|  |
| --- |
|  |

|  |
| --- |
| University of Central Florida |



|  |
| --- |
| CyberChess |
|  |
| Senior Design 1 Documentation |
| **Group 33: Louis Mason, Ryan Rivas, Scott Frazier, Steffen Sutton** |
| **12/6/2012** |

Contents

[1 - Executive Summary 1](#_Toc342568527)

[2 - Project Description 2](#_Toc342568528)

[2.1 - Basic Functions 2](#_Toc342568529)

[2.2 - Motivation 4](#_Toc342568530)

[2.3 - Technical Objectives 5](#_Toc342568531)

[2.4 - Requirements and Specifications 5](#_Toc342568532)

[3 - Research and Investigations 8](#_Toc342568533)

[3.1 - Existing Similar Projects and Products 8](#_Toc342568534)

[3.2 - Microprocessor vs. Microcontroller 10](#_Toc342568535)

[3.2.1 - Microcontrollers 10](#_Toc342568536)

[3.2.2 - Microprocessor 11](#_Toc342568537)

[3.2.3 - Bootloader 11](#_Toc342568538)

[3.2.4 - Optimization of Boot Performance 14](#_Toc342568539)

[3.2.5 - Power On Sequence 14](#_Toc342568540)

[3.2.6 - Power Off Sequence 15](#_Toc342568541)

[3.2.7 - Pin Multiplexing Control 15](#_Toc342568542)

[3.3 - Digital Signal Processing 17](#_Toc342568543)

[3.3.1 - DAC and ADC 17](#_Toc342568544)

[3.3.2 - Universal Parallel Port (uPP) 17](#_Toc342568545)

[3.3.3 - The Use of a Digital Signal Processor 18](#_Toc342568546)

[3.4 - L.E.D. Driver and Lighting 18](#_Toc342568547)

[3.5 - Memory 19](#_Toc342568548)

[3.5.1 - Our Choice: Secure Digital 19](#_Toc342568549)

[3.5.2 - Alternatives Not Used 19](#_Toc342568550)

[3.5.3 - SD/MMC Controller 20](#_Toc342568551)

[3.5.4 - RAM 22](#_Toc342568552)

[3.5.5 - RAM (DDR) SETUP 22](#_Toc342568553)

[3.5.6 - Memory (RAM) Placement 24](#_Toc342568554)

[3.6 - Development Board 26](#_Toc342568555)

[3.7 - Magnets and Motors 29](#_Toc342568556)

[3.8 - Sensing 32](#_Toc342568557)

[3.9 - The Chess Board (Physical) 33](#_Toc342568558)

[3.10 - Materials 34](#_Toc342568559)

[3.10.1 - The Enclosure 34](#_Toc342568560)

[3.10.2 - The Board 34](#_Toc342568561)

[3.10.3 - The Chess Pieces 35](#_Toc342568562)

[3.11 - Power Supply 35](#_Toc342568563)

[3.12 - Printed Circuit Board 36](#_Toc342568564)

[3.13 - Operating System 40](#_Toc342568565)

[3.13.1 - Arch Linux ARM 40](#_Toc342568566)

[3.13.2 - Debian ARM 41](#_Toc342568567)

[3.13.3 - Ubuntu ARM 41](#_Toc342568568)

[3.13.4 - Gentoo 42](#_Toc342568569)

[3.13.5 - Android 42](#_Toc342568570)

[3.14 - Shells 43](#_Toc342568571)

[3.14.1 - Bourne Shell 43](#_Toc342568572)

[3.14.2 - Bourne Again Shell (Bash) 43](#_Toc342568573)

[3.14.3 - Z Shell (zsh) 43](#_Toc342568574)

[3.15 - Speech Recognition 44](#_Toc342568575)

[3.15.1 - How Speech Recognition Works 44](#_Toc342568576)

[3.15.2 - Problems with PocketSphinx 45](#_Toc342568577)

[3.15.3 - Further Speech Recognition Research 46](#_Toc342568578)

[3.15.4 - Sphinx4 47](#_Toc342568579)

[3.15.5 - eSpeak 47](#_Toc342568580)

[3.15.6 - Android 47](#_Toc342568581)

[3.15.7 - Windows 48](#_Toc342568582)

[3.16 - Board Representation (Logical) 48](#_Toc342568583)

[3.10.1 - Language 48](#_Toc342568584)

[3.10.2 - Board Data Structure 51](#_Toc342568585)

[3.10.3 - Algorithms 52](#_Toc342568586)

[4 - CyberChess Hardware 57](#_Toc342568587)

[4.1 - ARM9 AM1808 Processor 57](#_Toc342568588)

[4.1.1 - Accessing GPIO 58](#_Toc342568589)

[4.1.2 - PLL 59](#_Toc342568590)

[4.1.3 - UPP 59](#_Toc342568591)

[4.2 - OLinuXino iMX233 Development Board 64](#_Toc342568592)

[4.3 - TLV320ADC3101 ADC 66](#_Toc342568593)

[4.4 - PCM1753 DAC 68](#_Toc342568594)

[4.5 - ROB10846 Stepper Motor 70](#_Toc342568595)

[4.6 - DRV8834 Stepper Motor Driver 71](#_Toc342568596)

[4.7 - ROB09064 Servo Motor 74](#_Toc342568597)

[4.8 - COM-10866 RGB L.E.D.s 75](#_Toc342568598)

[4.9 - TLC5930 L.E.D. Driver 76](#_Toc342568599)

[4.10 - COM-08642 Reed Switches 77](#_Toc342568600)

[4.11 - Memory (SD Card | RAM) 77](#_Toc342568601)

[4.12 - FE0202 Panel Display 77](#_Toc342568602)

[4.13 - ICM7211M LCD Panel Driver 78](#_Toc342568603)

[4.13.1 - Powering the Colon 79](#_Toc342568604)

[4.14 - Power Supply/Voltage Regulation 79](#_Toc342568605)

[4.15 - Other Hardware 80](#_Toc342568606)

[4.15.1 - M74HC238 Decoder 80](#_Toc342568607)

[4.15.2 - PJ-380 Phone Jack 82](#_Toc342568608)

[4.15.3 - TL082 Op-Amp 82](#_Toc342568609)

[4.15.4 - SPK110 Hobby Speaker 82](#_Toc342568610)

[5 - CyberChess Software 83](#_Toc342568611)

[5.1 - Operating System 83](#_Toc342568612)

[5.2 - Chess Engine 83](#_Toc342568613)

[5.2.1 - Data Structures 84](#_Toc342568614)

[5.2.2 - Game Saving and Loading 85](#_Toc342568615)

[5.2.3 - Timer 86](#_Toc342568616)

[5.3 - Board, Movement and Sound Engine (BMSE) 87](#_Toc342568617)

[5.4 - Speech Recognition Engine 88](#_Toc342568618)

[6 - Design Summary 90](#_Toc342568619)

[6.1 - Hardware 90](#_Toc342568620)

[6.1.1 - Movement System 90](#_Toc342568621)

[6.1.2 -  Sensor System 92](#_Toc342568622)

[6.1.3 - Sound System 93](#_Toc342568623)

[6.1.4 - Lighting system 94](#_Toc342568624)

[6.2 - Software 95](#_Toc342568625)

[6.2.1 - Chess Engine 96](#_Toc342568626)

[6.2.2 - Board, Movement, and Sound Engine (BSME) 97](#_Toc342568627)

[6.2.3 - Speech Recognition Engine 98](#_Toc342568628)

[7 - Prototype, Construction, and Coding 100](#_Toc342568629)

[7.1 - Parts Acquisition and Bill of Materials 100](#_Toc342568630)

[7.1.1 Bill OfMaterials 101](#_Toc342568631)

[7.2 - PlexiGlass Chess Board Construction 103](#_Toc342568632)

[7.3 - Encasing Construction 104](#_Toc342568633)

[7.4 - PCB Vendor and Assembly 105](#_Toc342568634)

[7.5 - Final Coding Plan 106](#_Toc342568635)

[7.5.1 - Chess Engine 106](#_Toc342568636)

[7.5.2 - Board and Movement Engine 106](#_Toc342568637)

[8 - Implementing the Design 108](#_Toc342568638)

[8.1 - Prototype Plan 108](#_Toc342568639)

[8.2 - Build Plan 108](#_Toc342568640)

[8.3 - Test Plan 108](#_Toc342568641)

[8.3.1 - Software 109](#_Toc342568642)

[8.3.2 - Hardware 111](#_Toc342568643)

[8.4 - Extending the Design 113](#_Toc342568644)

[8.4.1 - LCD Display/Video Out 113](#_Toc342568645)

[8.4.2 - Wifi 113](#_Toc342568646)

[8.4.3 - Camera 113](#_Toc342568647)

[8.4.4 - AI 114](#_Toc342568648)

[8.4.5 - Multiple Game Saves 114](#_Toc342568649)

[9 - Facilities and Equipment 115](#_Toc342568650)

[9.1 - IDE and SDK 115](#_Toc342568651)

[9.2 - Development Board 116](#_Toc342568652)

[9.3 - Chess Board 117](#_Toc342568653)

[10 - Administrative Content 118](#_Toc342568654)

[10.1 - Milestones 118](#_Toc342568655)

[10.2 Budget and Finance Information 120](#_Toc342568656)

[10.3 - Summary and Conclusion 120](#_Toc342568657)

[11 - Appendices I](#_Toc342568658)

[11.1 - Works Cited I](#_Toc342568659)

[11.2 - Copyright Permissions IV](#_Toc342568660)

# 1 - Executive Summary

CyberChess is an autonomous, voice-controlled chess board designed to allow two players to enjoy an aesthetically appealing game of chess entirely through spoken commands. Each player will use a wired headset to issue these commands, allowing them to have the system set up a game, execute moves, and display possible moves for any piece on the board. CyberChess will make use of a magnetic XY-plotter to physically move the pieces as needed. An array of L.E.D.s will provide visual notifications to the players throughout the game, supplemented by audio to communicate ideas directly to the user. An embedded Linux system will run on the ARM-based hardware, utilizing PocketSphinx to accept spoken commands and reacting to them with appropriate movement, lighting, and audio output.

