot,” Instructables. Available:http://www.instructables.com/id/How-to-Build-an-Arduino-Powered-Chess-Playing-Robot

[4] “EasyVR Speech Recognition Module 3.0,” Robotshop, Available: http://www.robotshop.com/en/easyvr-speech-recognition-module-30.html

[5] R. Clarke, "Magnetism: quantities, units and relationships", University of Surrey, Available: http://info.ee.surrey.ac.uk/Workshop/advice/coils/terms.html#eflen

[6] "Friction Formula", Tutorvista, Available: http://formulas.tutorvista.com/physics/friction-formula.html

[7] "Friction and Coefficients of Friction", The Engineering Toolbox, Available: http://www.engineeringtoolbox.com/friction-coefficients-d_778.html

[8] S. Norr, "Magnetic Circuits", University of Minnesota Duluth, Available: http://www.d.umn.edu/~snorr/ece4501s4/MAGCKTS.PDF

[9] "What is Hopkinson's Law?", Electrotecknik, Available: http://www.electrotechnik.net/2014/06/what-is-hopkinsons-law.html

[10] "Solenoid Properties", CalcTool, Available: http://www.calctool.org/CALC/phys/electromagnetism/solenoid

APPENDIX

A. Data Sheets

A.1. Microcontroller-http://www.cypress.com/file/139956/download

A.2. SoftPot (1) https://www.sparkfun.com/datasheets/Sensors/Flex/SoftPot.pdf

A.3. SoftPot (2) - https://www.sparkfun.com/datasheets/Sensors/Flex/SoftPot-Specs.pdf

A.6. Voice Recognition Chip - http://www.robotshop.com/media/files/pdf/manual-easyvr3.pdf

A.7. Stepper Motor Drivers-
https://www.automationdirect.com/static/manuals/surestepmanual/surestepdrive3_datasheet.pdf

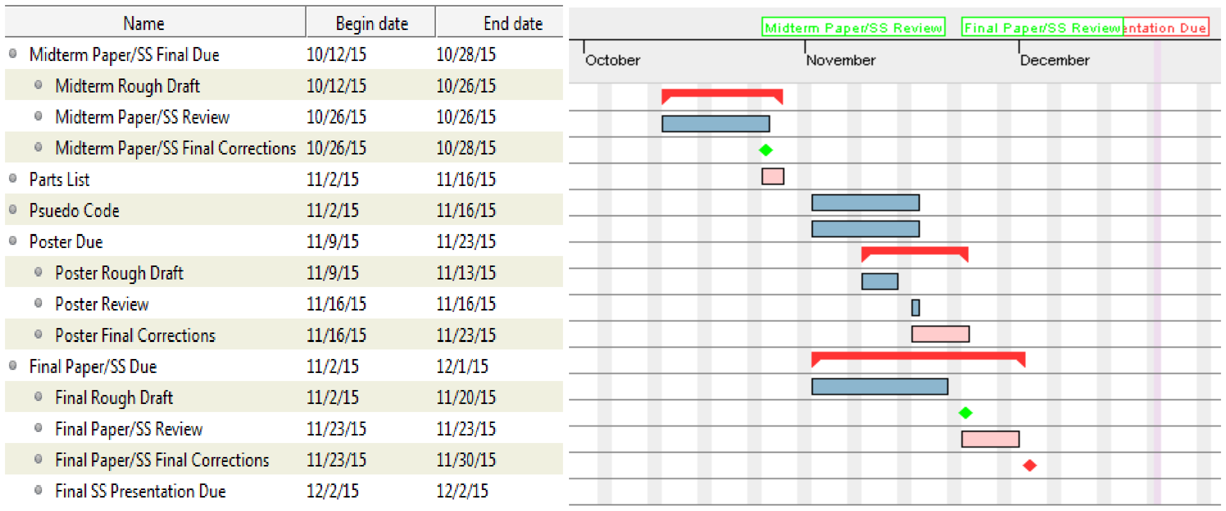
A.8. Stepper Motor - http://cdn.sparkfun.com/datasheets/Robotics/42BYGHM809.PDF

A.9. Inverter - http://www.jameco.com/Jameco/Products/ProdDS/893179.pdf

A.10. N-Channel MOSFET-
http://cdn.sparkfun.com/datasheets/Components/General/FQP30N06L.pdf

B. Gantt Chart

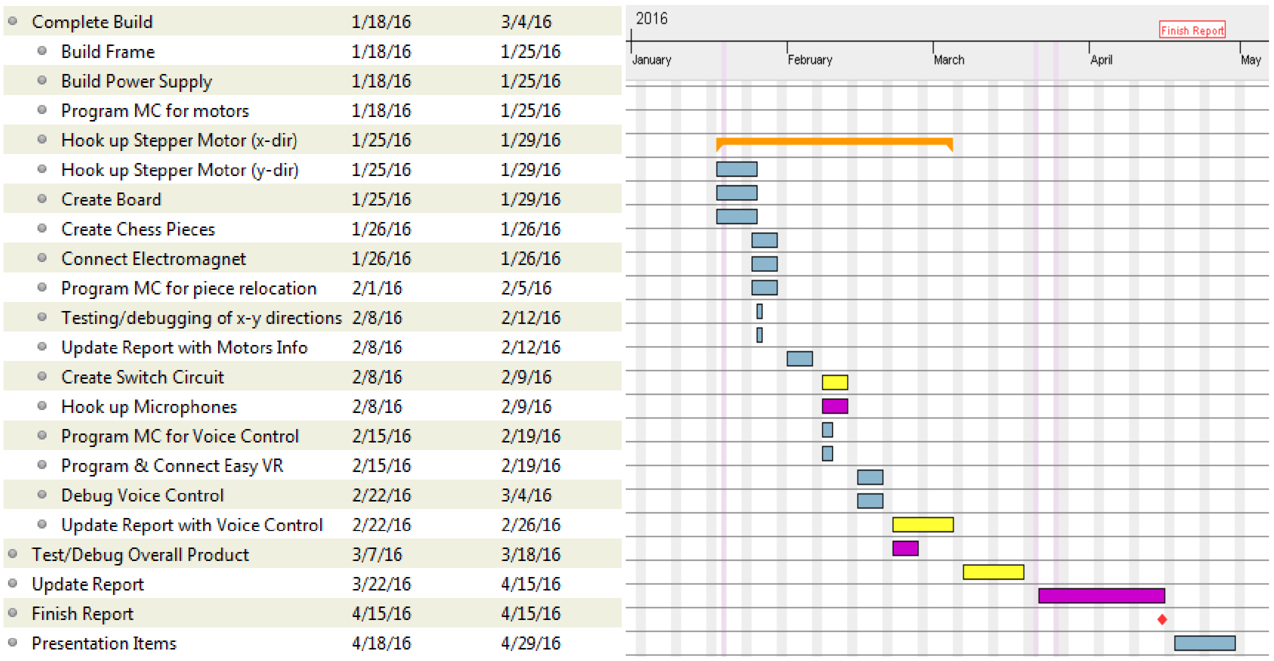
B.1 Fall Semester Schedule



Name	Begin date	End date
Midterm Paper/SS Final Due	10/12/15	10/28/15
Midterm Rough Draft	10/12/15	10/26/15
Midterm Paper/SS Review	10/26/15	10/26/15
Midterm Paper/SS Final Corrections	10/26/15	10/28/15
Parts List	11/2/15	11/16/15
Psuedo Code	11/2/15	11/16/15
Poster Due	11/9/15	11/23/15
Poster Rough Draft	11/9/15	11/13/15
Poster Review	11/16/15	11/16/15
Poster Final Corrections	11/16/15	11/23/15
Final Paper/SS Due	11/2/15	12/1/15
Final Rough Draft	11/2/15	11/20/15
Final Paper/SS Review	11/23/15	11/23/15
Final Paper/SS Final Corrections	11/23/15	11/30/15
Final SS Presentation Due	12/2/15	12/2/15

Fall Semester	
Red	Overall Task Length (Paperwork)
Blue	Paper Rough Draft
Green Diamond	Paper Review with Advisor
Pink	Final Paper Edits

B.2 Spring Semester Schedule



Complete Build	1/18/16	3/4/16
Build Frame	1/18/16	1/25/16
Build Power Supply	1/18/16	1/25/16
Program MC for motors	1/18/16	1/25/16
Hook up Stepper Motor (x-dir)	1/25/16	1/29/16
Hook up Stepper Motor (y-dir)	1/25/16	1/29/16
Create Board	1/25/16	1/29/16
Create Chess Pieces	1/26/16	1/26/16
Connect Electromagnet	1/26/16	1/26/16
Program MC for piece relocation	2/1/16	2/5/16
Testing/debugging of x-y directions	2/8/16	2/12/16
Update Report with Motors Info	2/8/16	2/12/16
Create Switch Circuit	2/8/16	2/9/16
Hook up Microphones	2/8/16	2/9/16
Program MC for Voice Control	2/15/16	2/19/16
Program & Connect Easy VR	2/15/16	2/19/16
Debug Voice Control	2/22/16	3/4/16
Update Report with Voice Control	2/22/16	2/26/16
Test/Debug Overall Product	3/7/16	3/18/16
Update Report	3/22/16	4/15/16
Finish Report	4/15/16	4/15/16
Presentation Items	4/18/16	4/29/16

Spring Semester	
Orange	Overall Task Length (Assembly)
Blue	Assembly Task
Yellow	Debugging
Purple	Update Final Report
Red Diamond	Finish Report Due Date

C. Parts List

Position System	**Part Number**	**Description**	**Quantity**	**Price per Unit**	**Price Total**	**Vendor**	**Website**
feedback	744-SP-L-0400-2033ST	400mm	1	$11.71	$11.71	Mouser	http://www.mouser.com/ProductDetail/Spectra-Symbol/SP-L-0400-203-3-ST/?qs=sGAEpiMZZMvWgbUE6GM3Oc6i9sqQyBsU5xk5VVnNQlQ%3d
feedback	744-SP-L-0750-2033ST	750 mm Linear pressure sensor	1	$27.88	$27.88	Mouser	http://www.mouser.com/ProductDetail/Spectra-Symbol/SP-L-0750-203-3-ST/?qs=sGAEpiMZZMvWgbUE6GM3Oc6i9sqQyBsUI0fLnd5p%2fsI%3d
	kit1005	30in shaft 12mm diameter shaft	1	$22.37	$22.37	Amazon	http://www.amazon.com/12mm-Shaft-Hardened-Linear-Motion/dp/B00FM3X9A8/ref=sr_1_3?s=industrial&ie=UTF8&qid=1448125359&sr=1-3&keywords=linear+motion+12mm+shaft
	SF80450	45cm 8mm diameter shaft	1	$30.24	$30.24	Amazon	http://www.amazon.com/FBT-Diameter-8-Hardened-Linear-Motion/dp/B00WJFRAHU/ref=sr_1_fkmr0_1?s=industrial&ie=UTF8&qid=1448125905&sr=1-1-fkmr0&keywords=linear+motion+12mm+shaft+450mm
	SK8	8mm shaft supports	1	$8.92	$8.92	Amazon	http://www.amazon.com/FBT-support-SK8-Linear-Support/dp/B00X75S9SA/ref=sr_1_1?m=A1VPX6Z58UA6YU&s=merchant-items&ie=UTF8&qid=1448127278&sr=1-1&keywords=8mm+shaft+support
	SK10	12mm shaft supports	1	$9.04	$9.04	Amazon	http://www.amazon.com/FBT-support-SK10-Linear-Support/dp/B00X75SGOW/ref=sr_1_22?m=A1VPX6Z58UA6YU&s=merchant-items&ie=UTF8&qid=1448126427&sr=1-22&keywords=12mm+bearing#biss-product-description-and-details
	LM12UU	12mm bearing	1	$4.01	$4.01	Amazon	http://www.amazon.com/SODIAL-SCS12UU-LM12UU-Bearing-Bushing/dp/B00U8MPSE8/ref=sr_1_1?s=hi&ie=UTF8&qid=1448117668&sr=1-1&keywords=Linear+Motion+Bearing+1%2F2in
	kit991	8mm bearing	2	$10.27	$20.54	Amazon	http://www.amazon.com/Linear-Motion-Bearing-Closed-Metric/dp/B002BBH5Z4/ref=sr_1_8?s=industrial&ie=UTF8&qid=1448127984&sr=1-8&keywords=8mm+linear+bearing
	1254N17	Belt pulley	2	$6.30	$12.60	McMaster	http://www.mcmaster.com/#1254n17/=zwq5bs
	7887K74	1/4in belt 5ft6in	1	$4.00	$4.00	McMaster	http://www.mcmaster.com/#timing-belts/=zydl6u
	7887K221	1/4in belt 3ft 2in	1	$2.71	$2.71	McMaster	http://www.mcmaster.com/#timing-belts/=zydl6u
	550	idler pulley	2	$5.45	$10.90	McMaster	http://openbuildspartstore.com/smooth-idler-pulley-wheel-kit/
		Total			**$164.92**		
Power Supply							
	102-2306-ND	DC-DC converter-out 5V 2A max	1	$17.53	$17.53	Digikey	http://www.digikey.com/product-detail/en/VYB10W-Q48-S5-T/102-2313-ND/2690090
		Total			**$17.53**		

Voice Activated Module							
	RB-Spa-1160	speaker	1	$0.95	$0.95	Robotshop	http://www.robotshop.com/en/025w-thin-speaker.html
Microphone	RB-Spa-304	Microphone board	2	7.95	$15.90	Robotshop	http://www.robotshop.com/en/sfe-breakout-board-electret-microphone.html
	RB-Spa-200	Microphone	2	0.95	$1.90	Robotshop	http://www.robotshop.com/en/sfe-electret-microphone.html
	276001 000002003	Metal gooseneck arm tubing 13"	2	$7.95	$15.90	Musiciansfreind	http://www.musiciansfriend.com/accessories/musicians-gear-goose-neck/276001000002003?cntry=us&source=3WWRWXGP&gclid=COG65vLttskCFRCGaQodl6wAOQ&kwid=productads-plaid^142912834101-sku^276001000002003@ADL4MF-adType^PLA-device^c-adid^82795616067
	CD40106BE	inverter	1	$0.39	$0.39	jameco	http://www.jameco.com/webapp/wcs/stores/servlet/ProductDisplay?langId=-1&storeId=10001&productId=893179&catalogId=10001&CID=GOOG&gclid=CLK5wPOvpckCFQseHwod5fQAHg
	FQP30N06L	nmos	2	$0.95	$1.90	sparkfun	https://www.sparkfun.com/products/10213
		Total			**$36.94**		
Misc.							
		.25 thk, 36x20 inches, clear plexiglass,f/m	1	$40.37	$40.37	professionalplastics	http://www.professionalplastics.com/PLEXIGLASS-ACRYLICSHEET-EXTRUDED
		.098 thk, 36x20 inches, clear plexiglass, f/m	1	$21.93	$21.93	professionalplastics	http://www.professionalplastics.com/PLEXIGLASS-ACRYLICSHEET-EXTRUDED
		Total			**$62.30**		
		Complete Total			**$281.69**		

D. Microcontroller Software Code

```
/* ========================================
 *
 * Copyright YOUR COMPANY, THE YEAR
 * All Rights Reserved
 * UNPUBLISHED, LICENSED SOFTWARE.
 *
 * CONFIDENTIAL AND PROPRIETARY INFORMATION
 * WHICH IS THE PROPERTY OF your company.
 *
 * ========================================
*/
#include <project.h>
#include <stdlib.h>
#include "EasyVR.h"

char *cBoard[8][14];                                  //the board
int kingAR, kingAC, kingBR, kingBC = 0;         //keep track of king
locations
int sR, sC, dR, dC;                                         //keep track of
given commands
int currentC = 4, currentR = 4;                       //keep track of current
location
char player = 'A';                                          //keep track of
the player
int pieceMoved[6];                                          //keep track of
rook and king movement for castling
int stepsPerSq = 172;                                 //amount of steps to
move 1 square
int width = 7u;

void playSX(){
      Buzz_Write(1u);
    CyDelay(1000u);
    Buzz_Write(0u);
}

void moveBackMore(int r, int c){
    int k;
    CyDelay(300u);
    if(sR!=r){
        for(k=0;k<30;k++){
            Motor2Steps_Write(1u);
                CyDelay(width);
                Motor2Steps_Write(0u);
                CyDelay(width);
        }
    }
    if(sC!=c){
        for(k=0;k<30;k++){
            Motor1Steps_Write(1u);
                CyDelay(width);
                Motor1Steps_Write(0u);
                CyDelay(width);
        }
    }
    CyDelay(20u);
    Couple_Write(0u);
    CyDelay(20u);

    if(Motor1Pos_Read()==1u){
        Motor1Pos_Write(0u);
        Motor1Neg_Write(1u);
    }
    else{
        Motor1Pos_Write(1u);
        Motor1Neg_Write(0u);
    }

    if(Motor2Pos_Read()==1){
        Motor2Pos_Write(0u);
        Motor2Neg_Write(1u);
    }
    else{
        Motor2Pos_Write(1u);
        Motor2Neg_Write(0u);
    }

    if(sR!=r){
        for(k=0;k<35;k++){
            Motor2Steps_Write(1u);
                CyDelay(width);
                Motor2Steps_Write(0u);
                CyDelay(width);
        }
    }
    if(sC!=c){
        for(k=0;k<35;k++){
            Motor1Steps_Write(1u);
                CyDelay(width);
                Motor1Steps_Write(0u);
                CyDelay(width);
        }
    }

    if(Motor1Pos_Read()==1u){
        Motor1Pos_Write(0u);
        Motor1Neg_Write(1u);
    }
    else{
        Motor1Pos_Write(1u);
        Motor1Neg_Write(0u);
    }

    if(Motor2Pos_Read()==1){
        Motor2Pos_Write(0u);
        Motor2Neg_Write(1u);
    }
    else{
        Motor2Pos_Write(1u);
        Motor2Neg_Write(0u);
    }
```

```
}

//check to see if the move about to be done won't leave/make the king in
check
int verifyNotCheck(){
      int kingR, kingC;
    int k, j;
    char *tempBoard[8][14];
      int firstPiece;

    for(k=0;k<8;k++){
        for(j=0;j<14;j++){
            tempBoard[k][j] = cBoard[k][j];
        }
    }
    tempBoard[dR][dC] = tempBoard[sR][sC];
      tempBoard[sR][sC] = 0;

    if(player == 'A'){
            if(strcmp(cBoard[sR][sC],"A-Ki")==0){
                  kingR = dR;
                  kingC = dC;
            }else{
                  kingR = kingAR;
                  kingC = kingAC;
            }

            //check forward and backwards for danger
        if(kingR>0){
            k = kingR-1;
            j = kingC;
            if(strcmp(tempBoard[k][j],"B-Ki")==0){
                  return 0;
            }
            firstPiece = 0;
            while((k>=0)&&(firstPiece==0)){
                  if(tempBoard[k][j][0]!='\0'){
                        if(strcmp(tempBoard[k][j],"B-
R1")==0||strcmp(tempBoard[k][j],"B-R2")==0||strcmp(tempBoard[k][j],"B-
Q")==0){
                              return 0;
                        }
                        firstPiece = 1;
                  }
                  k--;
            }
            }

        if(kingR<7){
            k = kingR+1;
            j = kingC;
            if(strcmp(tempBoard[k][j],"B-Ki")==0){
                  return 0;
            }
            firstPiece = 0;
            while((k<=7)&&(firstPiece==0)){
                  if(tempBoard[k][j][0]!='\0'){
                        if(strcmp(tempBoard[k][j],"B-
R1")==0||strcmp(tempBoard[k][j],"B-R2")==0||strcmp(tempBoard[k][j],"B-
Q")==0){
                              return 0;
                        }
                        firstPiece = 1;
                  }
                  k++;
            }
        }

            //check sideways in each direction for danger
            k = kingR;
            j = kingC-1;
            if(strcmp(tempBoard[k][j],"B-Ki")==0){
                  return 0;
            }
            firstPiece = 0;
            while((j>=3)&&(firstPiece==0)){
                  if(tempBoard[k][j][0]!='\0'){
                        if(strcmp(tempBoard[k][j],"B-
R1")==0||strcmp(tempBoard[k][j],"B-R2")==0||strcmp(tempBoard[k][j],"B-
Q")==0){
                              return 0;
                        }
                        firstPiece = 1;
                  }
                  j--;
            }

            k = kingR;
            j = kingC+1;
            if(strcmp(tempBoard[k][j],"B-Ki")==0){
                  return 0;
            }
            firstPiece = 0;
            while((j<=10)&&(firstPiece==0)){
                  if(tempBoard[k][j][0]!='\0'){
                        if(strcmp(tempBoard[k][j],"B-
R1")==0||strcmp(tempBoard[k][j],"B-R2")==0||strcmp(tempBoard[k][j],"B-
Q")==0){
                              return 0;
                        }
                        firstPiece = 1;
                  }
                  j++;
            }

            //check diagonally in each direction for danger
        if(kingR>0){
            k = kingR-1;
            j = kingC-1;
            if(strcmp(tempBoard[k][j],"B-Ki")==0){
                  return 0;
            }
            firstPiece = 0;
            while((k>=0)&&(j>=3)&&(firstPiece==0)){
                  if(tempBoard[k][j][0]!='\0'){
                        if(strcmp(tempBoard[k][j],"B-
B")==0||strcmp(tempBoard[k][j],"B-Q")==0){
                              return 0;
                        }
                        firstPiece = 1;
                  }
                  k--;
                  j--;
            }

            k = kingR-1;
            j = kingC+1;
            if(strcmp(tempBoard[k][j],"B-Ki")==0){
                  return 0;
            }
            firstPiece = 0;
            while((j<=10)&&(k>=0)&&(firstPiece==0)){
                  if(tempBoard[k][j][0]!='\0'){
                        if(strcmp(tempBoard[k][j],"B-
B")==0||strcmp(tempBoard[k][j],"B-Q")==0){
                              return 0;
                        }
                        firstPiece = 1;
                  }
                  k--;
                  j++;
            }
        }

        if(kingR<7){
            k = kingR+1;
            j = kingC-1;
            if(strcmp(tempBoard[k][j],"B-Ki")==0||strcmp(tempBoard[k][j],"B-
P")==0){
                  return 0;
            }
            firstPiece = 0;
            while((j>=3)&&(k<=7)&&(firstPiece==0)){
                  if(tempBoard[k][j][0]!='\0'){
                        if(strcmp(tempBoard[k][j],"B-
B")==0||strcmp(tempBoard[k][j],"B-Q")==0){
                              return 0;
                        }
                        firstPiece = 1;
                  }
                  k++;
                  j--;
            }

            k = kingR+1;
            j = kingC+1;
            if(strcmp(tempBoard[k][j],"B-Ki")==0||strcmp(tempBoard[k][j],"B-
P")==0){
                  return 0;
            }
            firstPiece = 0;
```

```
            while((j<=10)&&(k<=7)&&(firstPiece==0)){
                  if(tempBoard[k][j][0]!='\0'){
                        if(strcmp(tempBoard[k][j],"B-
B")==0||strcmp(tempBoard[k][j],"B-Q")==0){
                              return 0;
                        }
                        firstPiece = 1;
                  }
                  k++;
                  j++;
            }
        }

            //check for knights
        if(kingR<7){
            if(kingC<9){
                  if(strcmp(tempBoard[kingR+1][kingC+2],"B-K")==0)
                        return 0;
            }
            if(kingC>5){
                  if(strcmp(tempBoard[kingR+1][kingC-2],"B-K")==0)
                        return 0;
            }
        }
        if(kingR<6){
            if(strcmp(tempBoard[kingR+2][kingC+1],"B-K")==0)
                  return 0;
            if(strcmp(tempBoard[kingR+2][kingC-1],"B-K")==0)
                  return 0;
        }
        if(kingR>0){
            if(kingC<9){
                  if(strcmp(tempBoard[kingR-1][kingC+2],"B-K")==0)
                        return 0;
            }
            if(kingC>5){
                  if(strcmp(tempBoard[kingR-1][kingC-2],"B-K")==0)
                        return 0;
            }
        }
        if(kingR>1){
            if(strcmp(tempBoard[kingR-2][kingC+1],"B-K")==0)
                  return 0;
            if(strcmp(tempBoard[kingR-2][kingC-1],"B-K")==0)
                  return 0;
        }
    }
    else if(player == 'B'){
        if(strcmp(cBoard[sR][sC],"B-Ki")==0){
                  kingR = dR;
                  kingC = dC;
            }else{
                  kingR = kingBR;
                  kingC = kingBC;
            }

            //check forward and backwards for danger
        if(kingR>0){
            k = kingR-1;
            j = kingC;
            if(strcmp(tempBoard[k][j],"A-Ki")==0){
                  return 0;
            }
            firstPiece = 0;
            while((k>=0)&&(firstPiece==0)){
                  if(tempBoard[k][j][0]!='\0'){
                        if(strcmp(tempBoard[k][j],"A-
R1")==0||strcmp(tempBoard[k][j],"A-R2")==0||strcmp(tempBoard[k][j],"A-
Q")==0){
                              return 0;
                        }
                        firstPiece = 1;
                  }
                  k--;
            }
            }

        if(kingR<7){
            k = kingR+1;
            j = kingC;
            if(strcmp(tempBoard[k][j],"A-Ki")==0){
                  return 0;
            }
            firstPiece = 0;
            while((k<=7)&&(firstPiece==0)){
                  if(tempBoard[k][j][0]!='\0'){
                        if(strcmp(tempBoard[k][j],"A-
R1")==0||strcmp(tempBoard[k][j],"A-R2")==0||strcmp(tempBoard[k][j],"A-
Q")==0){
                              return 0;
                        }
                        firstPiece = 1;
                  }
                  k++;
            }
            }

            //check sideways in each direction for danger
            k = kingR;
            j = kingC-1;
            if(strcmp(tempBoard[k][j],"A-Ki")==0){
                  return 0;
            }
            firstPiece = 0;
            while((j>=3)&&(firstPiece==0)){
                  if(tempBoard[k][j][0]!='\0'){
                        if(strcmp(tempBoard[k][j],"A-
R1")==0||strcmp(tempBoard[k][j],"A-R2")==0||strcmp(tempBoard[k][j],"A-
Q")==0){
                              return 0;
                        }
                        firstPiece = 1;
                  }
                  j--;
            }

            k = kingR;
            j = kingC+1;
            if(strcmp(tempBoard[k][j],"A-Ki")==0){
                  return 0;
            }
            firstPiece = 0;
            while((j<=10)&&(firstPiece==0)){
                  if(tempBoard[k][j][0]!='\0'){
                        if(strcmp(tempBoard[k][j],"A-
R1")==0||strcmp(tempBoard[k][j],"A-R2")==0||strcmp(tempBoard[k][j],"A-
Q")==0){
                              return 0;
                        }
                        firstPiece = 1;
                  }
                  j++;
            }

            //check diagonally in each direction for danger
        if(kingR>0){
            k = kingR-1;
            j = kingC-1;
            if(strcmp(tempBoard[k][j],"A-Ki")==0||strcmp(tempBoard[k][j],"A-
P")==0){
                  return 0;
            }
            firstPiece = 0;
            while((k>=0)&&(j>=3)&&(firstPiece==0)){
                  if(tempBoard[k][j][0]!='\0'){
                        if(strcmp(tempBoard[k][j],"A-
B")==0||strcmp(tempBoard[k][j],"A-Q")==0){
                              return 0;
                        }
                        firstPiece = 1;
                  }
                  k--;
                  j--;
            }

            k = kingR-1;
            j = kingC+1;
            if(strcmp(tempBoard[k][j],"A-Ki")==0||strcmp(tempBoard[k][j],"A-
P")==0){
                  return 0;
            }
            firstPiece = 0;
            while((j<=10)&&(k>=0)&&(firstPiece==0)){
                  if(tempBoard[k][j][0]!='\0'){
                        if(strcmp(tempBoard[k][j],"A-
B")==0||strcmp(tempBoard[k][j],"A-Q")==0){
                              return 0;
                        }
                        firstPiece = 1;
                  }
                  k--;
```

```
                    j++;
                }
            }

            if(kingR<7){
                k = kingR+1;
                j = kingC-1;
                if(strcmp(tempBoard[k][j],"A-Ki")==0){
                    return 0;
                }
                firstPiece = 0;
                while((j>=3)&&(k<=7)&&(firstPiece==0)){
                    if(tempBoard[k][j][0]!='\0'){
                        if(strcmp(tempBoard[k][j],"A-
B")==0||strcmp(tempBoard[k][j],"A-Q")==0){
                            return 0;
                        }
                        firstPiece = 1;
                    }
                    k++;
                    j--;
                }

                k = kingR+1;
                j = kingC+1;
                if(strcmp(tempBoard[k][j],"A-Ki")==0){
                    return 0;
                }
                firstPiece = 0;
                while((j<=10)&&(k<=7)&&(firstPiece==0)){
                    if(tempBoard[k][j][0]!='\0'){
                        if(strcmp(tempBoard[k][j],"A-
B")==0||strcmp(tempBoard[k][j],"A-Q")==0){
                            return 0;
                        }
                        firstPiece = 1;
                    }
                    k++;
                    j++;
                }
                }

                //check for knights
            if(kingR<7){
                if(kingC<9){
                    if(strcmp(tempBoard[kingR+1][kingC+2],"A-K")==0)
                        return 0;
                }
                if(kingC>5){
                    if(strcmp(tempBoard[kingR+1][kingC-2],"A-K")==0)
                        return 0;
                }
            }

            if(kingR<6){
                if(strcmp(tempBoard[kingR+2][kingC+1],"A-K")==0)
                    return 0;
                if(strcmp(tempBoard[kingR+2][kingC-1],"A-K")==0)
                      return 0;
            }

            if(kingR>0){
                if(kingC<9){
                      if(strcmp(tempBoard[kingR-1][kingC+2],"A-K")==0)
                            return 0;
                }
                if(kingC>5){
                      if(strcmp(tempBoard[kingR-1][kingC-2],"A-K")==0)
                            return 0;
                }
            }

            if(kingR>1){
                if(strcmp(tempBoard[kingR-2][kingC+1],"A-K")==0)
                      return 0;
                if(strcmp(tempBoard[kingR-2][kingC-1],"A-K")==0)
                      return 0;
            }
        }

      return 1;
}

void moveToSource(){
      int numSqsV = 0, numSqsH = 0;
      int stepsY, stepsX;

      //check and set direction