CyberChess comes from a love of technology and the desire to expand into new technological terrain. No single element of this project could be considered individually revolutionary, but the combination creates a unique take on the oft-addressed idea of automated chess play. The chance to gain experience with speech recognition, embedded Linux environments, ARM-based hardware, and with a sizable project all provide opportunities to supplement the knowledge and education of those involved.

The physical, aesthetic design of CyberChess will attempt to mimic that of actual chess boards, seeking to create a sense of familiarity for the players rather than requiring them to deal with an entirely new environment. All lighting effects will be designed to intuitively provide information and not detract from the user experience - as an obvious example, upon a player’s request to view the possible moves of a piece the relevant spots will be lit, providing simple access to the knowledge requested. Audio will be similarly created to directly communicate with the user and provide a generally pleasing gameplay experience.

The system to be implemented for movement of pieces is derived from those employed by similar projects that predate CyberChess. With the long history of mechanical engineering, true ingenuity in addressing such a problem would require extreme time investments and would likely rely on an unique idea already having been generated. Lacking these resources, CyberChess does not seek to reinvent the wheel, but instead to make effective use of it in the creation of a larger system. The software driving this will make use of an open source voice recognition system called PocketSphinx and custom software designed specifically for the project to implement the rules of the game and the corresponding hardware control.

# 2 - Project Description

## 2.1 - Basic Functions

The final CyberChess system will allow two users to play through an entire game of chess with no more physical interaction than powering on the system and wearing the headset used to issue spoken commands. At startup, the L.E.D. array will display an aesthetically pleasing sequence of light effects, welcoming the players to the game. In the event that a game has been saved to the system from a previous session, the program will prompt the users to choose between returning to the saved game, in which case all settings will reflect those chosen for that game, and starting a new game.

When a new game has been requested, CyberChess will go through a short list of settings, requesting user input on each. The players will jointly decide on the use of a chess clock - they may choose to have timing in place for individual moves, individual players, for the entire game, or not make use of the chess clock at all. Players may also be given the option to adjust aesthetics, including adjustment of volume and which colors are used in lighting the board. Upon conclusion of selecting these settings, the game will begin, indicated by a short light sequence.

Throughout the game, the current player will dictate their move into a microphone using the standard format of “[Piece Initial][Current Square] to [Destination Square]” (e.g. “RB1 to C3” for Rook at B1 to C3”). Each time a player requests a move, the internal chess engine will determine the validity and legality of the move. Should the player have requested a move that is either illegal or invalid, the problem will be communicated through the speaker system, audibly informing the player that their move cannot be made. This notification may be accompanied by use of the L.E.D.s to visually show the problem. The player may ensure that they will request a valid and legal move by asking for all possible moves for a piece using phrasing along the lines of “Show possible moves for [Piece Initial][Current Square]” (e.g. “Show possible moves for QD3”). The engine will then generate a list of moves and use this to light up the squares to which the piece can move.

Various aspects of gameplay need not be directly commanded by the players. When a piece is captured, the captured piece will be magnetically removed to a spot on the side of the board that has been designated for that particular piece. When a pawn is promoted, the program will prompt the user to choose which piece to replace it with, and the chosen piece will be magnetically moved into place. If the player attempts to promote the pawn to a piece that is still in play, the program will leave the pawn in place and treat it as if it were the replacement piece. This discrepancy may be corrected if the appropriate piece becomes available later in play. Castling will be requested by having the king move to its destination location, e.g. “KE1 to G1” to have the white king castle kingside. When a player’s king is put in check, the program will say “Check!” through the speakers, and the L.E.D.s will display an accompanying light pattern.

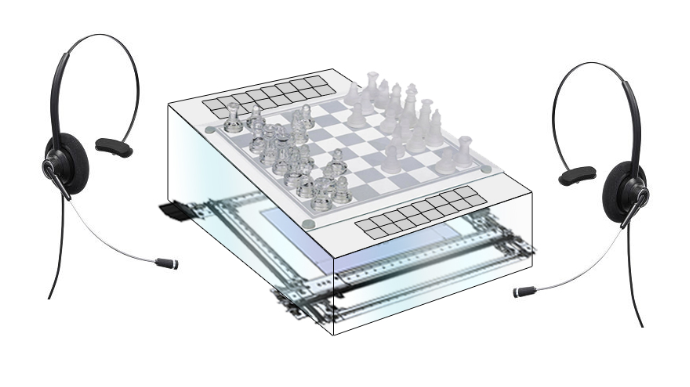
Game conclusion takes place in the normal means available in any standard chess game. If a checkmate occurs, CyberChess will display an end-of-game lighting sequence and audibly inform the players of the outcome, acknowledging the winner. Similarly, if a draw occurs due to stalemate or the presence of insufficient resources to checkmate, the game will indicate the cause of the draw and display a corresponding lighting sequence. When the opportunity to offer a draw, due either to triplication of a move or there having been fifty moves with no pawn movement or piece capture, is available but not required, the players will be informed and asked if they would like to declare a draw. Additionally, either player may offer a draw at any time, which will end the game if the other player accepts. In any of these instances, the board will display the appropriate lighting sequence and gameplay will terminate. Finally, either player may choose to resign at any time and, after receiving confirmation that the player does really want to resign, the game will end and the winner will be indicated.

Players may also request that the game be paused at any time. After receiving confirmation that their opponent agrees to pause the game, the system will save the current game state to memory and shut down, leaving all pieces in their current positions to allow gameplay to resume quickly. When the system is turned back on, it will simply ask whether the players wish to continue the paused game and continue accordingly. Should players desire to save the game for later, they may at any point say “Save game,” and the current game state will be saved. This can be used to have a sort of checkpoint to go back to after continuing with one line of play, or to allow the system to return the board to the saved state at a later point after other games. Current design specifications indicate that only one game may be saved to be recalled at a time, for the sake of simplifying the interactions involved in loading an old game. At any point a player may request a new game, and, if their opponent acquiesces, the CyberChess will move all pieces to their starting position and begin a new game. As long as the previous game was saved, it can be returned to at any point.

A set of reed switches will be meticulously placed beneath the board to constantly check on the physical status of the board.  These sensors will send information to the main program about where pieces are physically located on the board.  If a piece is in a square that the main program shows is unoccupied, CyberChess will halt the game and ask the players to place the pieces in the proper positions.  Likewise, the same halt will occur if a piece is missing from a spot that the main program shows should be occupied. This measure is in place to ensure that the system can continue working fully autonomously - the game, being designed to work without human interaction, expects there to be no external interference with the location of pieces.

## 2.2 - Motivation

We wanted to create a project that stimulated our personal interests and combines the aspects of engineering that we enjoy the most (programming, electromechanics, critical thinking, etc.). We have discussed themes to base this project around which we think will be catchy and will help sell the design to others. There have been many senior design projects that go after chess automation, but we have yet to discover a chess board that allows two people to play on the same board with full automation, let alone to do that with your voice. A scene from one of the popular *Harry Potter* movies sparked the initial idea of including speech recognition in the project. Speech recognition in our project allows the game of chess to be played without physically touching the pieces, which could bring about a welcome change for those who are physically disabled.



**Figure 2.1 - A RoughDigital Sketch of CyberChess**

The aspect of speech recognition in our project allows us to justify the use of a fast ARM9 microprocessor and an embedded Linux environment as opposed to using a microcontroller. Having experience in ARM and Embedded Linux will be a nice resume boost which helped influence our decision to design the project this way. Alternatively the team could have gone a simpler route, using a microcontroller to control all of the hardware with a laptop nearby for speech recognition. While this project would have sufficed and met all of the requirements a Senior Design project must have, the team felt as true engineers this would be a cop out.

The L.E.D. Lighting Array we will integrate into the board was motivated and inspired by a discarded senior design project idea called LoudLight, which was a box that you could link with your phone via bluetooth to receive visual notifications. These visual notifications will be translated into CyberChess, providing visual feedback to players on turns, moves, and game actions.

## 2.3 - Technical Objectives

When CyberChess is finished, it will be run only by its internal controller.  No laptop, PC, or any other similar product will be needed to run CyberChess. Our plan is to build from the component level up, and program using embedded Linux environment. The laptop or desktop used must also be compatible with Pocketsphinx voice recognition. We believe the best way to build CyberChess is with a microprocessor that will run the speech recognition program we need.  The CyberChess program will be created on a development board.  And when we have tested the program to its full extent we will load it onto a custom PCB designed by us, with all the memory, I/Os, D/A and A/D converters, and other needed parts soldered into it.  The XY-plotter will consist of a chassis with vex racks and sliders built onto it.  The movement of the magnet will be controlled by stepper motors (to move in two dimensions) and a servo motor (to raise/lower the magnet to grab/release the chess pieces).

The board itself will be large enough to give the magnetic pieces plenty of buffer space between each other.  And it will be made of two layers of plexiglass. The top layer will have the chessboard pattern and the pieces will sit on it.  Between the top plexiglass and the lower plexiglass, the sensors and L.E.D.s will be placed.  And below the bottom plexiglass, the XY-plotter will be placed.  More detailed descriptions of the hardware will be explained later.  In the next chapter, we will discuss the research that goes into finding the right hardware and software to make CyberChess come to life.

## 2.4 - Requirements and Specifications

CyberChess is an autonomous chess set designed for two human players.  There are many challenging ideas that we would like to implement into CyberChess, but at the very least the board must be speech activated and must have the capability to move chess pieces to the desired locations without the assistance of a human hand.  We also plan on writing a Game Rules Engine which the CyberChess program will use to detect illegal moves called by the users.  A set of sensors should be used to detect human interference with the game (i.e., if a player moves a manually moves a piece on the board).  An array of L.E.D.s will make up our lighting system and will light up the corners of all the squares to give the game some cool effects.  And LCD screens will be integrated into the system as optional chess clocks as well.  An audio system will be integrated into CyberChess as well.