      if(sR > currentR){
            numSqsV = sR - currentR;
            Motor2Pos_Write(1u);
            Motor2Neg_Write(0u);
      }
      else if(sR < currentR){
            numSqsV = currentR - sR;
            Motor2Pos_Write(0u);
            Motor2Neg_Write(1u);
      }

      if(sC > currentC){
            numSqsH = sC - currentC;
            Motor1Pos_Write(1u);
            Motor1Neg_Write(0u);
      }
      else if(sC < currentC){
            numSqsH = currentC - sC;
            Motor1Pos_Write(0u);
            Motor1Neg_Write(1u);
      }

      CyDelay(20u);

      //calculate and move the motor
      stepsY = numSqsV*stepsPerSq;
      stepsX = numSqsH*stepsPerSq;
      int k;
      for(k = 0; k < stepsX; k++){
            Motor1Steps_Write(1u);
            CyDelay(width);
            Motor1Steps_Write(0u);
            CyDelay(width);
      }
      for(k = 0; k < stepsY; k++){
            Motor2Steps_Write(1u);
            CyDelay(width);
            Motor2Steps_Write(0u);
            CyDelay(width);
      }

      //set the new position
      currentR = sR;
      currentC = sC;
    CyDelay(1000u);
}

//Checks the resistive strips for the right value range
void CheckRes(int r, int c){
      int16 result;
      float volts;
      result = ADC1_GetResult16(0u);
      volts = ADC1_CountsTo_Volts(0u, result);
    volts = volts*5/3.3;
      int okay = 0;
      int neg = 0, pos = 0;
      //CyDelay(500u);
    Couple_Write(1u);

      //check the y direction
    switch(r){
            case 0 :
                  if(volts>0.13&&volts<0.15){
                        okay = 1;
                  }
                  else if(volts<0.13){
                        pos = 1;
                  }else if(volts>0.15){
                        neg = 1;
                  }
            case 1 :
                  if(volts>1.01&&volts<1.02){
                        okay = 1;
                  }else if(volts<1.01){
                        pos = 1;
                  }else if(volts>1.02){
                        neg = 1;
                  }
            case 2 :
                  if(volts>1.76&&volts<1.78){
                        okay = 1;
                  }else if(volts<1.76){
                        pos = 1;
```

```
                }else if(volts>1.78){
                        neg = 1;
                }
        case 3 :
                if(volts>2.28&&volts<2.3){
                        okay = 1;
                }else if(volts<2.28){
                        pos = 1;
                }else if(volts>2.3){
                        neg = 1;
                }
        case 4 :
                if(volts>2.57&&volts<2.67){
                        okay = 1;
                }else if(volts<2.57){
                        pos = 1;
                }else if(volts>2.67){
                        neg = 1;
                }
        case 5 :
                if(volts>2.94&&volts<2.95){
                        okay = 1;
                }else if(volts<2.94){
                        pos = 1;
                }else if(volts>2.95){
                        neg = 1;
                }
        case 6 :
                if(volts>3.138&&volts<3.16){
                        okay = 1;
                }else if(volts<3.138){
                        pos = 1;
                }else if(volts>3.16){
                        neg = 1;
                }
    case 7 :
        if(volts>3.333&&volts<3.345){
                        okay = 1;
                }else if(volts<3.333){
                        pos = 1;
                }else if(volts>3.345){
                        neg = 1;
                }
}

if(okay==0){
        if(pos==1){
                Motor2Pos_Write(1u);
                Motor2Neg_Write(0u);
        }else if(neg==1){
                Motor2Pos_Write(0u);
                Motor2Neg_Write(1u);
        }
        Motor2Steps_Write(1u);
        CyDelay(width);
        Motor2Steps_Write(0u);
        CyDelay(width);
        Motor2Steps_Write(1u);
        CyDelay(width);
        Motor2Steps_Write(0u);
        CyDelay(width);
        CheckRes(r,c);
  }

result = ADC1_GetResult16(1u);
CyDelay(20u);
  volts = ADC1_CountsTo_Volts(1u, result);
volts = volts*5/3.3;
  okay = 0;
pos = 0;
neg = 0;
  switch(c){
        case 0 :
        if(volts>0.132&&volts<0.175){
                    okay = 1;
              }else if(volts<0.132){
                    pos = 1;
              }else if(volts>0.175){
                    neg = 1;
              }
        case 1 :
        if(volts>0.62&&volts<0.658){
                    okay = 1;
              }else if(volts<0.62){
                    pos = 1;
              }else if(volts>0.658){
                    neg = 1;
              }
        case 3 :
        if(volts>1.4&&volts<1.42){
                    okay = 1;
              }else if(volts<1.4){
                    pos = 1;
              }else if(volts>1.42){
                    neg = 1;
              }
        case 4 :
        if(volts>1.7&&volts<1.718){
                    okay = 1;
              }else if(volts<1.7){
                    pos = 1;
              }else if(volts>1.718){
                    neg = 1;
              }
        case 5 :
        if(volts>1.95&&volts<1.96){
                    okay = 1;
              }else if(volts<1.95){
                    pos = 1;
              }else if(volts>1.96){
                    neg = 1;
              }
        case 6 :
        if(volts>2.17&&volts<2.19){
```

```
                        okay = 1;
                }else if(volts<2.17){
                        pos = 1;
                }else if(volts>2.19){
                        neg = 1;
                }
        case 7 :
        if(volts>2.362&&volts<2.37){
                        okay = 1;
                }else if(volts<2.362){
                        pos = 1;
                }else if(volts>2.37){
                        neg = 1;
                }
        case 8 :
        if(volts>2.537&&volts<2.55){
                        okay = 1;
                }else if(volts<2.537){
                        pos = 1;
                }else if(volts>2.55){
                        neg = 1;
                }
        case 9 :
        if(volts>2.69&&volts<2.7){
                        okay = 1;
                }else if(volts<2.69){
                        pos = 1;
                }else if(volts>2.7){
                        neg = 1;
                }
        case 10 :
        if(volts>2.83&&volts<2.835){
                        okay = 1;
                }else if(volts<2.83){
                        pos = 1;
                }else if(volts>2.835){
                        neg = 1;
                }
    case 12 :
        if(volts>3.06&&volts<3.065){
                        okay = 1;
                }else if(volts<3.06){
                        pos = 1;
                }else if(volts>2.065){
                        neg = 1;
                }
    case 13 :
        if(volts>3.17&&volts<3.175){
                        okay = 1;
                }else if(volts<3.17){
                        pos = 1;
                }else if(volts>3.175){
                        neg = 1;
                }
}
if(okay==0){
        if(pos==1){
                Motor2Pos_Write(1u);
                Motor2Neg_Write(0u);
          }else if(neg==1){
                Motor2Pos_Write(0u);
                Motor2Neg_Write(1u);
          }
          Motor1Steps_Write(1u);
          CyDelay(width);
          Motor1Steps_Write(0u);
          CyDelay(width);
          Motor1Steps_Write(1u);
          CyDelay(width);
          Motor1Steps_Write(0u);
          CyDelay(width);
          CheckRes(r,c);
    }

    Couple_Write(0u);
}

void moveToGrave(int r, int c){
    CyDelay(1000u);
      int numSqsV = 0, numSqsH = 0;
      int stepsY, stepsX;

    //first move to the piece
      if(r > currentR){
            numSqsV = r - currentR;
            Motor2Pos_Write(1u);
            Motor2Neg_Write(0u);
      }
      else if(r < currentR){
            numSqsV = currentR - r;
            Motor2Pos_Write(0u);
            Motor2Neg_Write(1u);
      }

      if(c > currentC){
            numSqsH = c - currentC;
            Motor1Pos_Write(1u);
            Motor1Neg_Write(0u);
      }
      else if(c < currentC){
            numSqsH = currentC - c;
            Motor1Pos_Write(0u);
            Motor1Neg_Write(1u);
      }

      CyDelay(10u);

      stepsY = numSqsV*stepsPerSq;
      stepsX = numSqsH*stepsPerSq;
      int k;
      for(k = 0; k < stepsX; k++){
            Motor1Steps_Write(1u);
            CyDelay(width);
            Motor1Steps_Write(0u);
            CyDelay(width);
      }
      for(k = 0; k < stepsY; k++){
            Motor2Steps_Write(1u);
            CyDelay(width);
            Motor2Steps_Write(0u);
            CyDelay(width);
      }

    //figure out where the piece is moving to
    int gR = 0, gC = 0;

    if(strcmp(cBoard[r][c],"A-P")==0){
        gC = 0;
        for(k = 7; k>-1; k--){
            if(cBoard[k][gC][0]=='\0'){
                gR = k;
            }
        }
    }
    else if(strcmp(cBoard[r][c],"B-P")==0){
        gC = 13;
        for(k = 0; k<8; k++){
            if(cBoard[k][gC][0]=='\0'){
                gR = k;
            }
        }
    }
    else if(strcmp(cBoard[r][c],"A-R1")==0||strcmp(cBoard[r][c],"A-R2")==0){
        gC = 1;
        for(k = 3; k>1; k--){
            if(cBoard[k][gC][0]=='\0'){
                gR = k;
            }
        }
    }
    else if(strcmp(cBoard[r][c],"B-R1")==0||strcmp(cBoard[r][c],"B-R2")==0){
        gC = 12;
        for(k = 2; k<4; k++){
            if(cBoard[k][gC][0]=='\0'){
                gR = k;
            }
        }
    }
    else if(strcmp(cBoard[r][c],"A-B")==0){
        gC = 1;
        for(k = 5; k>3; k--){
            if(cBoard[k][gC][0]=='\0'){
                gR = k;
            }
        }
    }
    else if(strcmp(cBoard[r][c],"B-B")==0){
        gC = 12;
        for(k = 4; k<6; k++){
            if(cBoard[k][gC][0]=='\0'){
                gR = k;
            }
        }
    }
    else if(strcmp(cBoard[r][c],"A-K")==0){
        gC = 1;
        for(k = 1; k>-1; k--){
            if(cBoard[k][gC][0]=='\0'){
                gR = k;
            }
        }
    }
    else if(strcmp(cBoard[r][c],"B-K")==0){
        gC = 12;
        for(k = 6; k<8; k++){
            if(cBoard[k][gC][0]=='\0'){
                gR = k;
            }
        }
    }
    else if(strcmp(cBoard[r][c],"A-Q")==0){
        gC = 1;
        gR = 7;
    }
    else if(strcmp(cBoard[r][c],"B-Q")==0){
        gC = 12;
        gR = 0;
    }

    CyDelay(1000u);
    if(r>0){
        Motor2Pos_Write(0u);
          Motor2Neg_Write(1u);
    }
    else{
        Motor2Pos_Write(1u);
          Motor2Neg_Write(0u);
    }
    CyDelay(20u);
    //calculate direction and do the movement
    if(r > gR){
            numSqsV = r - gR;
            Motor2Pos_Write(0u);
            Motor2Neg_Write(1u);
      }
      else if(r < gR){
            numSqsV = gR - r;
            Motor2Pos_Write(1u);
            Motor2Neg_Write(0u);
      }

      if(c > gC){
            numSqsH = c - gC;
            Motor1Pos_Write(0u);
            Motor1Neg_Write(1u);
      }
      else if(c < gC){
            numSqsH = gC - c;
            Motor1Pos_Write(1u);
            Motor1Neg_Write(0u);
      }

    CyDelay(10u);

      stepsY = (numSqsV-1)*stepsPerSq;
      stepsX = (numSqsH-1)*stepsPerSq;
    Couple_Write(1u);
    CyDelay(20u);
      for(k = 0; k < stepsPerSq/2; k++){
            Motor1Steps_Write(1u);
            CyDelay(width);
            Motor1Steps_Write(0u);
            CyDelay(width);
      }
    CyDelay(50u);
      for(k = 0; k < stepsPerSq/2; k++){
            Motor2Steps_Write(1u);
            CyDelay(width);
            Motor2Steps_Write(0u);
            CyDelay(width);
      }
    CyDelay(50u);
      for(k = 0; k < stepsX; k++){
            Motor1Steps_Write(1u);
            CyDelay(width);
            Motor1Steps_Write(0u);
            CyDelay(width);
      }
      for(k = 0; k < stepsY; k++){
            Motor2Steps_Write(1u);
            CyDelay(width);
            Motor2Steps_Write(0u);
            CyDelay(width);
      }
    CyDelay(50u);
      for(k = 0; k < stepsPerSq/2; k++){
            Motor1Steps_Write(1u);
            CyDelay(width);
            Motor1Steps_Write(0u);
            CyDelay(width);
      }
    CyDelay(50u);
    if(r!=gR){
      for(k = 0; k < stepsPerSq/2; k++){
            Motor2Steps_Write(1u);
            CyDelay(width);
            Motor2Steps_Write(0u);
            CyDelay(width);
      }
    }
    else{
        if(r>0){
            Motor2Pos_Write(1u);
          Motor2Neg_Write(0u);
        }
```

```
        else{
            Motor2Pos_Write(0u);
          Motor2Neg_Write(1u);
        }

        CyDelay(20u);
        for(k = 0; k < stepsPerSq/2; k++){
            Motor2Steps_Write(1u);
            CyDelay(width);
            Motor2Steps_Write(0u);
            CyDelay(width);
      }
    }
    moveBackMore(gR, gC);
    Couple_Write(0u);
    CyDelay(20u);

    //set the current place and move back to source
      cBoard[gR][gC] = cBoard[r][c];
    cBoard[r][c] = 0;
    currentR = gR;
    currentC = gC;
    moveToSource();
}

//picks a piece for the promotion movement
int listenForPiece(){
    char8 result, command = 0u;
    int r = 10;
    int i;
      int eCount = 0;
    Key5_Write(0u);
    Key6_Write(0u);
    Key7_Write(0u);

      if(player=='A'){
            for(i = 0; i<8; i++){
                  if(cBoard[i][1][0]=='\0'){
                        eCount=eCount+1;
            }
            }
      }else if(player=='B'){
            for(i = 0; i<8; i++){
                  if(cBoard[i][12][0]=='\0'){
                        eCount=eCount+1;
            }
            }
      }

      if(eCount==8){
            return 8;
      }

    CyDelay(500u);
      for(;;){
            result = UART_1_UartGetChar();
        if(result > 0u){
```

```
if(result == 's'){
    CyDelay(20);
    UART_1_UartPutChar(' ');
    CyDelay(20);
    while(command == 0u){
        command = UART_1_UartGetChar();
        if(command == 'B'){
                            //look for Rook in graveyard
            if(player=='A'){
                                  if(cBoard[2][1]!='\0')
                                        r = 2;
                                  else if(cBoard[3][1]!='\0')
                                        r = 3;
                                  else{
                                        playSX();
                                        return 10;
                                  }
                            }
                            else if(player=='B'){
                                  if(cBoard[3][12]!='\0')
                                        r = 3;
                                  else if(cBoard[2][12]!='\0')
                                        r = 2;
                                  else{
                                        playSX();
                                        return 10;
                                  }
                            }
        }
        else if(command == 'C'){
            //look for Knight in graveyard
            if(player=='A'){
                                  if(cBoard[1][1]!='\0')
                                        r = 1;
                                  else if(cBoard[0][1]!='\0')
                                        r = 0;
                                  else{
                                        playSX();
                                        return 10;
                                  }
                            }
                            else if(player=='B'){
                                  if(cBoard[7][12]!='\0')
                                        r = 7;
                                  else if(cBoard[6][12]!='\0')
                                        r = 6;
                                  else{
                                        playSX();
                                        return 10;
                                  }
                            }
        }
        else if(command == 'D'){
            //look for Bishop in graveyard
            if(player=='A'){
                                  if(cBoard[4][1]!='\0')
                                        r = 4;
                                  else if(cBoard[5][1]!='\0')
                                        r = 5;
                                  else{
                                        playSX();
                                        return 10;
                                  }
                            }
                            else if(player=='B'){
                                  if(cBoard[5][12]!='\0')
                                        r = 5;
                                  else if(cBoard[4][12]!='\0')
                                        r = 4;
                                  else{
                                        playSX();
                                        return 10;
                                  }
                            }
            }
            else if(command == 'E'){
                //look for Queen in garveyard
                if(player=='A'){
                                  if(cBoard[7][1]!='\0')
                                        r = 7;
                                  else{
                                        playSX();
                                        return 10;
                                  }
                            }
                            else if(player=='B'){
                                  if(cBoard[0][12]!='\0')
                                        r = 0;
                                  else{
                                        playSX();
                                        return 10;
                                  }
                            }
            }
                }
        command = 0u;
          }
    }
CyDelay(20);

Key5_Write(1u);
CyDelay(15);
    if(Key1_Read() == 1u){
        //look for Rook in graveyard
        if(player=='A'){
                      if(cBoard[2][1]!='\0')
                            r = 2;
                      else if(cBoard[3][1]!='\0')
                            r = 3;
                      else{
                            playSX();
                            return 10;
                      }
                }
```

```
                        else if(player=='B'){
                              if(cBoard[3][12]!='\0')
                                    r = 3;
                              else if(cBoard[2][12]!='\0')
                                    r = 2;
                              else{
                                    playSX();
                                    return 10;
                              }
                        }
                CyDelay(500u);
          }
          else if(Key2_Read() == 1u){
                //look for Queen in garveyard
                if(player=='A'){
                              if(cBoard[7][1]!='\0')
                                    r = 7;
                              else{
                                    playSX();
                                    return 10;
                              }
                        }
                        else if(player=='B'){
                              if(cBoard[0][12]!='\0')
                                    r = 0;
                              else{
                                    playSX();
                                    return 10;
                              }
                        }
                CyDelay(500u);
          }
          else if(Key4_Read() == 1u){
                              UART_1_UartPutChar(CMD_RECOG_SI);
                              CyDelay(10u);
                              UART_1_UartPutChar('D');
                  LEDMIC_Write(1u);
                              CyDelay(2000u);
                  LEDMIC_Write(0u);
                 }
      Key5_Write(0u);
      Key6_Write(1u);
      CyDelay(15);
          if(Key1_Read() == 1u){
                //look for Knight in graveyard
                if(player=='A'){
                              if(cBoard[1][1]!='\0')
                                    r = 1;
                              else if(cBoard[0][1]!='\0')
                                    r = 0;
                              else{
                                    playSX();
                                    return 10;
                              }
                        }
                        else if(player=='B'){
                              if(cBoard[7][12]!='\0')
```

```
                                    r = 7;
                              else if(cBoard[6][12]!='\0')
                                    r = 6;
                              else{
                                    playSX();
                                    return 10;
                              }
                        }
                CyDelay(500u);
            }
        Key6_Write(0u);
            Key7_Write(1u);
                  CyDelay(15);
                if(Key1_Read() == 1u){
                //look for Bishop in graveyard
                if(player=='A'){
                              if(cBoard[4][1]!='\0')
                                    r = 4;
                              else if(cBoard[5][1]!='\0')
                                    r = 5;
                              else{
                                    playSX();
                                    return 10;
                              }
                        }
                        else if(player=='B'){
                              if(cBoard[5][12]!='\0')
                                    r = 5;
                              else if(cBoard[4][12]!='\0')
                                    r = 4;
                              else{
                                    playSX();
                                    return 10;
                              }
                        }
                CyDelay(500u);
                }
            Key7_Write(0u);

        if(r!=10){
            return r;
        }
      }
}

//does a promotion movement
void promMove(){
      char8 result, command = 0u;
      int stepsX, stepsY;
      int numSqsH = 0, numSqsV = 0;
    int r = 10, c = 0;
    Key5_Write(0u);
    Key6_Write(0u);
    Key7_Write(0u);

      while(r == 10){
            result = UART_1_UartGetChar();
        if(result > 0u){
            if(result == 's'){
                CyDelay(20);
                UART_1_UartPutChar(' ');
                CyDelay(20);
                while(command == 0u){
                    command = UART_1_UartGetChar();
                    if(command == 'B'){
                                    r = listenForPiece();
                    }
                        }
                command = 0u;
                  }
            }
        CyDelay(20);

        Key5_Write(1u);
        CyDelay(15);
        if(Key4_Read() == 1u){
                              UART_1_UartPutChar(CMD_RECOG_SI);
                              CyDelay(10u);
                              UART_1_UartPutChar('A');
                    LEDMIC_Write(1u);
                              CyDelay(2000u);
                    LEDMIC_Write(0u);
                        }
        Key5_Write(0u);
        CyDelay(15);
            Key7_Write(1u);
        CyDelay(15);
                if(Key4_Read() == 1u){
                    r = listenForPiece();
                    CyDelay(500u);
                }
            Key7_Write(0u);

            if(r==8){
                  playSX();
                  return;
            }
      }

      if(player=='A'){
            c = 1;
      }
      else if(player=='B'){
            c = 12;
      }

      moveToGrave(dR,dC);
    sR = r;
    sC = c;
    currentR = dR;
    currentC = dC;
      moveToSource();

      //calculate direction and do the movement
```

```
if(r>0){
    Motor2Pos_Write(0u);
      Motor2Neg_Write(1u);
}
else{
    Motor2Pos_Write(1u);
      Motor2Neg_Write(0u);
}

if(r > dR){
        numSqsV = r - dR;
        Motor2Pos_Write(0u);
        Motor2Neg_Write(1u);
  }
  else if(r < dR){
        numSqsV = dR - r;
        Motor2Pos_Write(1u);
        Motor2Neg_Write(0u);
  }

  if(c > dC){
        numSqsH = c - dC;
        Motor1Pos_Write(0u);
        Motor1Neg_Write(1u);
  }
  else if(c < dC){
        numSqsH = dC - c;
        Motor1Pos_Write(1u);
        Motor1Neg_Write(0u);
  }

CyDelay(10u);

  stepsY = (numSqsV-1)*stepsPerSq;
  stepsX = (numSqsH-1)*stepsPerSq;
Couple_Write(1u);
CyDelay(20u);
int k;
  for(k = 0; k < stepsPerSq/2; k++){
        Motor1Steps_Write(1u);
        CyDelay(width);
        Motor1Steps_Write(0u);
        CyDelay(width);
  }
CyDelay(50u);
  for(k = 0; k < stepsPerSq/2; k++){
        Motor2Steps_Write(1u);
        CyDelay(width);
        Motor2Steps_Write(0u);
        CyDelay(width);
  }
CyDelay(50u);
  for(k = 0; k < stepsX; k++){
        Motor1Steps_Write(1u);
        CyDelay(width);
        Motor1Steps_Write(0u);
        CyDelay(width);
```

```
          }
          for(k = 0; k < stepsY; k++){
                Motor2Steps_Write(1u);
                CyDelay(width);
                Motor2Steps_Write(0u);
                CyDelay(width);
          }
        CyDelay(50u);
          for(k = 0; k < stepsPerSq/2; k++){
                Motor1Steps_Write(1u);
                CyDelay(width);
                Motor1Steps_Write(0u);
                CyDelay(width);
          }
        CyDelay(50u);
          if(r!=dR){
          for(k = 0; k < stepsPerSq/2; k++){
                Motor2Steps_Write(1u);
                CyDelay(width);
                Motor2Steps_Write(0u);
                CyDelay(width);
          }
        }
        else{
            if(r>0){
                Motor2Pos_Write(1u);
              Motor2Neg_Write(0u);
                CyDelay(10u);
            }
            else{
                Motor2Pos_Write(0u);
              Motor2Neg_Write(1u);
                CyDelay(10u);
            }

            CyDelay(20u);
            for(k = 0; k < stepsPerSq/2; k++){
                Motor2Steps_Write(1u);
                CyDelay(width);
                Motor2Steps_Write(0u);
                CyDelay(width);
          }
        }
        moveBackMore(dR, dC);
        Couple_Write(0u);
        CyDelay(20u);

        //set the current place
          cBoard[dR][dC] = cBoard[sR][sC];
          cBoard[sR][sC] = 0;
        currentR = dR;
        currentC = dC;
}

//decides whose turn it is
void switchPlayer(){
        int k;
```

```
    if(player == 'A'){
        for(k=0;k<30;k++){
            Turn_Write(1u);
            CyDelayUs(700);
            Turn_Write(0u);
            CyDelayUs(19300);
        }
        LEDA_Write(0u);
        LEDB_Write(1u);
        player = 'B';
    }
    else if(player == 'B'){
        for(k=0;k<30;k++){
            Turn_Write(1u);
            CyDelayUs(1900);
            Turn_Write(0u);
            CyDelayUs(18100);
        }
        LEDA_Write(1u);
        LEDB_Write(0u);
        player = 'A';
    }
}

//find out which rook they want to castle with
int whichRook(){
      char8 result, command = 0u;
    Key5_Write(0u);
    Key6_Write(0u);
    Key7_Write(0u);

      for(;;){
            result = UART_1_UartGetChar();
        if(result > 0u){
            if(result == 's'){
                CyDelay(20);
                UART_1_UartPutChar(' ');
                CyDelay(20);
                while(command == 0u){
                    command = UART_1_UartGetChar();
                    if(command == 'B'){
                                return 3;
                    }
                              if(command == 'I'){
                                return 10;
                              }
                    else{
                        return 0;
                    }
                        }
                  }
            }

            Key5_Write(1u);
                  CyDelay(15);
                if(Key1_Read() == 1u){
                    return 3;
                }
                        else if(Key4_Read() == 1u){
                              UART_1_UartPutChar(CMD_RECOG_SI);
                              CyDelay(10u);
                              UART_1_UartPutChar('D');
                              CyDelay(2000u);
                        }
            Key5_Write(0u);
        Key6_Write(1u);
                  CyDelay(15);
                if(Key3_Read() == 1u){
                    return 10;
                }
            Key6_Write(0u);
      }
}

//does a castling movement if conditions satisfy
void doCastle(){
    int stepsX;
      int c = 0;
      int notCheck;
    int endC;

      if(player=='A'){
            if(pieceMoved[4]==1){
                  playSX();
                  return;
            }else if(pieceMoved[0]==1&&pieceMoved[1]==1){
                  playSX();
                  return;
            }

            sR = kingAR;
            sC = kingAC;
      }
      else if(player == 'B'){
            if(pieceMoved[5]==1){
                  playSX();
                  return;
            }else if(pieceMoved[2]==1&&pieceMoved[3]==1){
                  playSX();
                  return;
            }

            sR = kingBR;
            sC = kingBC;
      }

    c = whichRook();

    if(c==0){
        playSX();
        return;
    }

      if(player=='A'){
            if(c==3){
                  if(pieceMoved[0]==1){
                        playSX();
                        return;
                  }else if(cBoard[sR][sC-1][0]!='\0'||cBoard[sR][sC-
2][0]!='\0'){
                        playSX();
                        return;
                  }
                  else{
                        moveToSource();
                        Couple_Write(1u);
                        CyDelay(100u);
                        Motor1Pos_Write(0u);
                        Motor1Neg_Write(1u);
                        CyDelay(10u);

                        cBoard[0][6] = cBoard[0][3];
                        cBoard[0][3] = 0;
                        dR = 0;
                        dC = sC-2;
                        notCheck = verifyNotCheck();
                        if(notCheck == 0){
                              cBoard[0][3] = cBoard[0][6];
                              cBoard[0][6] = 0;
                              playSX();
                              return;
                        }

                        stepsX = 2*stepsPerSq;
                        int k;
                        for(k = 0; k < stepsX; k++){
                              Motor1Steps_Write(1u);
                              CyDelay(width);
                              Motor1Steps_Write(0u);
                              CyDelay(width);
                        }
                moveBackMore(dR, dC);
                        Couple_Write(0u);

                        currentR = sR;
                        currentC = sC-2;
                        cBoard[sR][dC] = cBoard[sR][sC];
                        cBoard[sR][sC] = 0;
                        sC = 3;
                        moveToSource();

                        stepsX = 3*stepsPerSq;
                endC = sC + 3;
                        Motor1Pos_Write(1u);
                        Motor1Neg_Write(0u);
                        Motor2Pos_Write(1u);