So how do we make the chess pieces move by themselves?  There are many ideas out there, some of which may be discussed later in the research.  However, we plan on using an XY-plotter equipped with a magnet to magnetically grab the chess pieces and move them.  The chess pieces will have metal built into them (or possibly magnets) so the XY magnet can grab them.  We will go into further detail about the XY-plotter when we discuss hardware.

Table 2.4.1 below summarizes the requirements of CyberChess.

|  |  |
| --- | --- |
| 1. | The game shall provide all game feedback via audio and lighting |
| 2. | The game shall allow any move to be completed without touching a game piece |
| 3. | The game shall have sensors to detect the game pieces location |
| 4. | The game shall set up the chess board automatically |
| 5. | The game shall display possible moves if a user asks for them |
| 6. | The game shall display a visual and audible countdown for modes with timers |
| 7. | The movement system and hardware shall be visible from the top of the game |
| 8. | The game shall be powered by a standard wall outlet |
| 9. | The game shall have a specific startup sequence for lights and audio |
| 10. | The game shall have a specific shutdown sequence for lights and audio |
| 11. | The game shall notify a user of an illegal move through lights and audio |

**Table 2.4.1 - Requirements List**

In addition to the various design elements stated as required above, the team has created a list of design specifications set as goals to aim for in development. This will encourage constant progress and improvement upon the design as we seek to not only meet these specifications, but surpass them to the best of our abilities.

Table 2.4.2 below summarizes the design specifications of CyberChess.

|  |  |
| --- | --- |
| Be Able to Complete Any Move in : | **10 Seconds** |
| Be Able to Set up Board from Start in: | **5 minutes** |
| Be Able to Save and Load: | **1 Game** |
| Final Product may weigh no more than: | **10 lbs** |
| The sound system must be audible from at least: | **2 feet** |
| Microphone cords must be at least: | **2 feet** |
| Final Product may be no larger than: | **3x3 feet** |
| Automatically turn off after (prolonged inactivity) | **5 minutes** |

**Table 2.4.2 - Specifications List**

# 3 - Research and Investigations

In order to build a good design plan, research must be done to ensure that our money is spent wisely and that we understand the tasks at hand prior to executing them. The following sections discuss some existing projects that inspired our design plans as well as the many options we reviewed to optimize our budget and make CyberChess unique. While numerous aspects of CyberChess are derived from observations of the projects listed below, each element was thoroughly investigated to attempt to supplement past successes with any creative thoughts of the team.

## 3.1 - Existing Similar Projects and Products

The first step in our research on how to build a fully automated chess game was to find other instances where similar projects were successfully constructed. The plan is to use these existing projects as references for good general design schemes, while making specific details of our project unique. The core hardware that is most important to the basic functionality of the project is the system that will physically move  the pieces, and various ideas to solve this were sought.

One approach to an automated chess system was found in “Gambit: A Robust Chess-Playing Robotic System” The Gambit is a robotic chess system that allows the user to play chess against a computer. Gambit is not fully autonomous as it requires the human player to move their own chess pieces. A 6 DoF robotic arm is mounted on the opposite side of the user and is responsible for controlling the movement of the computers pieces. The sensor system for Gambit consist of a camera installed onto the end of the mechanical arm that can capture the entire playing board environment. A similar feature that CyberChess shares with Gambit is the capability of resetting the chess board after a match but needs initial setup of the game of the first game played. Unlike CyberChess, Gambit is capable of playing against another robot opponent using any arbitrary chess set.(C. Matuszek, B. Mayton, R. Aimi, M.P. Deisenroth, L. Bo, R. Chu, M. Kung, L. LeGrand, J.R. Smith, and D. Fox)

We found a very descriptive document titled “How to Build an Arduino Powered Chess Playing Robot”, by Max Justicz.  The document describes all of the parts, assembly, and coding that went into his autonomous chess board.  Unlike the human vs. human plan for our chessboard, his board was designed for a human player versus a robot.  The human player manually makes a move, and the embedded system determines the move the human made and autonomously makes a move in return. The design consisted of an XY-plotter with movement controlled by stepper motors.  The grabbing of the chess pieces is done using a neodymium magnet attached to a servomotor.  An Arduino Uno MCU was responsible for driving the motors, and an Arduino Mega was responsible for determining the move made by the human, via reed switches, and using the information to determine its next move.

An automated chess set was documented by Brett Rankin, Paul Conboy, Samantha Lickteig, and Stephen Bryant; titled “Interactive Automated Chess Set”.  Their design utilizes a mechanical crane to lift the chess pieces from above and move them in that way.  This design is driven in a similar manner as the xy-plotter described by Max Justicz in “How to Build a Chess Playing Robot”.  However, the force used to hold the chess pieces is not magnetic.  The Interactive Automated Chess Set is built of plexiglass, and an L.E.D. system that lights up individual chess squares to show players potential moves.  A player chooses their moves by pushing buttons corresponding to the row/column where they desire to move their piece. (Rankin, Conboy, Lickteig, Bryant)

Once it was decided that an XY-plotter would be used to control the motion of CyberChess, multiple variations of how to design the XY-plotter were considered. One method researched in designing the XY-plotter was to create a belt system using pulleys to move the magnet base of the plotter instead of the linear motion gears the vex rack supplies. To do this we would attach a stepper motor at one end of both drawer tracks of the x axis and instead of using a circular gear use pulleys placed on both ends of the track that a belt can be placed upon. One of the pulleys would be attached to the stepper motor and this would give us our linear motion.

The problem with this method lies in attaching the y axis to the belt. This would be unstable, inefficient, and would mainly affect the distance of the magnet to the sensor board. This changes the distance between magnet to magnet which should remain a constant throughout the build due to the strength of attraction of the magnets being the main source of motion. If the board is shaken and the belt becomes loose for any reason, the y axis might be able to move up which could cause the larger magnet to attract neighboring pieces that shouldn’t be moved, or the y axis could sag a little and might cause the magnets not to connect at all which would lead to a piece being unable to move. Both flaws would compromise the game. (bdeakyne)

In our CyberChess project, we plan to implement an XY-plotter because it is the only concept which we could find that has been successful at creating the illusion that the chess pieces are moving themselves.  We discussed other ideas for movement systems that involved more electronics, more motors, and focused on assigning magnets and motors to individual chess pieces.  All in all, from a design standpoint, these other ideas were far more complex than the XY-plotter with little to no performance advantages.  And the extra costs were especially not worth the trade-offs.  We have also decided to design our XY-plotter similar to the one described in “How to Build an Arduino Powered Chess Playing Robot”, using drawer slides to stand in as linear actuators and providing the linear motion as opposed to using pulleys like the belt system.

The Excalibur Electronic Phantom Force Electronic Chess Set is one of the only products on the market that offers automated chess but only for the computer, essentially creating a real world version of a computer chess game. Players can either play against the computer or watch the computer play itself. The board talks in English, French or Spanish while it is playing you and also sets up itself. Figure 3.1.1 below shows off the small and compact design of the Excalibur Phantom Force Electronic Chess Set as well as the input method and LCD display.



**Figure 3.1.1 - Excalibur Phantom Force Electronic Chess Set**

***Permission Pending***

On the software side of things, a number of open source implementations of chess engines are readily available online. A number of Python chess games are hosted on PyGame, a website dedicated to games written in Python. Investigating the source code of these revealed that there are indeed a wide variety of ways to approach creating a chess engine, with the two most popular PyGame programs varying quite a bit throughout. The majority of available code was aimed at creating Artificial Intelligence systems for gameplay, and thus focus was more heavily placed on move generation and position analysis than is strictly necessary for CyberChess, which needs stronger support for player interaction.

## 3.2 - Microprocessor vs. Microcontroller

### 3.2.1 - Microcontrollers

Microcontrollers are low-powered embedded computers that are usually dedicated to specific purposes. These single integrated circuits often come with ROM for the program to reside in, RAM for variables, multiple purpose I/O, and a microprocessor for the CPU. While most Senior Design projects would function just fine using a microcontroller, CyberChess’s requirements lean towards the use of a microprocessor. Most microcontrollers come with anywhere between 4KB and 512KB of flash memory which would not even be enough to store the PocketSphinx libraries. Additionally, microcontrollers operate at a much lower frequency and thus have a lot less processing power which could create problems and delays when processing speech commands.

In the initial phases of the research we were fixated on creating an embedded Linux system and once research began on microcontrollers, we began to research how we could run Linux on a microcontroller. This led us to uClinux, an embedded Linux/microcontroller project that focuses on porting Linux to systems without a Memory Management Unit (MMU). While this seemed like an excellent solution to our problem of running Linux on a microcontroller we quickly learned running Linux with such limited specifications would come at a cost. uClinux while having support for multitasking, has a few limitations. uClinux for example does not automatically grow stack and has no brk() method. All memory allocations must be done using mmap(), which is actually done by most modern code anyways. uClinux also does not implement fork(), but instead uses vfork(). This means that the parent blocks any other code from running until the child does exec() or exit(). While this means multitasking is still possible, it definitely means we would have to be more careful in coding. For these reasons it was decided to move to a microprocessor and use an ARM Linux distribution.

### 3.2.2 - Microprocessor

In deciding where the brains of CyberChess would actually come from, it was immediately determined that we would need more processing power and memory than a microcontroller could provide in order to properly support speech recognition. To allow for speech recognition we are aiming to create a platform similar to that of a mid-level Smartphone from two years ago. This means we need the processing power of about 200-600 MHz, which lead us to look at ARM based processors, namely the ARM9. Texas Instruments sells a handful of different ARM9 based Microprocessors, all of which vary very slightly in features from LCD support to the amount of USB's or possible MMC/SD slots. We decided to go with the AM1808 Sitara ARM Microprocessor because free samples were available and it fits our requirements on clock speed, USB, SD, and LCD support.

### 3.2.3 - Bootloader

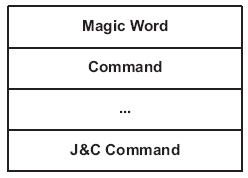
The AM1808 Sitara ARM Microprocessor makes use of the AM18xx bootloader read-only memory (ROM) image. The technical document provided by Texas Instruments gives a breakdown of the Application Image Script (AIS) boot process, an AISgen tool used to generate boot scripts, protocol for booting from an external master device, a UART Boot Host GUI for booting from a host PC, as well as any limitations, default settings, and assumptions made by the bootloader. (Coombs, Joseph)

This bootloader is key to the success of CyberChess because of what it does. A bootloader is responsible for at least configuring the memory system, loading the kernel image at the correct memory address, initializing the RAM, initializing the boot parameters to pass to the kernel, obtaining the ARM Linux machine type, and entering the kernel with the appropriate register values.

To first check that the AM18xx bootloader is compatible with our device, we must check our version using Code Composer Studio. By connecting the device and viewing the memory location 0xFFFD0000, one can see which ROM version they are running. Currently the AM18xx bootloader supports versions d800k008, d800k002, d008k004, and d800k006.

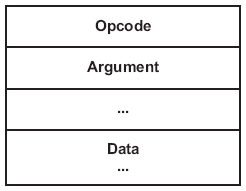
The bootloader supports booting from many embedded memory devices in master mode as well as many external devices using slave mode. A majority of the boot modes, excluding host port interface (HPI) and two of the three NOR-boot modes, use Application Image Script for the boot process. Using AIS for booting provides the developer with a unified interface even when using the AISgen tool.

Application Image Script or AIS is a format type for storing the boot image. AIS is a binary language accessed in 4-byte words in little-endian format. A magic word (0x41504954) starts of the AIS, followed by a series of AIS commands executed in sequential manner, and ends with a jump and close command. The structure of AIS is shown below in Figure 3.2.1.



**Fig 3.2.1 - Application Image Script (AIS) Structure**

Between the magic word and the J&C command lies the AIS commands. Each command consists of an operation code, optionally followed by one or more arguments, followed by optional data. This structure is shown below in Figure 3.2.2.



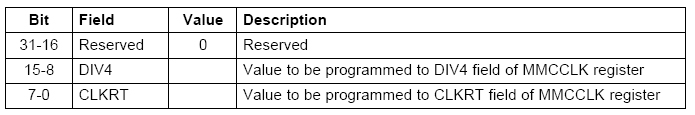
**Fig 3.2.2 - ASI Command Structure**

The operation code and its optional arguments are each one 4-byte wordwide. Should the command not fit in an even multiple of 4 bytes, it will be padded with 0’s until it is of appropriate size.

MMC/SD boot mode is done by transferring the AIS boot image to the user data area of the memory device, which in this case will be an SD card. The bootloader attempts to find the AIS image within the SD card, and upon finding it, it then looks for the magic word (the same as before). If the magic word is not found, the bootloader increments by 0x200 and searches again. This process repeats until the bootloader has searched the first 2MB of the memory card. The bootloader usually tries to check for an SD card first but this can be avoided if an MMC card is being used. To make sure the bootloader is searching for an SD card, boot pin BOOT[5] must be set low. Setting this pin high skips the SD search and goes straight to MMC. The boot pins are latched by the bootloader when the device exists reset, or the rising edge of reset. The MMC/SD register is shown below in Figure 3.2.3 and described in table 3.2.1. (Coombs, Joseph)

https://lh4.googleusercontent.com/9WmaY6QdVmlsQU0ejhaI1-sjwRrJfbmiN4CFaB4hQo2zPGX4GlXVqRqcWcHag8oXjq92EgJsApQIdkifG64WT903wLasbICXu5Hs9tk8p7ltzx_Cu2Q4

**Fig 3.2.3 - SD/MMC Register**



**Table 3.2.1 - SD/MMC Register Breakdown**

### 3.2.4 - Optimization of Boot Performance

Customization of the bootloader and Linux kernel is required to decrease the boot time associated with the cold boot performed on CyberChess when initially powered on. The optimization will focus on two areas of the boot up, size and speed. Both the bootloader and kernel come in a default configuration capable of handling most additional hardware components and protocol. A lot of these components will be unnecessary for CyberChess and will add time to the boot process through initialization. Stripping unneeded components reduces the size of the source code and eliminates steps up to the boot of the kernel leading to better performance of the cold boot time.

Another method of increasing boot performance involves modules or components necessary to the system that are called for initialization early in the booting process. There is the option to defer the loading of a module until after the system is booted if the module is taking a large amount of time to load. This is a simple task that can be done with a little alteration of the source code and a command from the user console.

There are two other methods that can help increase performance but are only recommended after all prototyping is finished due to lack of feedback to the administrator of the system. The first is removing the debug interface from the bootloader and kernel, as it will not be needed for the finished product. The last option is silencing the console and system by removing all unnecessary print statements. These techniques will allow a general bootloader and linux kernel be modified into a more product specific kernel and bootloader for CyberChess.

### 3.2.5 - Power On Sequence

The AM1808 should be powered on in the following order:

1. RTC(RTC\_CVDD) - This may be powered prior to all other supplies being applied or can be powered at the same time as CVDD. In the case where the RTC is not in use, RTC\_CVDD should be connected to CVDD. If CVDD is powered, RTC\_CVDD must not be unpowered.
2. The core logic supplies should be powered on next including both all variable 1.2V - 1.0V core logic supplies (CVDD) and all static core logic supplies (RVDD, PLL0\_VDDA, PLL1\_VDDA, USB\_CVDD, SATA\_VDD). Since we are not planning on using voltage scaling, both of these core logic supplies can be powered up together.
3. Now all static 1.8V IO supplies including DVDD18, DDR\_DVDD18, USB0\_VDDA18, USB1\_VDDA18 and SATA\_VDDR must be powered on
4. Finally all analog 3.3V PHY supplies invluding USB0\_VDDA33 and USB1\_VDDA33. We currently do not have plans to make use of the USB connectivity so we can ignore these.

For the above power sequence there is no specific required voltage ramp rate so long as LVCMOS supplies operated at 3.3V never exceed the STATIC 1.8V supplies (step 3) by more than 2 volts. (AM1808 dattasheet)

### 3.2.6 - Power Off Sequence

As just previously stated, as long as LVCMOS supplies operated at 3.3V

(DVDD3318\_A, DVDD3318\_B, or DVDD3318\_C) never exceed static 1.8V supplies by more than 2 volts then the power supplies can be powered-off in any order.

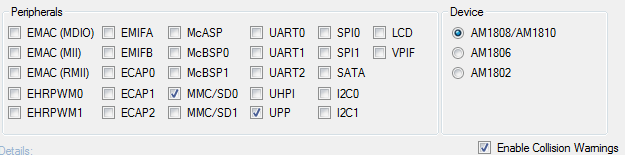
### 3.2.7 - Pin Multiplexing Control

Pin multiplexing is controlled by registers PINMUX0 - PINMUX19 in the SYSCFG module. For the AM18XX family, multiplexing can be done on a pin-by-pin basis, and is controlled by a 4-bit field in one of the PINMUX registers. The values determine which peripheral pin functions controls the pin’s IO buffer data as well as the output enable values. Default values for nearly every pin result in selecting none of the functions available, which leaves the pin’s IO buffer held tri-stated. Table 3.2.2 below shows the PINMUX values for our specific application, making use of the MMC/SD0 and UPP controllers. These values were determined using the AM18XX Pin Setup Utility from Texas Instruments

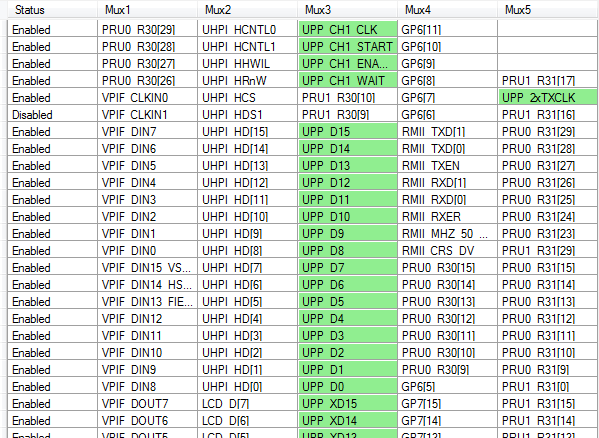
|  |  |
| --- | --- |
| **PIN** | **Value** |
| PINMUX0-9 | - |
| PINMUX10 | 0x22222222 |
| PINMUX11 | 0x00000022 |
| PINMUX12 | - |
| PINMUX13 | 0x44440000 |
| PINMUX14 | 0x44444400 |
| PINMUX15 | 0x44444444 |
| PINMUX16 | 0x44444444 |
| PINMUX17 | 0x44444444 |
| **PIN** | **Value** |
| PINMUX18 | 0x00444444 |

**Table 3.2.2 - Pin Multiplexing Register Values from Pin Setup Utility**

The Pin Setup utility allows us to select which features and hardware controllers we are using. Figure 3.2.4 below shows the selection options and Figure 3.2.5 shows their application to the pins.



**Fig. 3.2.4 - Pin Setup Peripheral Options and Device Selection**



**Fig. 3.2.5 - Pin Setup Active Pins**

## 3.3 - Digital Signal Processing

One of the challenges presented by working with audio (input and output) is the necessity of being able to put the audio input into a form whichthe processor can handle it as digital binary bits. Converting analog audio signals into digital bits is done using analog to digital converters, and reversed using digital to analog converters.  This section discusses how we plan to set up our hardware and use our processor to manipulate this data.