                        Motor2Neg_Write(0u);
                        Couple_Write(1u);
                        CyDelay(100u);
                        for(k = 0; k < stepsPerSq/2; k++){
                              Motor2Steps_Write(1u);
```

```
                                    CyDelay(width);
                                    Motor2Steps_Write(0u);
                                    CyDelay(width);
                              }
                              CyDelay(50u);
                              for(k = 0; k < stepsX; k++){
                                    Motor1Steps_Write(1u);
                                    CyDelay(width);
                                    Motor1Steps_Write(0u);
                                    CyDelay(width);
                              }
                              Motor2Pos_Write(0u);
                              Motor2Neg_Write(1u);
                              CyDelay(50u);
                              for(k = 0; k < stepsPerSq/2; k++){
                                    Motor2Steps_Write(1u);
                                    CyDelay(width);
                                    Motor2Steps_Write(0u);
                                    CyDelay(width);
                              }
                      moveBackMore(dR, endC);
                              Couple_Write(0u);
                        }
                  }else if(c==10){
                        if(pieceMoved[1]==1){
                              playSX();
                              return;
                        }else
      if(cBoard[sR][sC+1][0]!='\0'||cBoard[sR][sC+2][0]!='\0'){
                              playSX();
                              return;
                        }
                        else{
                              moveToSource();
                              Couple_Write(1u);
                              CyDelay(100u);
                              Motor1Pos_Write(1u);
                              Motor1Neg_Write(0u);
                              CyDelay(10u);

                              cBoard[0][8]  = cBoard[0][10];
                              cBoard[0][10]  = 0;
                              dR = 0;
                              dC = sC+2;
                              notCheck = verifyNotCheck();
                              if(notCheck == 0){
                                    cBoard[0][10]  = cBoard[0][8];
                                    cBoard[0][8]  = 0;
                                    playSX();
                                    return;
                              }

                              stepsX = 2*stepsPerSq;
                              int k;
                              for(k = 0; k < stepsX; k++){
                                    Motor1Steps_Write(1u);
                                    CyDelay(width);
                                    Motor1Steps_Write(0u);
                                    CyDelay(width);
                              }
                  moveBackMore(dR, dC);
                              Couple_Write(0u);

                              currentR = sR;
                              currentC = sC+2;
                              cBoard[sR][dC] = cBoard[sR][sC];
                              cBoard[sR][sC] = 0;
                              sC = 10;
                              moveToSource();

                              stepsX = 2*stepsPerSq;
                  endC = sC - 2;
                              Motor1Pos_Write(0u);
                              Motor1Neg_Write(1u);
                              Motor2Pos_Write(1u);
                              Motor2Neg_Write(0u);
                              Couple_Write(1u);
                              CyDelay(100u);
                              for(k = 0; k < stepsPerSq/2; k++){
                                    Motor2Steps_Write(1u);
                                    CyDelay(width);
                                    Motor2Steps_Write(0u);
                                    CyDelay(width);
                              }
                              CyDelay(50u);
                              for(k = 0; k < stepsX; k++){
                                    Motor1Steps_Write(1u);
                                    CyDelay(width);
                                    Motor1Steps_Write(0u);
                                    CyDelay(width);
                              }
                              Motor2Pos_Write(0u);
                              Motor2Neg_Write(1u);
                              CyDelay(50u);
                              for(k = 0; k < stepsPerSq/2; k++){
                                    Motor2Steps_Write(1u);
                                    CyDelay(width);
                                    Motor2Steps_Write(0u);
                                    CyDelay(width);
                              }
                  moveBackMore(dR, endC);
                              Couple_Write(0u);
                        }
                  }
            }
            else if(player == 'B'){
                  if(c==3){
                        if(pieceMoved[2]==1){
                              playSX();
                              return;
                        }else if(cBoard[sR][sC-1][0]!='\0'||cBoard[sR][sC-
2][0]!='\0'){
                              playSX();
                              return;
```

```
  }
  else{
        moveToSource();
        Couple_Write(1u);
        CyDelay(100u);
        Motor1Pos_Write(0u);
        Motor1Neg_Write(1u);
        CyDelay(10u);

        cBoard[7][6] = cBoard[7][3];
        cBoard[7][3] = 0;
        dR = 7;
        dC = sC-2;
        notCheck = verifyNotCheck();
        if(notCheck == 0){
              cBoard[7][3] = cBoard[7][6];
              cBoard[7][6] = 0;
              playSX();
              return;
        }

        stepsX = 2*stepsPerSq;
        int k;
        for(k = 0; k < stepsX; k++){
              Motor1Steps_Write(1u);
              CyDelay(width);
              Motor1Steps_Write(0u);
              CyDelay(width);
        }
moveBackMore(dR, dC);
        Couple_Write(0u);

        currentR = sR;
        currentC = sC-2;
        cBoard[sR][dC] = cBoard[sR][sC];
        cBoard[sR][sC] = 0;
        sC = 3;
        moveToSource();

        stepsX = 3*stepsPerSq;
endC = sC + 3;
        Motor1Pos_Write(1u);
        Motor1Neg_Write(0u);
        Motor2Pos_Write(0u);
        Motor2Neg_Write(1u);
        Couple_Write(1u);
        CyDelay(100u);
        for(k = 0; k < stepsPerSq/2; k++){
              Motor2Steps_Write(1u);
              CyDelay(width);
              Motor2Steps_Write(0u);
              CyDelay(width);
        }
        CyDelay(50u);
        for(k = 0; k < stepsX; k++){
              Motor1Steps_Write(1u);
              CyDelay(width);
```

```
                              Motor1Steps_Write(0u);
                              CyDelay(width);
                        }
                        Motor2Pos_Write(1u);
                        Motor2Neg_Write(0u);
                        CyDelay(50u);
                        for(k = 0; k < stepsPerSq/2; k++){
                              Motor2Steps_Write(1u);
                              CyDelay(width);
                              Motor2Steps_Write(0u);
                              CyDelay(width);
                        }
                moveBackMore(dR, endC);
                        Couple_Write(0u);
                  }
            }else if(c==10){
                  if(pieceMoved[3]==1){
                        playSX();
                        return;
                  }else
if(cBoard[sR][sC+1][0]!='\0'||cBoard[sR][sC+2][0]!='\0'){
                        playSX();
                        return;
                  }
                  else{
                        moveToSource();
                        Couple_Write(1u);
                        CyDelay(100u);
                        Motor1Pos_Write(1u);
                        Motor1Neg_Write(0u);
                        CyDelay(10u);

                        cBoard[7][8] = cBoard[7][10];
                        cBoard[7][10] = 0;
                        dR = 7;
                        dC = sC+2;
                        notCheck = verifyNotCheck();
                        if(notCheck == 0){
                              cBoard[7][10] = cBoard[7][8];
                              cBoard[7][8] = 0;
                              playSX();
                              return;
                        }

                        stepsX = 2*stepsPerSq;
                        int k;
                        for(k = 0; k < stepsX; k++){
                              Motor1Steps_Write(1u);
                              CyDelay(width);
                              Motor1Steps_Write(0u);
                              CyDelay(width);
                        }
                moveBackMore(dR, dC);
                        Couple_Write(0u);

                        currentR = sR;
                        currentC = sC+2;
                        cBoard[sR][dC] = cBoard[sR][sC];
                        cBoard[sR][sC] = 0;
                        sC = 10;
                        moveToSource();

                        stepsX = 2*stepsPerSq;
                endC = sC - 2;
                        Motor1Pos_Write(0u);
                        Motor1Neg_Write(1u);
                        Motor2Pos_Write(0u);
                        Motor2Neg_Write(1u);
                        Couple_Write(1u);
                        CyDelay(100u);
                        for(k = 0; k < stepsPerSq/2; k++){
                              Motor2Steps_Write(1u);
                              CyDelay(width);
                              Motor2Steps_Write(0u);
                              CyDelay(width);
                        }
                        CyDelay(50u);
                        for(k = 0; k < stepsX; k++){
                              Motor1Steps_Write(1u);
                              CyDelay(width);
                              Motor1Steps_Write(0u);
                              CyDelay(width);
                        }
                        Motor2Pos_Write(1u);
                        Motor2Neg_Write(0u);
                        CyDelay(50u);
                        for(k = 0; k < stepsPerSq/2; k++){
                              Motor2Steps_Write(1u);
                              CyDelay(width);
                              Motor2Steps_Write(0u);
                              CyDelay(width);
                        }
                moveBackMore(dR, endC);
                        Couple_Write(0u);
                  }
            }
      }
    currentR = dR;
    currentC = endC;
      switchPlayer();
}

void setupBoard(){
      //set the pawn locations for both players
    int i;

    for(i = 0; i<8; i++){
        cBoard[i][0] = 0;
        cBoard[i][1] = 0;
        cBoard[i][12] = 0;
        cBoard[i][13] = 0;
    }

      for(i = 3; i<11; i++){
        cBoard[2][i] = 0;
        cBoard[3][i] = 0;
        cBoard[4][i] = 0;
        cBoard[5][i] = 0;
    }

      for(i = 3; i < 11; i++){
            cBoard[1][i] = "A-P";
            cBoard[6][i] = "B-P";
      }

      //set the rook locations
      cBoard[0][3] = "A-R1";
      cBoard[0][10] = "A-R2";
      cBoard[7][3] = "B-R1";
      cBoard[7][10] = "B-R2";

      //set the knight locations
      cBoard[0][4] = "A-K";
    cBoard[0][9] = "A-K";
      cBoard[7][4] = "B-K";
    cBoard[7][9] = "B-K";

      //set the bishop locations
      cBoard[0][5] = "A-B";
    cBoard[0][8] = "A-B";
      cBoard[7][5] = "B-B";
    cBoard[7][8] = "B-B";

      //set the queen locations
      cBoard[0][6] = "A-Q";
      cBoard[7][6] = "B-Q";

      //set the king locations
      cBoard[0][7] = "A-Ki";
      kingAR = 0;
      kingAC = 7;

      cBoard[7][7] = "B-Ki";
      kingBR = 7;
      kingBC = 7;

      player = 'A';
    for(i=0;i<30;i++){
            Turn_Write(1u);
            CyDelayUs(1900);
            Turn_Write(0u);
            CyDelayUs(18100);
    }
    LEDA_Write(1u);
    LEDB_Write(0u);

    for(i=0;i<6;i++){
        pieceMoved[i] = 0;
    }
}
```

```
//moves all the pieces to their original spots and reset the board
void moveBackPieces(){
    int stepsX, stepsY;
      int numSqsH, numSqsV;
      int k, j;
    int x;
    int i;
      for(j=0;j<14;j++){
            for(k=0;k<8;k++){
                  if(cBoard[k][j][0]!='\0'){
                        sR = k;
                        sC = j;
                        if(strcmp(cBoard[sR][sC],"A-P")==0){
                              dR=1;
                              int c;
                              for(c=10;c>2;c--){
                        if(sR==1){
                            dC = sC;
                        }
                                    else
if(cBoard[dR][c][0]=='\0'||(strcmp(cBoard[dR][c],"A-P")!=0)){
                                          dC = c;
                                    }
                              }
                        }
                        else if(strcmp(cBoard[sR][sC],"B-P")==0){
                              dR=6;
                              int c;
                              for(c=10;c>2;c--){
                        if(sR==6){
                            dC = sC;
                        }
                                    else
if(cBoard[dR][c][0]=='\0'||(strcmp(cBoard[dR][c],"B-P")!=0)){
                                          dC = c;
                                    }
                              }
                        }
                        else if((strcmp(cBoard[sR][sC],"A-R1")==0)){
                              dR = 0;
                              dC = 3;
                        }
                        else if((strcmp(cBoard[sR][sC],"A-R2")==0)){
                              dR = 0;
                              dC = 10;
                        }
                        else if((strcmp(cBoard[sR][sC],"B-R1")==0)){
                              dR = 7;
                              dC = 3;
                        }
                        else if((strcmp(cBoard[sR][sC],"B-R2")==0)){
                              dR = 7;
                              dC = 10;
                        }
                        else if((strcmp(cBoard[sR][sC],"A-B")==0)){
                              dR = 0;
                              if((strcmp(cBoard[dR][5],"A-B")!=0)){
                                    dC = 5;
                              }else if((strcmp(cBoard[dR][8],"A-B")!=0)){
                                    dC = 8;
                              }else{
                                    dC = sC;
                              }
                        }
                        else if((strcmp(cBoard[sR][sC],"B-B")==0)){
                              dR = 7;
                              if((strcmp(cBoard[dR][5],"B-B")!=0)){
                                    dC = 5;
                              }else if((strcmp(cBoard[dR][8],"B-B")!=0)){
                                    dC = 8;
                              }else{
                                    dC = sC;
                              }
                        }
                        else if((strcmp(cBoard[sR][sC],"A-K")==0)){
                              dR = 0;
                              if((strcmp(cBoard[dR][4],"A-K")!=0)){
                                    dC = 4;
                              }else if((strcmp(cBoard[dR][9],"A-K")!=0)){
                                    dC = 9;
                              }else{
                                    dC = sC;
                              }