### 3.3.1 - DAC and ADC

Our primary reason for needing analog-to-digital and digital-to-analog converters is for input and output audio.  We do not need a stereo system, however we do need two input channels for microphones (one for each player).  Since our audio input will be strictly voice, we need an analog-to-digital converter that with vocal friendly sampling capabilities.  The human voice can range from 60 to 7000 Hz, so our analog-to-digital converter must sample at least 14 kbps, according to the Nyquist criterion.

We have decided to go with the PCM1753 DAC and the TLV320 ADC from Texas Instruments because they are inexpensive (and TI had free samples) and their sampling capabilities are more than enough to support the audio processes in CyberChess. Other options were available, but they mostly provided higher sampling rates and were thus more expensive. It was determined that these will be connected to our ARM processor via the Universal Parallel Port (uPP). Data conversion and other DSP related chips can be performance optimized using uPP, by allowing the CPU clock to adjust to a more suitable time for the converter.  Some converters have a wide range of options for time constraints otherwise.

### 3.3.2 - Universal Parallel Port (uPP)

This port allows two independent channels at speeds up to 75MHz and is designed to interface with converters with up to 16-bit data width (per channel). uPP also features dedicated data lines, minimal control lines, and can be interconnected with field-programmable gate arrays (FPGAs) or other uPP devices. It can operate in both send and receive mode or in duplex mode, where individual channels operate in opposite directions. Included inside the uPP device is an internal Direct Memory Access (DMA) controller. This DMA controller allows for maximum throughput with minimized CPU overhead during high-speed transmission.

### 3.3.3 - The Use of a Digital Signal Processor

A Digital Signal Processor (DSP) is a microprocessor specifically designed to process data in real time. Its design means it contains specialized functions for digital signals so it has the ability to process the complex and mathematically intensive calculations much more efficiently than a regular microprocessor. While a regular microprocessor is still able to do digital signal processing, its lack of specialization results fails to make the most efficient use of processing power. As such, using a plain microprocessor for digital signal processing will require it to work much harder than a DSP would. This means more heat output, more energy expenditure, less processing power available for other operations, and possibly a delay in output. These problems are product breaking in cases like a cell phone, which are expected to maintain a small form factor and have usage requirements. However, in the case of CyberChess, none of these factors come into play. (Radio-Electronics)

The large form factor of our design means heat dissipation will not be a problem, even if our processor has to work extra hard on the digital signals. Unlike a cell phone, our product is not processing loads of data beyond the digital audio signals from the microphones, so dedicating a majority of our processor to DSP will not be a problem. In the case of a cell phone, a user may decide to browse the internet while using their phone, which makes the use of a DSP all the more necessary. Delays in a cell phone conversation would be incredibly annoying to users, which makes the use of a DSP necessary. In our case, a few seconds of processing between a command and game action is not product breaking. (Texas Instruments)

While we have ordered sample Digital Signal Processors, because of these reasons the team has decided one will not be necessary. We may discover though that using an OMAP (DSP) processor may be necessary due to delays in audio processing or a need in reduction of overhead, which won't be a problem because the AM1808 can scale up to OMAP-L1x to integrate digital signal processing on a separate digital signal processor.

## 3.4 - L.E.D. Driver and Lighting

To light the board we initially wanted an effect only possible with EL Wire. Upon further investigation we found EL Wire was difficult to work with and hard to provide proper currents/voltages to, in addition to the fact that we would have to wire every square on the board individually if we wanted to light them independently. Because of these problems we decided an L.E.D. array would suffice, but we would need a driver. From Texas Instruments we found the TLC5930 12 Channel L.E.D. Driver, which supports four RGB’s. The 12 channel chip was chosen because we found it provided a good cost/channel ratio and had free samples available.

## 3.5 - Memory

Memory inside CyberChess will be used to store all of our code, operating system, and speech recognition libraries. Our memory choices include embedded solutions like NAND/NOR Flash Memory or removal options such as Secure Digital (SD) or a USB Flash Drive.

### 3.5.1 - Our Choice: Secure Digital

Secure Digital memory cards make use of NAND Flash Memory, but are obviously in card form. SD cards are available in three capacities, each with a variety of speed options marked by a Speed Class symbol. The three capacities, SD, SDHC, and SDXC allow developers to choose the correct amount of memory for their specific use. Since SDXC cards come in capacities of 32GB to 2TB we will not be discussing them as options because they provide memory that far exceeds our needs. Instead we will focus on standard SD and SDHC. The SD Standard allows for cards up to 2GB while SDHC ranges from 2GB to 32GB. For our needs, a 1-2GB SD card would suffice but due to falling prices of SDHC cards, we could have extra space for a minor increase in cost. This extra space could be used for higher quality audio, more powerful operating system, or be used to facilitate more advanced speech recognition. (SD Association)

### 3.5.2 - Alternatives Not Used

1. **NAND/NOR Flash Memory**

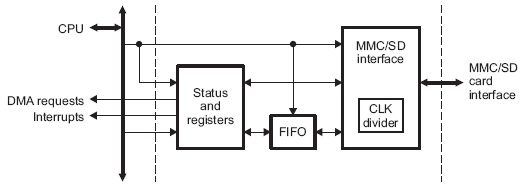
NAND and NOR Flash Memory are an embeddable memory solution for CyberChess. They two specifications, NAND and NOR, refer to the types of logic gates being used internally. NAND Flash Memory has a significantly higher capacity than NOR but NOR Flash Memory is faster, and thus more expensive. A device may choose to use both NAND and NOR memory because of their specific benefits. For example, a netbook may use embedded NOR for the operating system and a removeable NAND card for additional memory and storage. While embedded memory would create a faster environment, modifying our software would require a wired connection like serial or USB to be added to our PCB. In addition to this, our development board uses a Secure Digital Card for all of its memory, so mimicking this setup will allow for easy transition between the development board and our PCB. (Rouse, Margaret)

1. **Flash Drive**

A flash drive is essentially the third form of NAND Flash Memory that we have available to us. This option is stored in an enclosure with a USB connection, and is thus limited by the transfer speeds of USB, around 480 Mbit/s. Use of a USB Flash Drive would also require us to add a USB connector to our PCB. With other solutions available that do not require such physical modifications and offer sufficient transfer rates, this is not practical.

### 3.5.3 - SD/MMC Controller

The AM1808 processor comes with a MMC/SD card controller which will be used to interface with our secure digital memory card. The controller also features a programmable frequency of the clock that controls the timing of transfers between the SD controller and memory card and 512-bit read/write FIFO to lower system overhead. The card controller allows use of either the CPU or EDMA (external direct memory access) to write and/or read to and from the card by accessing the registers in the card controller. On one side the controller transfers data between the CPU and the EDMA controller and the MMC/SD card on the other. This separation allows the use of either the CPU or EDMA and is shown below in Figure 3.5.1. (AM1808 Technical Reference Manual)

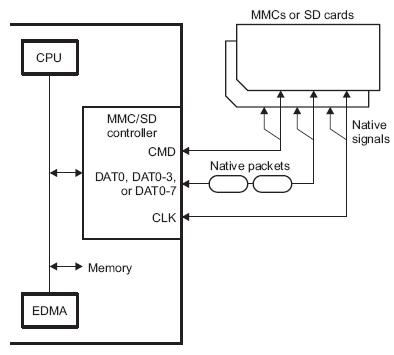


**Fig 3.5.1** - **AM1808 SD/MMC Card Controller**

In terms of industry standard compliance, the card controller supports the following industry standards:

* Multimedia Card (MMC) Specification v4.0
* Secure Digital (SD) Physical layer Specification v1.1
* Secure Digital Input Output (SDIO) Specification v2.0

The card controller communicates using the MMC/SD protocol and contains support for use of one or more MMC/SD cards, which are selected by using identification broadcast on the data line. The figure and table below summarize the interface.



**Fig 3.5.2 - MMC/SD Card Controller Interface**

|  |  |
| --- | --- |
| **Pin** | **Use** |
| MMCSD\_CMD | This pin is used to provide communication between the connected card and the MMC/SD controller. The commands are transmitted to the card using this pin and the memory card drives response to the commands on this pin. |
| MMCSD\_DAT0, MMCSD\_DAT0-3, or MMCSD\_DAT0-7 | MMC Cards only use one data line (DAT0), four data lines (DAT0-3) or eight data lines (DAT0-7A). SD Cards use one data line (DAT0), or four data lines (DAT0-3). The number of pins being used is set by the WIDTH bit in the MMC control register (MMCCTL). |
| **Pin** | **Use** |
| MMCSD\_CLK | This pin provides the clock to the memory card from the MMC/SD controller. |

**Table 3.5.1 - Pin descriptions for MMC/SD Card Controller Interface**

### 3.5.4 - RAM

Random Access Memory can be either Dynamic or Static memory. Dynamic Random Access Memory (DRAM) provides a structural advantage over SRAM because it requires only one transistor and a capacitor per bit, as opposed to SRAM which can take four to six. This advantage allows DRAM to reach high densities. Unlike both types of flash memory previously discussed, RAM is volatile, meaning it loses its data when power is lost. Our development board contains only 64MB of RAM and since this may cause problems we have decided it would be wise to add at least 128MB to our final PCB.

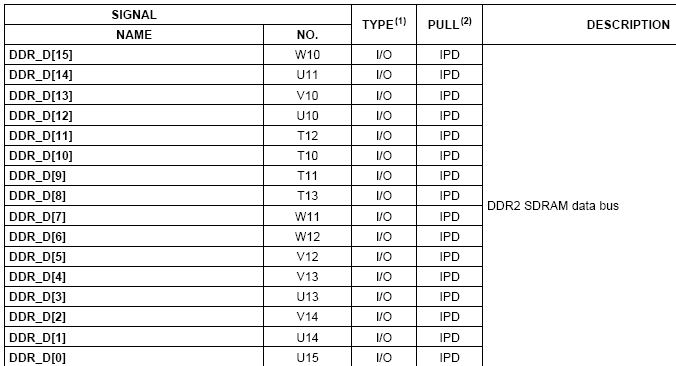
### 3.5.5 - RAM (DDR) SETUP

While the team has not decided whether we will need random access memory, DDR memory is configurable from within ASIgen when the configure DDR checkbox is selected. This configuration allows a developer to set the DDR controller registers. ASIgen supports both mDDR and DDR2 memory, with certain registers specifically for mDDR. The registers and settings are shown below in Figure 3.2.4. (Coombs, Joseph)



**Fig 3.5.3 - ASIgen DDR Settings**

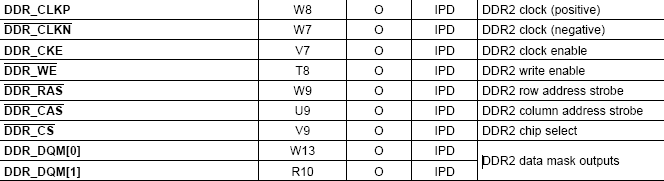
The DDR2 Controller contains a total of 39 pins, 16 of which are used for the data bus, 14 for the row/column address, and the remaining 9 for the clock, write enable, row address and column address strobe, chip select, and data mask outputs. This information is summarized below in tables 3.5.2, 3.5.3, and 3.5.4 respectively (Page 23). (AM1808 datasheet)



**Table 3.5.2 - DDR2 SDRAM Data Bus Pins**



**Table 3.5.3 - DDR2 Row/Column Address Pins**



**Table 3.5.4  - DDR2 Additional Pins**

Typically speaking, high-speed design flow is a very complex and time-intensive endeavor. Designing the PCB involves many iterative simulations which can cause problems due to inaccurate models, bugs in tools, use of incorrect environmental conditions, and errors in processing the large amounts of data involved in the simulation. A solution to this problem is to avoid simulations completely. By making the correct choices in the system specification, these timing specifications can be communicated without simulation models or timing numbers.

Design rules for the DDR/mDDR interface constrain PCB trace length, flight time delay and trace skew, signal integrity and impedance matching, cross-talk, and signal timing. In order to properly create a reliable memory system these rules must be followed. (Shust and Cobb)

*Trace, Flight Delay and Skew*

Flight Delay and Skew involve the placement of the components. A value called maximum placement refers to the maximum distance between devices permitted. Controlling this value can limit the maximum trace delay to about the longest Manhattan distance of the signals inside the clock domain. As all shorter nets are lengthened to skew match the longest one, this value is guaranteed to be the longest distance. Flight time delay, flight time delay skew, and the maximum trace lengths are thus factors of the maximum placement. (Shust and Cobb)

*Signal Integrity and Impedance Matching*

Signal integrity involves overshoot, ring back, and transition edges. With a constrained length it is possible to control signal integrity by controlling the trace topology of the different segments of an interface. The impedance of the PCB traces must be controlled and is governed by the trace width as well as the thickness and dielectric constant of the insulating material. Luckily for us, this is usually left to the PCB fabricator. (Shust and Cobb)

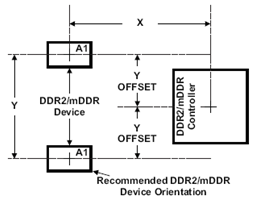
*Crosstalk*

Crosstalk is dependent on the PCB stackup and minimum trace spacing. While good crosstalk simulation can be difficult, a good solution involves high-quality signal return paths and the spreading of the signal traces. Every routing layer should have an adjacent full ground plane to allow for the shortest return current path.

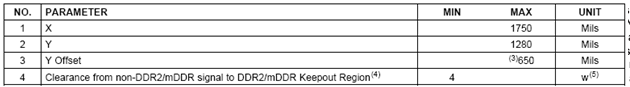
### 3.5.6 - Memory (RAM) Placement

Proper placement of the RAM memory module limits the maximum trace lengths and allows for proper routing space. For our purposes, we will be using only one memory device so the second DDR/mDDR device may be omitted from the images below. (AM1808 datasheet)

Figure 3.5.4shows the required placement for both the memory controller – or CPU in this case – and the memory devices themselves. Table 3.5.2 defines the dimensions.

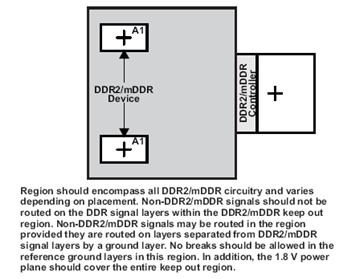


**Fig 3.5.4 - Device and DDR2/mDDR Device Placement**



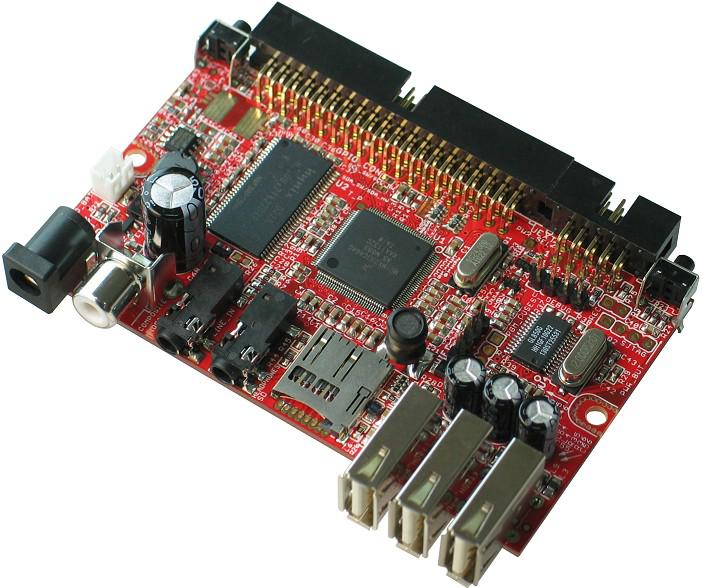
**Table 3.5.2 - Placement Specifications and Dimensions**

Additionally, the DDR/mDDR region on the PCB must be isolated from other signals. A keep out region is thus defined to serve this purpose. The size of the keep out region depends on the placement and DDR routing. The keep out region and its specifications are shown below in Figure 3.5.5. (AM1808 datasheet)



**Fig 3.5.5 - DDR2/mDDR Keep Out Region**

## 3.6 - Development Board



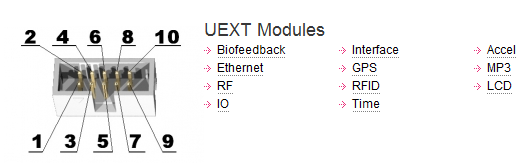
**Fig. 3.6.1 - iMX233-OLinuXino Development Board**

***Permission Pending***

The team deliberated between a few development boards and ended up going with the OLinuXino iMX233 development board because of its Audio I/O, ARM9 processor and SD Card support. The OLinuXino iMX233 runs a ARM926J processor at 454 Mhz and carries 64 MB of RAM. The main reason for choosing this dev board was the compatability it had with both an ARM9 processor and linux kernel. In order to set up the development board in proper optimal conditions it is required that it has the following:

* + - SD card with Linux Image
    - 5V power source with a max amperage of 1A

The board also has an audio codec that provides us an DAC with 99dB SNR and ADC with 85dB SNR.