                        }
                        else if((strcmp(cBoard[sR][sC],"B-K")==0)){
                              dR = 7;
                              if((strcmp(cBoard[dR][4],"B-K")!=0)){
                                    dC = 4;
                              }else if((strcmp(cBoard[dR][9],"B-K")!=0)){
                                    dC = 9;
                              }else{
                                    dC = sC;
                              }
                        }
                        else if((strcmp(cBoard[sR][sC],"A-Q")==0)){
                              dR = 0;
                              dC = 6;
                        }
                        else if((strcmp(cBoard[sR][sC],"B-Q")==0)){
                              dR = 7;
                              dC = 6;
                        }
                        else if((strcmp(cBoard[sR][sC],"A-Ki")==0)){
                              dR = 0;
                              dC = 7;
                        }
                        else if((strcmp(cBoard[sR][sC],"B-Ki")==0)){
                              dR = 7;
                              dC = 7;
                        }

                if((sR!=dR)||(sC!=dC)){

if(cBoard[dR][dC][0]!='\0'&&(strcmp(cBoard[sR][sC],cBoard[dR][dC])!=0)){
                        sR = dR;
                        sC = dC;
                        dR = i;
                        dC = 11;
                        i++;
                        moveToSource();
                        if(i==7){
                          i = 0;
                        }
                        k--;
                      }else{
                        moveToSource();
                      }
                      if(sR>0){
                        Motor2Pos_Write(0u);
                       Motor2Neg_Write(1u);
                      }
                      else{
                        Motor2Pos_Write(1u);
                       Motor2Neg_Write(0u);
                      }
                      Motor1Pos_Write(1u);
                     Motor1Neg_Write(0u);
                      CyDelay(10u);
                        //calculate direction and do the movement
                        numSqsH = 0;
                        numSqsV = 0;
                        if(sR > dR){
                              numSqsV = sR - dR;
                              Motor2Pos_Write(0u);
                              Motor2Neg_Write(1u);
                        }
                        else if(sR < dR){
                              numSqsV = dR - sR;
                              Motor2Pos_Write(1u);
                              Motor2Neg_Write(0u);
                        }

                        if(sC > dC){
                              numSqsH = sC - dC;
                              Motor1Pos_Write(0u);
                              Motor1Neg_Write(1u);
                        }
                        else if(sC < dC){
                              numSqsH = dC - sC;
                              Motor1Pos_Write(1u);
                              Motor1Neg_Write(0u);
                        }

                        CyDelay(10u);

                        stepsY = (numSqsV-1)*stepsPerSq;
                     stepsX = (numSqsH-1)*stepsPerSq;
                      Couple_Write(1u);
                      CyDelay(20u);
                     for(x = 0; x < stepsPerSq/2; x++){
                        Motor1Steps_Write(1u);
```

```
        CyDelay(width);
        Motor1Steps_Write(0u);
        CyDelay(width);
}
  CyDelay(20u);
for(x = 0; x < stepsPerSq/2; x++){
        Motor2Steps_Write(1u);
        CyDelay(width);
        Motor2Steps_Write(0u);
        CyDelay(width);
}
  CyDelay(20u);
for(x = 0; x < stepsX; x++){
        Motor1Steps_Write(1u);
        CyDelay(width);
        Motor1Steps_Write(0u);
        CyDelay(width);
}
  CyDelay(20u);
for(x = 0; x < stepsY; x++){
        Motor2Steps_Write(1u);
        CyDelay(width);
        Motor2Steps_Write(0u);
        CyDelay(width);
}
  CyDelay(20u);
  if(sC!=dC){
        for(x = 0; x < stepsPerSq/2; x++){
                Motor1Steps_Write(1u);
                CyDelay(width);
                Motor1Steps_Write(0u);
                CyDelay(width);
        }
  }
  else{
        Motor1Pos_Write(0u);
        Motor1Neg_Write(1u);
        CyDelay(20u);
        for(x = 0; x < stepsPerSq/2; x++){
                Motor1Steps_Write(1u);
                CyDelay(width);
                Motor1Steps_Write(0u);
                CyDelay(width);
        }
  }
  CyDelay(20u);
  if(sR!=dR){
        for(x= 0; x < stepsPerSq/2; x++){
                Motor2Steps_Write(1u);
                CyDelay(width);
                Motor2Steps_Write(0u);
                CyDelay(width);
        }
  }
  else{
        if(sR>0){
            Motor2Pos_Write(1u);
```

```
                                    Motor2Neg_Write(0u);
                                    CyDelay(10u);
                                }
                                else{
                                    Motor2Pos_Write(0u);
                                    Motor2Neg_Write(1u);
                                    CyDelay(10u);
                                }

                                CyDelay(20u);
                                for(x = 0; x < stepsPerSq/2; x++){
                                      Motor2Steps_Write(1u);
                                      CyDelay(width);
                                      Motor2Steps_Write(0u);
                                      CyDelay(width);
                                }
                            }
                            moveBackMore(dR, dC);
                            Couple_Write(0u);
                            CyDelay(20u);
                            currentR = dR;
                                    currentC = dC;
                            numSqsH = 0;
                            numSqsV = 0;
                            cBoard[dR][dC]=cBoard[sR][sC];
                            cBoard[sR][sC] = 0;
                        }
                          }
                }
        }

        sR = 4;
        sC = 4;
        moveToSource();
        setupBoard();
}

//does the movement of each piece
void doMovement(int fourMoves[]){
        //grab the source and destination
        sR = fourMoves[0]-1;
        sC = fourMoves[1]+2;
        dR = fourMoves[2]-1;
        dC = fourMoves[3]+2;
        int stepsx, stepsy;
      int stepsoff = 0;
      int numSqsX = 0, numSqsY = 0;
        int k=0;

      //Set the enemy
      char oppPlayer = 'P';
      if(player == 'A'){
          oppPlayer = 'B';
      }
      else if(player == 'B'){
          oppPlayer = 'A';
      }
```

```
    //check to see if this movement will leave the king in check
      int notCheck = verifyNotCheck();
      if(notCheck == 0){
        playSX();
        LEDMIC_Write(1u);
        CyDelay(400);
        LEDMIC_Write(0u);
        CyDelay(400);
        LEDMIC_Write(1u);
        CyDelay(400);
        LEDMIC_Write(0u);
        return;
      }

    //check if the source piece exists and is the player's
      if(cBoard[sR][sC][0]!='\0'){
        if(cBoard[sR][sC][0]==player){
                moveToSource();
        }
        else{
            playSX();
            return;
        }
      }
      else{
            playSX();
            return;
      }

      //pawn movements
      if(strcmp(cBoard[sR][sC],"A-P")==0){

        //check if the pawn is capturing another piece
            if(dC!=sC){
                  if((dC==(sC+1))||(dC==(sC-1))){

      if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer))
                              moveToGrave(dR,dC);
                        else if((strcmp(cBoard[sR][dC],"B-
P")==0)&&(cBoard[sR][dC][0]==oppPlayer))
                              moveToGrave(sR,dC);
                        else{
                              playSX();
                              return;
                        }
                        stepsx = stepsPerSq;
                        stepsy = stepsPerSq;

                if(dC==sC+1){
                    Motor1Pos_Write(1u);
                    CyDelay(10u);
                        Motor1Neg_Write(0u);
                    CyDelay(10u);
                }
                else{
                    Motor1Pos_Write(0u);
```

```
                        CyDelay(10u);
                            Motor1Neg_Write(1u);
                        CyDelay(10u);
                    }
                            CyDelay(10u);
                    Motor2Pos_Write(1u);
                    CyDelay(10u);
                        Motor2Neg_Write(0u);
                    CyDelay(10u);

                    Couple_Write(1u);
                    CyDelay(20u);
                            for(k = 0; k < stepsx/2; k++){
                                  Motor1Steps_Write(1u);
                                  CyDelay(width);
                                  Motor1Steps_Write(0u);
                                  CyDelay(width);
                            }
                    CyDelay(10u);
                            for(k = 0; k < stepsy; k++){
                                  Motor2Steps_Write(1u);
                                  CyDelay(width);
                                  Motor2Steps_Write(0u);
                                  CyDelay(width);
                            }
                    CyDelay(10u);
                            for(k = 0; k < stepsx/2; k++){
                                  Motor1Steps_Write(1u);
                                  CyDelay(width);
                                  Motor1Steps_Write(0u);
                                  CyDelay(width);
                            }
                    moveBackMore(dR, dC);
                            Couple_Write(0u);
                    CyDelay(20u);
                      }
                      else{
                            playSX();
                            return;
                      }
                }
                else{                   //otherwise check if the pawn is just
moving correctly
                      if(dR==(sR+2)){
                            if(sR!=1){
                                  playSX();
                                  return;
                            }
                            else
if((cBoard[sR+1][dC][0]!='\0')||(cBoard[sR+2][dC][0]!='\0')){
                                  playSX();
                                  return;
                            }
                            stepsy = stepsPerSq*2+stepsoff;
                      }
                      else if((dR==(sR+1))&&(cBoard[sR+1][dC][0]=='\0')){
                            stepsy = stepsPerSq+stepsoff;
                      }
                      else{
                            playSX();
                            return;
                      }

                Motor2Pos_Write(1u);
                CyDelay(10u);
                    Motor2Neg_Write(0u);
                CyDelay(10u);

                Couple_Write(1u);
                CyDelay(20u);
                      for(k = 0; k < stepsy; k++){
                            Motor2Steps_Write(1u);
                            CyDelay(width);
                            Motor2Steps_Write(0u);
                            CyDelay(width);
                      }
                moveBackMore(dR, dC);
                      Couple_Write(0u);
                CyDelay(20u);
                }

                if(dR==7){
                      promMove();
                }
          }
          else if(strcmp(cBoard[sR][sC],"B-P")==0){

            //check if the pawn is capturing another piece
                if(dC!=sC){
                      if((dC==(sC+1))||(dC==(sC-1))){

          if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer))
                                  moveToGrave(dR,dC);
                            else if((strcmp(cBoard[sR][dC],"A-
    P")==0)&&(cBoard[sR][dC][0]==oppPlayer))
                                  moveToGrave(sR,dC);
                            else{
                                  playSX();
                                  return;
                            }
                            stepsx = stepsPerSq+stepsoff;
                            stepsy = stepsPerSq+stepsoff;

                    if(dC==sC+1){
                        Motor1Pos_Write(1u);
                        CyDelay(10u);
                            Motor1Neg_Write(0u);
                        CyDelay(10u);
                    }
                    else{
                        Motor1Pos_Write(0u);
                        CyDelay(10u);
                            Motor1Neg_Write(1u);
                        CyDelay(10u);
                    }
                    CyDelay(10u);

                    Motor2Pos_Write(0u);
                    CyDelay(10u);
                        Motor2Neg_Write(1u);
                    CyDelay(10u);

                    Couple_Write(1u);
                    CyDelay(20u);
                            for(k = 0; k < stepsx/2; k++){
                                  Motor1Steps_Write(1u);
                                  CyDelay(width);
                                  Motor1Steps_Write(0u);
                                  CyDelay(width);
                            }
                    CyDelay(10u);
                            for(k = 0; k < stepsy; k++){
                                  Motor2Steps_Write(1u);
                                  CyDelay(width);
                                  Motor2Steps_Write(0u);
                                  CyDelay(width);
                            }
                    CyDelay(10u);
                            for(k = 0; k < stepsx/2; k++){
                                  Motor1Steps_Write(1u);
                                  CyDelay(width);
                                  Motor1Steps_Write(0u);
                                  CyDelay(width);
                            }
                    moveBackMore(dR, dC);
                            Couple_Write(0u);
                    CyDelay(20u);
                      }
                      else{
                            playSX();
                            return;
                      }
                }
                else{                   //otherwise check if the pawn is moving
    correctly
                      if(dR==(sR-2)){
                            if(sR!=6){
                                  playSX();
                                  return;
                            }
                            else if((cBoard[sR-1][dC][0]!='\0')||(cBoard[sR-
    2][dC][0]!='\0')){
                                  playSX();
                                  return;
                            }
                            stepsy = stepsPerSq*2+stepsoff;
                      }
                      else if((dR==(sR-1))&&(cBoard[sR-1][dC][0]=='\0')){
                            stepsy = stepsPerSq+stepsoff;
                      }
                      else{
                            playSX();
                            return;
                      }

                Motor2Pos_Write(0u);
                CyDelay(10u);
                    Motor2Neg_Write(1u);
                CyDelay(10u);

                Couple_Write(1u);
                CyDelay(20u);
                      for(k = 0; k < stepsy; k++){
                            Motor2Steps_Write(1u);
                            CyDelay(width);
                            Motor2Steps_Write(0u);
                            CyDelay(width);
                      }
                moveBackMore(dR, dC);
                      Couple_Write(0u);
                CyDelay(20u);
                }

                if(dR==0){
                      promMove();
                }
          }
          else if((strcmp(cBoard[sR][sC],"A-R1")==0)||(strcmp(cBoard[sR][sC],"A-
    R2")==0)||(strcmp(cBoard[sR][sC],"B-R1")==0)||(strcmp(cBoard[sR][sC],"B-
    R2")==0)){

            //check if the rook is moving correctly
            if(dR!=sR){
                if(dC!=sC){
                    playSX();
                    return;
                }
            }

            //check the direction the rook is moving
            //also check if there are pieces in the way
            int i=0;
            if(sR>dR){
                for(i=(sR-1);i>dR;i--){
                    if(cBoard[i][dC][0]!='\0'){
                        playSX();
                        return;
                    }
                }

                if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer)){
                    moveToGrave(dR, dC);
                }
                      else if(cBoard[dR][dC][0]==player){
                            playSX();
                            return;
                      }
```

```
    numSqsY = sR-dR;
    Motor2Pos_Write(0u);
    CyDelay(10u);
        Motor2Neg_Write(1u);
    CyDelay(10u);
}
else if(dR>sR){
    for(i=(sR+1);i<dR;i++){
        if(cBoard[i][dC][0]!='\0'){
            playSX();
            return;
        }
    }

    if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer)){
        moveToGrave(dR, dC);
    }
          else if(cBoard[dR][dC][0]==player){
                playSX();
                return;
          }

    numSqsY = dR-sR;
    Motor2Pos_Write(1u);
    CyDelay(10u);
        Motor2Neg_Write(0u);
    CyDelay(10u);
}
else if(sC>dC){
    for(i=(sC-1);i>dC;i--){
        if(cBoard[dR][i][0]!='\0'){
            playSX();
            return;
        }
    }

    if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer)){
        moveToGrave(dR, dC);
    }
          else if(cBoard[dR][dC][0]==player){
                playSX();
                return;
          }

    numSqsX = sC-dC;
    Motor1Pos_Write(0u);
    CyDelay(10u);
        Motor1Neg_Write(1u);
    CyDelay(10u);
}
else if(dC>sC){
    for(i=(sC+1);i<dC;i++){
        if(cBoard[dR][i][0]!='\0'){
            playSX();
            return;
        }
    }
```

```
            if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer)){
                moveToGrave(dR, dC);
            }
                  else if(cBoard[dR][dC][0]==player){
                        playSX();
                        return;
                  }

            numSqsX = dC-sC;
            Motor1Pos_Write(1u);
            CyDelay(10u);
                Motor1Neg_Write(0u);
            CyDelay(10u);
        }
        else{
            playSX();
            return;
        }

        stepsy=numSqsY*stepsPerSq+stepsoff;
        stepsx=numSqsX*stepsPerSq+stepsoff;

        Couple_Write(1u);
        CyDelay(20u);
        for(k = 0; k < stepsx; k++){
                  Motor1Steps_Write(1u);
                  CyDelay(width);
                  Motor1Steps_Write(0u);
                  CyDelay(width);
            }
        CyDelay(10u);
            for(k = 0; k < stepsy; k++){
                  Motor2Steps_Write(1u);
                  CyDelay(width);
                  Motor2Steps_Write(0u);
                  CyDelay(width);
            }
        moveBackMore(dR, dC);
        Couple_Write(0u);
        CyDelay(20u);

        //keep track of which rooks moved for castling
        if(strcmp(cBoard[sR][sC],"A-R1")==0)
            pieceMoved[0]=1;
        else if(strcmp(cBoard[sR][sC],"A-R2")==0)
            pieceMoved[1]=1;
        else if(strcmp(cBoard[sR][sC],"B-R1")==0)
            pieceMoved[2]=1;
        else if(strcmp(cBoard[sR][sC],"B-R2")==0)
            pieceMoved[3]=1;

      }
      else if((strcmp(cBoard[sR][sC],"A-B")==0)||(strcmp(cBoard[sR][sC],"B-
B")==0)){

        //check invalid movement
        if((sR==dR)||(sC==dC)){
                playSX();
            return;
        }
        else if(abs(sR-dR)!=abs(sC-dC)){
                playSX();
            return;
        }

        //set directions and check spaces
        int j = 0, i = 0;
        if(sR>dR){
            j = sR-1;
            if(sC>dC){

                for(i = sC-1;i>dC;i--){
                    if(cBoard[j][i][0]!='\0'){
                        playSX();
                        return;
                    }
                    j--;
                }


if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer)){
                    moveToGrave(dR, dC);
                }
                        else if(cBoard[dR][dC][0]==player){
                              playSX();
                              return;
                        }

                Motor1Pos_Write(0u);
                CyDelay(10u);
                    Motor1Neg_Write(1u);
                CyDelay(10u);
            }
            else if(sC<dC){

                for(i = sC+1;i<dC;i++){
                    if(cBoard[j][i][0]!='\0'){
                        playSX();
                        return;
                    }
                    j--;
                }


if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer)){
                    moveToGrave(dR, dC);
                }
                        else if(cBoard[dR][dC][0]==player){
                              playSX();
                              return;
                        }

                Motor1Pos_Write(1u);
                CyDelay(10u);
                    Motor1Neg_Write(0u);
                CyDelay(10u);
            }
            Motor2Pos_Write(0u);
            CyDelay(10u);
                Motor2Neg_Write(1u);
            CyDelay(10u);
            numSqsX = sR-dR;
        }
        else if(sR<dR){
            j = sR+1;
            if(sC>dC){

                for(i = sC-1;i>dC;i--){
                    if(cBoard[j][i][0]!='\0'){
                        playSX();
                        return;
                    }
                    j++;
                }


if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer)){
                    moveToGrave(dR, dC);
                }
                        else if(cBoard[dR][dC][0]==player){
                              playSX();
                              return;
                        }

                Motor1Pos_Write(0u);
                CyDelay(10u);
                    Motor1Neg_Write(1u);
                CyDelay(10u);

            }
            else if(sC<dC){

                for(i = sC+1;i<dC;i++){
                    if(cBoard[j][i][0]!='\0'){
                        playSX();
                        return;
                    }
                    j++;
                }


if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer)){
                    moveToGrave(dR, dC);
                }
                        else if(cBoard[dR][dC][0]==player){
                              playSX();
                              return;
                        }

                Motor1Pos_Write(1u);
                CyDelay(10u);
                    Motor1Neg_Write(0u);
                CyDelay(10u);
            }
            Motor2Pos_Write(1u);
            CyDelay(10u);
                Motor2Neg_Write(0u);
            CyDelay(10u);
            numSqsX = dR-sR;
        }

        stepsx  = stepsPerSq*numSqsX+stepsoff;
        Couple_Write(1u);
        CyDelay(20u);
        for(k = 0; k < stepsx; k++){
                  Motor1Steps_Write(1u);
            Motor2Steps_Write(1u);
                  CyDelay(width);
                  Motor1Steps_Write(0u);
            Motor2Steps_Write(0u);
                  CyDelay(width);
            }
        moveBackMore(dR, dC);
        Couple_Write(0u);
        CyDelay(20u);
      }
      else if((strcmp(cBoard[sR][sC],"A-K")==0)||(strcmp(cBoard[sR][sC],"B-
K")==0)){

        //check movement and set directions
            if((sR-dR)==2){

            if((sC-dC)==1){
                if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer))
                    moveToGrave(dR, dC);
                        else if(cBoard[dR][dC][0]==player){
                              playSX();
                              return;
                        }

                Motor2Pos_Write(0u);
                CyDelay(10u);
                    Motor2Neg_Write(1u);
                CyDelay(10u);
                Motor1Pos_Write(0u);
                CyDelay(10u);
                    Motor1Neg_Write(1u);
                CyDelay(10u);
            }
            else if((dC-sC)==1){
                if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer))
                    moveToGrave(dR, dC);
                        else if(cBoard[dR][dC][0]==player){
                              playSX();
                              return;
                        }
                Motor2Pos_Write(0u);
                CyDelay(10u);
                    Motor2Neg_Write(1u);
                CyDelay(10u);
                Motor1Pos_Write(1u);
                CyDelay(10u);
                    Motor1Neg_Write(0u);
                CyDelay(10u);
            }
            else{
                playSX();
                return;
            }

            stepsx = stepsPerSq+stepsoff;
            stepsy = 2*stepsPerSq+stepsoff;
            Couple_Write(1u);
            CyDelay(20u);
            for(k = 0; k < stepsx/2; k++){
                      Motor1Steps_Write(1u);
                      CyDelay(width);
                      Motor1Steps_Write(0u);
                      CyDelay(width);
                }
            for(k = 0; k < stepsy; k++){
                      Motor2Steps_Write(1u);
                      CyDelay(width);
                      Motor2Steps_Write(0u);
                      CyDelay(width);
                }
            for(k = 0; k < stepsx/2; k++){
                      Motor1Steps_Write(1u);
                      CyDelay(width);
                      Motor1Steps_Write(0u);
                      CyDelay(width);
                }
            moveBackMore(dR, dC);
            Couple_Write(0u);
            CyDelay(20u);
        }
        else if((dR-sR)==2){

            if((sC-dC)==1){
                if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer))
                    moveToGrave(dR, dC);
                        else if(cBoard[dR][dC][0]==player){
                              playSX();
                              return;
                        }

                Motor2Pos_Write(1u);
                CyDelay(10u);
                    Motor2Neg_Write(0u);
                CyDelay(10u);
                Motor1Pos_Write(0u);
                CyDelay(10u);
                    Motor1Neg_Write(1u);
                CyDelay(10u);
            }
            else if((dC-sC)==1){
                if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer))
                    moveToGrave(dR, dC);
                        else if(cBoard[dR][dC][0]==player){
                              playSX();
                              return;
                        }

                Motor2Pos_Write(1u);
                CyDelay(10u);
                    Motor2Neg_Write(0u);
                CyDelay(10u);
                Motor1Pos_Write(1u);
                CyDelay(10u);
                    Motor1Neg_Write(0u);
                CyDelay(10u);
            }
            else{
                playSX();
                return;
            }

            stepsx = stepsPerSq+stepsoff;
            stepsy = 2*stepsPerSq+stepsoff;
            Couple_Write(1u);
            CyDelay(20u);
            for(k = 0; k < stepsx/2; k++){
                      Motor1Steps_Write(1u);
                      CyDelay(width);
                      Motor1Steps_Write(0u);
                      CyDelay(width);
                }
            for(k = 0; k < stepsy; k++){
                      Motor2Steps_Write(1u);
                      CyDelay(width);
                      Motor2Steps_Write(0u);
                      CyDelay(width);
                }
            for(k = 0; k < stepsx/2; k++){
                      Motor1Steps_Write(1u);
                      CyDelay(width);
                      Motor1Steps_Write(0u);
                      CyDelay(width);
                }
            moveBackMore(dR, dC);
            Couple_Write(0u);
            CyDelay(20u);
        }
        else if((sR-dR)==1){

            if((sC-dC)==2){
                if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer))
                    moveToGrave(dR, dC);
                        else if(cBoard[dR][dC][0]==player){
                              playSX();
                              return;
                        }

                Motor2Pos_Write(0u);
                CyDelay(10u);
                    Motor2Neg_Write(1u);
                CyDelay(10u);
                Motor1Pos_Write(0u);
                CyDelay(10u);
                    Motor1Neg_Write(1u);
                CyDelay(10u);
            }
            else if((dC-sC)==2){
                if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer))
                    moveToGrave(dR, dC);
                        else if(cBoard[dR][dC][0]==player){
                              playSX();
                              return;
                        }

                Motor2Pos_Write(0u);
                CyDelay(10u);
                    Motor2Neg_Write(1u);
                CyDelay(10u);
                Motor1Pos_Write(1u);
                CyDelay(10u);
                    Motor1Neg_Write(0u);
                CyDelay(10u);
            }
            else{
                playSX();
                return;
            }

            stepsx = 2*stepsPerSq+stepsoff;
            stepsy = stepsPerSq+stepsoff;
            Couple_Write(1u);
            CyDelay(20u);
            for(k = 0; k < stepsy/2; k++){
                      Motor2Steps_Write(1u);
                      CyDelay(width);
                      Motor2Steps_Write(0u);
                      CyDelay(width);
                }
            for(k = 0; k < stepsx; k++){
                      Motor1Steps_Write(1u);
                      CyDelay(width);
                      Motor1Steps_Write(0u);
                      CyDelay(width);
                }
            for(k = 0; k < stepsy/2; k++){
                      Motor2Steps_Write(1u);
                      CyDelay(width);
                      Motor2Steps_Write(0u);
                      CyDelay(width);
                }
            moveBackMore(dR, dC);
```

```
        Couple_Write(0u);
        CyDelay(20u);
    }
    else if((dR-sR)==1){

        if((sC-dC)==2){
            if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer))
                moveToGrave(dR, dC);

            Motor2Pos_Write(1u);
            CyDelay(10u);
                Motor2Neg_Write(0u);
            CyDelay(10u);
            Motor1Pos_Write(0u);
            CyDelay(10u);
                Motor1Neg_Write(1u);
            CyDelay(10u);
        }
        else if((dC-sC)==2){
            if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer))
                moveToGrave(dR, dC);

            Motor2Pos_Write(1u);
            CyDelay(10u);
                Motor2Neg_Write(0u);
            CyDelay(10u);
            Motor1Pos_Write(1u);
            CyDelay(10u);
                Motor1Neg_Write(0u);
            CyDelay(10u);
        }
        else{
            playSX();
            return;
        }

        stepsx = 2*stepsPerSq+stepsoff;
        stepsy = stepsPerSq+stepsoff;
        Couple_Write(1u);
        CyDelay(20u);
        for(k = 0; k < stepsy/2; k++){