**Fig. 3.6.2 - iMX233 UEXT Modules**

***Permission Pending***

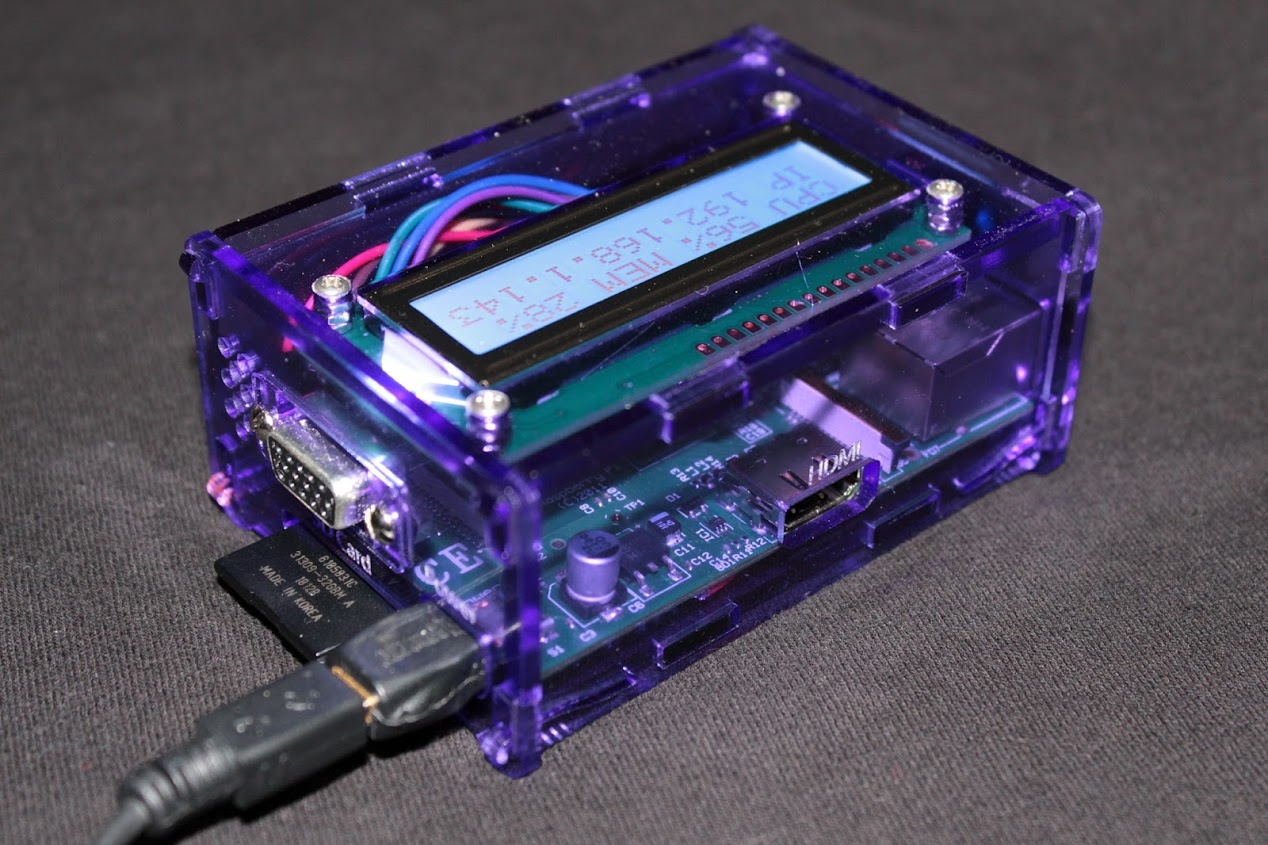
*Alternatives Not Used:*

**BeagleBoard**

The BeagleBoard is an open source single-board computer and can be purchased for $125. It is capable of running the Linux operating system desired for CyberChess and comes with 256MB of flash memory as well as 256MB of RAM. This board uses the 600 MHz Cortex-A8 microprocessor which provides more processing power than what was needed for the purpose of this project. The main reason the BeagleBoard was not considered an option was the high price of the platform compared to the hardware it had to offer on its stock board. The Raspberry Pi is another platform that for a quarter of the price allows you to achieve the same processing power as the BeagleBoard.

**Raspberry Pi**

A credit card sized microprocessor based single-board computer that is capable of desktop like processing abilities. The Raspberry Pi is as basic of a platform as it gets. It was designed specifically for educational expansion and it does not come with any additional hardware components, what you get out of the box is all you have to work with. This single-board computer drives a 700MHz ARM processor comfortably at 5V and contains up to 512KB of RAM. The Pi also meets the requirement for CyberChess’s environment being compatible with a Linux OS.



**Figure 6.3.3 – A Raspberry Pi in a custom case**

***Permission Pending***

For $35 you are able to purchase a Raspberry Pi and have access to audio, video, USB, and Ethernet. Some of these add-ons can cost twice as much for other platforms making this an attractive choice for the development board. Limited production resulted in difficulty with obtaining a Raspberry Pi for development, but the idea of its use was strongly supported, if only for the novelty of getting to use the latest and greatest in the world of embedded Linux.

**Arduino Uno**

The Arduino Uno is a open source single-board microcontroller that focuses on being user friendly. The Uno contains an 8 bit 16 MHz Atmel AVR microcontroller with 2 KB of RAM and requires a 5V power supply with very few mW. The Arduino platforms also have a wide selection of additional I/O and interface configurations to choose from. The Uno without any additional features goes for $25. With only 14 I/O pins on the Uno, additional I/O  would be needed for CyberChess leading to the purchase of a $25 multiplexer shield compatible with Arduino. This doubles the cost of purchasing the Uno, a negative for this option. Another would be that Arduino has to be programmed in an Arduino sketch in order to be compiled. Even though the Arduino sketch is similar to the C programming language, it would require the user to learn a new programming language. This takes up time and resources that can be used more efficiently in other areas of designing CyberChess. The Arduino platform combined with the Voice Control Module would have been a sufficient development environment, but it does not mimic our final environment and design specification.

## 3.7 - Magnets and Motors

Magnets come in two forms, electromagnets and permanent magnets.  Although both magnets have their advantages and disadvantages for other applications, we have decided that a permanent magnet would be a better choice to use for CyberChess.

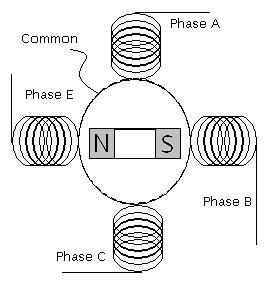
Electromagnets are basically extra big inductors wrapped around a solenoid.  They require a power source to drive a current through its wiring and create a magnetic field, the same way a transformer creates a magnetic field to drive a step-down or step-up current in a power line.  Electromagnets are very expensive to buy pre-made, and usually end up being too bulky when homemade--because it takes a lot of wrappings to produce a sufficient magnetic field for our purposes.  Because permanent magnets are typically much smaller and stronger, don’t require extra power to operate, and are perfectly capable of holding and carrying our chess pieces as flawlessly as possible; permanent magnets will be embedded in the CyberChess chess pieces and used in the XY-plotter to grab the pieces.  Small, weak magnets will be embedded in the chess pieces, while a stronger, neodymium magnet will be lifted by a servo motor to grab attract the magnetic chess pieces.

The other important options taken into consideration in the XY-plotter are what types of mechanical driving forces could be used to create linear motion.  Because CyberChess is a relatively small electrical device, motor forces such as hydraulics or gas power were not even considered.  Stepper and servo motors are small, electrically controlled, and strong enough to operate sufficiently in CyberChess.  Every similar existing project we have researched that implements anxy-plotter uses the rotational motion of a stepper and/or servo motor to run the show.  We will be using stepper motors to control the location of the magnet in two dimensions, parallel to the chessboard.  We will use a servo motor to control the position of the magnet in the third dimension.  More details about how this is accomplished are in the design chapter.

There are two main wiring schemes of stepper motors that we researched, bipolar and unipolar.  The unipolar stepper motor often results in a lower holding torque at low speeds.  However, at higher speeds, the torque of a unipolar stepper motor holds up fairly well.  A bipolar stepper motor, on the other hand, may have great torque at low speeds and much lower torque at its higher speeds.  For CyberChess, we need the best of both worlds, so a type of stepper motor called a full coil bipolar stepper motor would be the optimal choice for our design.  The trade-off to using such a well rounded stepper motor is that it requires more current, and thus more power to drive it.

Another thing to consider when looking for stepper motors is how smoothly it will work. It is imperative in CyberChess that the stepper motors we use move smoothly, because erratic motions from the main magnet could cause the chess piece to move of course.  For this reason, we have to consider the step modes of a stepper motor. There are a number of types of stepping: full step, half step, and microstepping are the main ones important to the project.

Figure 3.7.1 shows the internal diagram for a stepper motor.  By sending current through the wirings in each phase, a magnetic field is created which controls the position of the magnet in the middle.  In full step mode, a current is sent through one phase at a time, pulling the magnet four times for every revolution.  This kind of stepping uses little power and allows the magnet time to slow down before being pulled again.  This will ultimately cause the gear attached to the motor to move rather erratic.



**Figure 3.7.1 - Basic Stepper Motor Diagram**

***Permission Pending***

Half stepping is where one phase goes on by itself, causing the magnet to rotate; then the adjacent phase turns on while the first phase is still on.  The first phase eventually turns off, leaving the adjacent phase on by itself before the process continues with the next adjacent phase.  This creates a pull on the magnet eight times during every revolution, therefore the force on the magnet is steadier during the revolution than in full step mode.  (Lazaridis, Giorgios)

Tables 3.7.1 and 3.7.2 below show the order of events for these half and full step modes, respectively.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Event** | **A** | **B** | **C** | **E** | **Angle** |
| 1 | 1 | 0 | 0 | 0 | 0 deg |
| 2 | 1 | 1 | 0 | 0 | 45 |
| 3 | 0 | 1 | 0 | 0 | 90 |
| 4 | 0 | 1 | 1 | 0 | 135 |
| 5 | 0 | 0 | 1 | 0 | 180 |
| 6 | 0 | 0 | 1 | 1 | 225 |
| 7 | 0 | 0 | 0 | 1 | 270 |
| 8 | 1 | 0 | 0 | 1 | 315 |

**Table 3.7.1 - Half Step Mode**

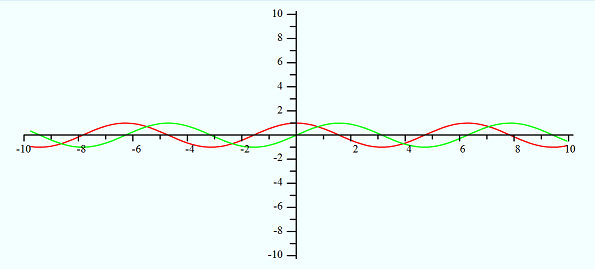
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Event** | **A** | **B** | **C** | **E** | **Magnet Angle** |
| 1 | 1 | 0 | 0 | 0 | 0 deg |
| 2 | 0 | 1 | 0 | 0 | 90 |
| 3 | 0 | 0 | 1 | 0 | 180 |
| 4 | 0 | 0 | 0 | 1 | 270 |

**Table 3.7.2 - Full Step Mode**

Another even smoother type of stepping is called microstepping.  In microstepping, the current fed to each phase is broken into different levels as well.  While one pole of the magnet passes between two phases, the current through the phase that it recently passed will incrementally weaken, while the current into the phase the pole is approaching will incrementally increase. (Lazaridis, Giorgios)

Stepper drivers with indexing capabilities can microstep at 16 to 256 steps per 90 degree angle change in the motor.  A 1/32 microstepping index would control a stepper motor with about 238 different levels of current sent through the windings during a revolution.  Assuming the steps occur frequently and at a steady pace, the force imposed on the internal magnet should be very constant, and thus the motion of the magnet should be very smooth.

The signals sent to the stepper motor almost trace out a sine and cosine wave (Figure 3.7.2).  The out-of-phase, sinusoidal inputs cause the north and south poles of the internal magnet to constantly see the same magnitude and direction of the induced magnetic field, which in turn causes the internal magnet to continually (and steadily) experience the same repulsing force from the field.  To create an analog sine wave, a DAC would have to be implemented between the processor and the motor.  Also, complex circuitry would have to convert the signal sent by the processor into a waveform that could make the stepper run at different speeds (depending on the information from the processor).  Instead of trying to design a circuit that controls the signal, we could simply look into stepper motor drivers that will microstep at a small enough rate to give a smooth output.  Stepper motor drivers are inexpensive, so there is little to no financial benefit to building a driver.  And the time it would take is unnecessary.