                  Motor2Steps_Write(1u);
                  CyDelay(width);
                  Motor2Steps_Write(0u);
                  CyDelay(width);
            }
        for(k = 0; k < stepsx; k++){
                  Motor1Steps_Write(1u);
                  CyDelay(width);
                  Motor1Steps_Write(0u);
                  CyDelay(width);
            }
        for(k = 0; k < stepsy/2; k++){
                  Motor2Steps_Write(1u);
                  CyDelay(width);
                  Motor2Steps_Write(0u);
                  CyDelay(width);
            }
        moveBackMore(dR, dC);
        Couple_Write(0u);
        CyDelay(20u);
    }
    else{
        playSX();
        return;
    }

  }
  else if((strcmp(cBoard[sR][sC],"A-Q")==0)||(strcmp(cBoard[sR][sC],"B-
Q")==0)){
        stepsx = stepsPerSq*abs(sC-dC)+stepsoff;
    stepsy = stepsPerSq*abs(sR-dR)+stepsoff;
    int i = 0, j = 0;

    //check the spaces are empty
    if(sR==dR){
        i = sR;
        if(sC>dC){
            for(j=sC-1;j>dC;j--){
                if(cBoard[i][j][0]!='\0'){
                    playSX();
                    return;
                }
            }
            numSqsX = sC-dC;
        }
        else if(sC<dC){
            for(j=sC+1;j<dC;j++){
                if(cBoard[i][j][0]!='\0'){
                    playSX();
                    return;
                }
            }
            numSqsX = dC-sC;
        }
        else{
            playSX();
            return;
        }

        if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer)){
            moveToGrave(dR, dC);
        }
              else if(cBoard[dR][dC][0]==player){
                    playSX();
                    return;
              }

        //set movement directions
        if(sR>dR){
            Motor2Neg_Write(1u);
            Motor2Pos_Write(0u);
            CyDelay(10u);
        }
        else{
            Motor2Neg_Write(0u);
            Motor2Pos_Write(1u);
            CyDelay(10u);
        }
        if(sC>dC){
            Motor1Neg_Write(1u);
            Motor1Pos_Write(0u);
            CyDelay(10u);
        }
        else{
            Motor1Neg_Write(0u);
            Motor1Pos_Write(1u);
            CyDelay(10u);
        }

        stepsx=numSqsX*stepsPerSq+stepsoff;
        Couple_Write(1u);
        CyDelay(20u);
        for(k = 0; k < stepsx; k++){
                    Motor1Steps_Write(1u);
                    CyDelay(width);
                    Motor1Steps_Write(0u);
                    CyDelay(width);
            }
        moveBackMore(dR, dC);
        Couple_Write(0u);
        CyDelay(20u);
    }
    else if(sC==dC){
        j = sC;
        if(sR>dR){
            for(i=sR-1;i>dR;i--){
                if(cBoard[i][j][0]!='\0'){
                    playSX();
                    return;
                }
            }
            numSqsY = sR-dR;
        }
        else if(sR<dR){
            for(i=sR+1;i<dR;i++){
                if(cBoard[i][j][0]!='\0'){
                    playSX();
                    return;
                }
            }
            numSqsY = dR-sR;
        }

        if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer)){
            moveToGrave(dR, dC);
        }
               else if(cBoard[dR][dC][0]==player){
                    playSX();
                    return;
               }
        //set movement directions
        if(sR>dR){
            Motor2Neg_Write(1u);
            Motor2Pos_Write(0u);
            CyDelay(10u);
        }
        else{
            Motor2Neg_Write(0u);
            Motor2Pos_Write(1u);
            CyDelay(10u);
        }
        if(sC>dC){
            Motor1Neg_Write(1u);
            Motor1Pos_Write(0u);
            CyDelay(10u);
        }
        else{
            Motor1Neg_Write(0u);
            Motor1Pos_Write(1u);
            CyDelay(10u);
        }

        stepsy=numSqsY*stepsPerSq+stepsoff;
        Couple_Write(1u);
        CyDelay(20u);
        for(k = 0; k < stepsy; k++){
                  Motor2Steps_Write(1u);
                  CyDelay(width);
                  Motor2Steps_Write(0u);
                  CyDelay(width);
            }
        moveBackMore(dR, dC);
        Couple_Write(0u);
        CyDelay(20u);
    }
    else if(abs(sR-dR)==abs(sC-dC)){
        if(sR>dR){
            j = sR-1;
            if(sC>dC){
                for(i = sC-1;i>dC;i--){
                    if(cBoard[j][i][0]!='\0'){
                        playSX();
                        return;
                    }
                    j--;
                }
            }
            else if(sC<dC){
                for(i = sC+1;i<dC;i++){
                    if(cBoard[j][i][0]!='\0'){
                        playSX();
                        return;
                    }
                    j--;
                }
            }
            numSqsX = sR-dR;
        }
        else if(sR<dR){
            j = sR+1;
            if(sC>dC){
                for(i = sC-1;i>dC;i--){
                    if(cBoard[j][i][0]!='\0'){
                        playSX();
                        return;
                    }
                    j++;
                }
            }
            else if(sC<dC){
                for(i = sC+1;i<dC;i++){
                    if(cBoard[j][i][0]!='\0'){
                        playSX();
                        return;
                    }
                    j++;
                }
            }
            numSqsX = dR-sR;
        }
        stepsx  = stepsPerSq*numSqsX+stepsoff;

        if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer)){
            moveToGrave(dR, dC);
        }
              else if(cBoard[dR][dC][0]==player){
                    playSX();
                    return;
              }

        //set movement directions
        if(sR>dR){
            Motor2Neg_Write(1u);
            Motor2Pos_Write(0u);
            CyDelay(10u);
        }
        else{
            Motor2Neg_Write(0u);
            Motor2Pos_Write(1u);
            CyDelay(10u);
        }
        if(sC>dC){
            Motor1Neg_Write(1u);
            Motor1Pos_Write(0u);
            CyDelay(10u);
        }
        else{
            Motor1Neg_Write(0u);
            Motor1Pos_Write(1u);
            CyDelay(10u);
        }

        Couple_Write(1u);
```

```
            CyDelay(20u);
            for(k = 0; k < stepsx; k++){
                    Motor1Steps_Write(1u);
                Motor2Steps_Write(1u);
                    CyDelay(width);
                    Motor1Steps_Write(0u);
                Motor2Steps_Write(0u);
                    CyDelay(width);
                }
            moveBackMore(dR, dC);
            Couple_Write(0u);
            CyDelay(20u);
        }
        else{
            playSX();
            return;
        }
      }
      else if((strcmp(cBoard[sR][sC],"A-Ki")==0)||(strcmp(cBoard[sR][sC],"B-
Ki")==0)){
            if(((sR-dR)==1||(dR-sR)==1)&&((sC-dC)==1||(dC-sC)==1)){

            if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer)){
                moveToGrave(dR, dC);
            }
                  else if(cBoard[dR][dC][0]==player){
                        playSX();
                        return;
                  }

            if(sR>dR){
                Motor2Neg_Write(1u);
                Motor2Pos_Write(0u);
                CyDelay(10u);
            }
            else{
                Motor2Neg_Write(0u);
                Motor2Pos_Write(1u);
                CyDelay(10u);
            }
            if(sC>dC){
                Motor1Neg_Write(1u);
                Motor1Pos_Write(0u);
                CyDelay(10u);
            }
            else{
                Motor1Neg_Write(0u);
                Motor1Pos_Write(1u);
                CyDelay(10u);
            }

            stepsx = stepsPerSq+stepsoff;
            Couple_Write(1u);
            CyDelay(20u);
            for(k = 0; k < stepsx; k++){
                    Motor1Steps_Write(1u);
                Motor2Steps_Write(1u);
                CyDelay(width);
                Motor1Steps_Write(0u);
            Motor2Steps_Write(0u);
                CyDelay(width);
            }
        moveBackMore(dR, dC);
        Couple_Write(0u);
        CyDelay(20u);
    }
    else if((sR==dR)&&((sC-dC)==1||(dC-sC)==1)){

        if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer)){
            moveToGrave(dR, dC);
        }
              else if(cBoard[dR][dC][0]==player){
                    playSX();
                    return;
              }

        if(sC>dC){
            Motor1Neg_Write(1u);
            Motor1Pos_Write(0u);
            CyDelay(10u);
        }
        else{
            Motor1Neg_Write(0u);
            Motor1Pos_Write(1u);
            CyDelay(10u);
        }

        stepsx=stepsPerSq+stepsoff;
        Couple_Write(1u);
        CyDelay(20u);
        for(k = 0; k < stepsx; k++){
                Motor1Steps_Write(1u);
                CyDelay(width);
                Motor1Steps_Write(0u);
                CyDelay(width);
            }
        moveBackMore(dR, dC);
        Couple_Write(0u);
        CyDelay(20u);
    }
    else if((sC==dC)&&((sR-dR)==1||(dR-sR)==1)){

        if((cBoard[dR][dC][0]!='\0')&&(cBoard[dR][dC][0]==oppPlayer)){
            moveToGrave(dR, dC);
        }
              else if(cBoard[dR][dC][0]==player){
                    playSX();
                    return;
              }

        if(sR>dR){
            Motor2Neg_Write(1u);
            Motor2Pos_Write(0u);
            CyDelay(10u);
            }
            else{
                Motor2Neg_Write(0u);
                Motor2Pos_Write(1u);
                CyDelay(10u);
            }

            stepsy=stepsPerSq+stepsoff;
            Couple_Write(1u);
            CyDelay(20u);
            for(k = 0; k < stepsy; k++){
                      Motor2Steps_Write(1u);
                      CyDelay(width);
                      Motor2Steps_Write(0u);
                      CyDelay(width);
                }
            moveBackMore(dR, dC);
            Couple_Write(0u);
            CyDelay(20u);
        }
        else{
            playSX();
            return;
        }

        if(strcmp(cBoard[sR][sC],"A-Ki")==0){
            kingAC = dC;
            kingAR = dR;
                  pieceMoved[4]=1;
        }
        else if(strcmp(cBoard[sR][sC],"B-Ki")==0){
            kingBC = dC;
            kingBR = dR;
                  pieceMoved[5]=1;
        }
      }

      //CheckRes(dR, dC);
    cBoard[dR][dC] = cBoard[sR][sC];
    currentR = dR;
    currentC = dC;
    cBoard[sR][sC] = 0;
    switchPlayer();
}

void checkMate(){
    LEDA_Write(1u);
    LEDMIC_Write(1u);
    LEDB_Write(1u);
    playSX();
    CyDelay(1000);
    LEDA_Write(0u);
    LEDMIC_Write(0u);
    LEDB_Write(0u);
    CyDelay(1000);
    LEDA_Write(1u);
    LEDMIC_Write(0u);
    LEDB_Write(0u);
    playSX();
    CyDelay(1000);
    LEDA_Write(0u);
    LEDMIC_Write(1u);
    LEDB_Write(0u);
    CyDelay(1000);
    LEDA_Write(0u);
    LEDMIC_Write(0u);
    LEDB_Write(1u);
    playSX();
    CyDelay(1000);
    LEDA_Write(1u);
    LEDMIC_Write(1u);
    LEDB_Write(1u);
    CyDelay(1000);
    LEDA_Write(0u);
    LEDMIC_Write(0u);
    LEDB_Write(0u);
    CyDelay(1000);
    playSX();

    if(player == 'A'){
        LEDA_Write(1u);
        LEDB_Write(0u);
    }
    else if(player == 'B'){
        LEDA_Write(0u);
        LEDB_Write(1u);
    }
}

int main(){
    CyGlobalIntEnable;

    UART_1_Start();
      ADC1_Start();
      ADC1_StartConvert();
    PowerVR_Write(1u);
    setupBoard();
    char8 command, result;
      int movement[4];
      int i = 0;


    UART_1_UartPutChar(CMD_TIMEOUT);
    CyDelay(20u);
    UART_1_UartPutChar('C');
    CyDelay(20u);
    UART_1_UartPutChar(CMD_LANGUAGE);
    CyDelay(20u);
    UART_1_UartPutChar('A');
    CyDelay(20u);
    UART_1_UartPutChar(CMD_MIC_DIST);
    CyDelay(20u);
    UART_1_UartPutChar('@');
    CyDelay(20u);
```

```
UART_1_UartPutChar('C');
CyDelay(30u);
UART_1_UartGetChar();
UART_1_UartGetChar();
UART_1_UartGetChar();

for(;;)
{
    Key5_Write(0u);
    Key6_Write(0u);
    Key7_Write(0u);

    //Test SI command recognition
    result = UART_1_UartGetChar();
    if(result > 0u){
        if(result == 's'){
            CyDelay(20);
            UART_1_UartPutChar(' ');
            CyDelay(20);
            while(command == 0u){
                command = UART_1_UartGetChar();
                if(command == 'B'){
                                movement[i] = 1;
                                i++;
                }
                else if(command == 'C'){
                                movement[i] = 2;
                                i++;
                }
                else if(command == 'D'){
                                movement[i] = 3;
                                i++;
                }
                else if(command == 'E'){
                                movement[i] = 4;
                                i++;
                }
                else if(command == 'F'){
                                movement[i] = 5;
                                i++;
                }
                else if(command == 'G'){
                                movement[i] = 6;
                                i++;
                }
                else if(command == 'H'){
                                movement[i] = 7;
                                i++;
                }
                else if(command == 'I'){
                                movement[i] = 8;
                                i++;
                }
                else if(command == 'J'){
                    doCastle();
                }
                else if(command == 'K'){
```

```
                checkMate();
            }
            else if(command == 'A'){
                moveBackPieces();
            }
        }
        command = 0u;
    }
}
CyDelay(20);

//Code for keypad
Key5_Write(1u);
CyDelay(15);
    if(Key1_Read() == 1u){
        movement[i] = 1;
                i++;
        CyDelay(500u);
    }
    else if(Key2_Read() == 1u){
        movement[i] = 4;
                i++;
        CyDelay(500u);
    }
    else if(Key3_Read() == 1u){
        movement[i] = 7;
                i++;
        CyDelay(500u);
    }
          else if(Key4_Read() == 1u){
        UART_1_UartPutChar(CMD_RECOG_SI);
        CyDelay(10u);
        UART_1_UartPutChar('D');
        LEDMIC_Write(1u);
        CyDelay(1000u);
        LEDMIC_Write(0u);
          }
Key5_Write(0u);
Key6_Write(1u);
CyDelay(15);
    if(Key1_Read() == 1u){
        movement[i] = 2;
                i++;
        CyDelay(500u);
    }
    else if(Key2_Read() == 1u){
        movement[i] = 5;
                i++;
        CyDelay(500u);
    }
    else if(Key3_Read() == 1u){
        movement[i] = 8;
                i++;
        CyDelay(500u);
    }
          else if(Key4_Read() == 1u){
                moveBackPieces();
                CyDelay(500u);
                  }
        Key6_Write(0u);
        Key7_Write(1u);
        CyDelay(15);
            if(Key1_Read() == 1u){
                movement[i] = 3;
                        i++;
                CyDelay(500u);
            }
            else if(Key2_Read() == 1u){
                movement[i] = 6;
                        i++;
                CyDelay(500u);
            }
            else if(Key3_Read() == 1u){
                doCastle();
                CyDelay(500u);
            }
            else if(Key4_Read() == 1u){
                checkMate();
                CyDelay(500u);
            }
        Key7_Write(0u);
        CyDelay(20);
        if(i == 4){
                  i = 0;
                  doMovement(movement);
            }
    }
}
```