(Vac = green, Vbe = red)

**Figure 3.7.2 - Analog Signals to Operate Stepper**

## 3.8 - Sensing

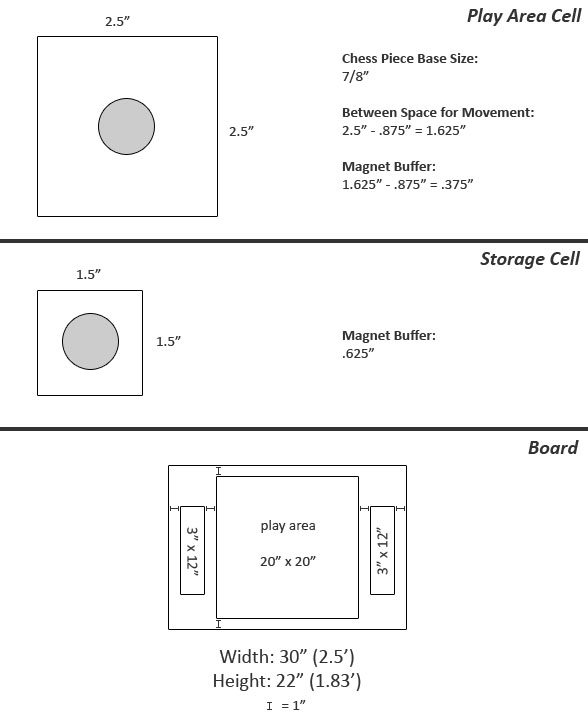
Hall Effect sensors were also investigated to be used as our proximity switch on the sensor board. In recent years the reed switch has been developed to improve its quality to rival the Hall Effect sensor making performance of both sensors not a deciding factor in which would be used. The reed switch was chosen over the Hall Effect sensor due to the price of the Hall Effect overall. Both components are priced fairly equally but the Hall Effect sensor requires additional power and wiring in order to work. The output of the Hall Effect sensor can also be low enough that an additional amplification circuit as well as a voltage regulator could be necessary.

The addition of these circuits makes the Hall Effect more costly in both time and resources. Another advantage of the reed switches are the ranges of input and output the switch can handle compared to the Hall Effect. The reed switch requires around 5 Gauss in order to close, whereas a hall effect sensor needs an input of 15 Gauss before it can be registered. This means larger magnets would be required to be placed under every chess piece. The issue with this method lies in the attraction from these larger magnets, if they are too big it can cause the chess pieces to be drawn together

## 3.9 - The Chess Board (Physical)

The chess board play area, or the chess board itself, can be either manufactured by our team or purchased from a manufacturer directly. Because our board requires special sizing due to our unique automation, we decided creating the board ourselves would be the best decision. The special sizing requirements are due to the fact that we have to be able to drag any piece from any point on the board to another. This means each play area cell must be big enough so that two together, with two chess pieces, still have enough room to allow a third piece to pass in between.

Using a chess piece size of ⅞”, we have created our initial play area cell size as well as storage cell size. These dimensions, shown in Figure 3.9.1, provide magnet buffers of .375” for the play area and .625” for the storage cell. Besides space for movement, this magnet buffer is the main variable in our dimensions. Since all of our chess pieces will have magnets attached to the bottom, for sensing purposes, the magnet buffer allows for the pieces to be moved independently, since other pieces will not be attracted to the movement magnet or each other. To color and create the play cells and provide styling to the plexiglass, clear acrylic paint will be used.



**Fig. 3.9.1 - Chess Board, Play Area, and Capture Area Dimensions**

## 3.10 - Materials

### 3.10.1 - The Enclosure

Once it was decided that the CyberChess playing board would be custom built, we had to decide what materials would be necessary to create the proper environment for the game. At first a glass enclosure was considered to house the whole project, the reason being that it would add to the visual appeal of CyberChess. The user will be able to see the movement system at work under the playing board and watch the mechanics behind the chess pieces moving. This approach would also include more L.E.D’s aligned along the borders of the whole enclosure to keep with the theme of the project. A glass enclosure would add a significant amount to the cost of CyberChess. Glass is not cheap and on top of purchasing the sheet of glass, additional cost will be added to employ someone to cut the glass to the specifications needed.

Wood was the main choice for building the enclosure of CyberChess because it can be handled at home with no need for special equipment or assistance and is readily available at low cost. Using wood will also make it easier to implement a shelf system that will provide easy access to the sensor board by allowing it to slide out of the main enclosure if maintenance is required. The wood needs to be treated correctly so it does not warp with time, this will affect the XY-plotter and full range of motion.

### 3.10.2 - The Board

Plexiglass was chosen as the main material for the chess board as it provides the clarity for the L.E.D.s as well as the ability to see through to our hardware and movement system. Plexiglass was also chosen because it provides an easy surface for the chess pieces to slide on, given they have felt bottoms.

Using ball bearings on the bottom of the chess pieces was an alternative to using felt on the bottom and would provide a lower coefficient of friction, making it easier for the chess pieces to move across the board. This would require the chess pieces to be hollowed out and have the ball bearings secured within. As the magnet beneath the sensor board could also attract the ball bearings in the pieces, interfering with their linear motion when a chess piece is moved, this possibility was deemed unfit for CyberChess.

### 3.10.3 - The Chess Pieces

The chess pieces used in CyberChess will be purchased fromWal-Mart due to ease. There were several choices for which chess piece set could be used for CyberChess, each could bring a different aspect to the design of the game. It was first thought to use clear chess pieces so they would be illuminated by the L.E.D.s beneath the playing board. Although this would add to the visual appeal of CyberChess, it would mean that each chess piece could not be fitted with any metal or magnet underneath to interact with the larger magnet of the moving system located under the board. This effectively renders CyberChess’s movement system useless, and thus is not a valid option.

Another alternative would be to purchase metal chess pieces. The metal chess pieces would be able to interact with the magnet attached to the XY-plotter allowing it to move across the playing board. This method would work for moving the chess pieces but would cause an issue with the sensor system leaving it ineffective. The reed switches in the sensor system require a magnetic field present to close the switch, if the chess pieces were metal the only magnet and reed switch interaction would be between a chess piece in motion.The only way to determine the location of every chess piece on the board is to fit each piece with a magnet small enough that it will not disrupt the movement but strong enough that it will activate the reed switch beneath it. As such, the chess set will likely be a simple plastic set, giving the team the ability to modify it as necessary throughout the project.

## 3.11 - Power Supply

The power supply used will be a wall wart with an output voltage in the range of 5V to 15V with a max output current of 1A. Depending on the build of the component purchased a bypass capacitor may be needed in between the power source and PCB. This capacitor will help reduce the amount of noise caused by remaining AC pulses of the current after it passes through the bridge rectifier built into the power supply. Multiple voltage regulators will be connected to the power supply to help maintain the proper voltage for each hardware component CyberChess is comprised of.

The two types of regulators that will be considered are linear voltage regulators and switching regulators each provide their own advantages and disadvantages for powering the circuit board that is CyberChess. Linear regulators tend to be cheaper and take up less space than switching regulators while still providing very little noise in the output. The main downside to using linear regulators is the heat that is distributed when the pass transistor receives heavy current when the difference between the input voltage and output voltage is large. They are best suitable for low powered projects where a high load current would not be an issue. (Torres, Gabriel)

A heat sink can be attached to the linear regulator to help solve the issue of heat given off but many heat sinks tend to be too bulky or expensive. This can lead to a pricy and clustered design if many regulators are needed. The linear regulator is also only capable of stepping down the input voltage, it is not able to produce an output voltage higher than the voltage used as input. The formula below is a general equation used to help determine if a linear regulator is wasting too much power.

Power wasted = ( Vi- Vo)   Iload

If the linear regulator is giving off more than a few watts of power it is advised to use a more efficient switching regulator in its place.

Switching regulators work by rapidly alternating the pass transistor between an on and off state sending the input voltage as small pieces and moving them to the output. This separation of energy causes the switching regulator to give off very little heat enabling it to handle much larger load from higher voltages.

When higher power is needed switching regulators have the advantage over linear regulators because additional components are not needed to remove heat. However, placement of switchers on the printed circuit board is important, if the loop gain or phase of the switching regulators is altered by interference the regulator can become unstable and oscillate.

The biggest factors to consider when deciding which regulators will be used in CyberChess are how much space the regulators will take up on the PCB and how much heat will dissipate from the components. To avoid complications with overheating hardware, it is advised that the power supply and regulators be external to the main PCB.

## 3.12 - Printed Circuit Board

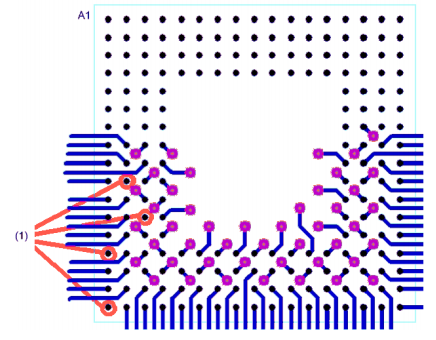
When building a printed circuit board, there are many small things that could negatively affect the performance of the design. The unwanted factors in the PCB design are known as parasitic elements.  It is imperative that the proper techniques are implemented to avoid these parasitic elements.  Many things come into consideration when dealing with electronic parasitics: capacitance, resistance, inductance and temperature are the most common unwanted occurrences.

Parasitic resistance can occur when a trace is too long.  The resistivity of a copper trace, for example, is 1.7 μΩ·cm.  For a copper trace of a constant area, a longer trace will create a higher overall resistance in the line.  For large currents and voltages, this may not be a huge problem, but PCB electrical characteristics are often are small as well.  Some MOSFET transistors which utilize leakage currents to run their operations may have currents as low as a few nanoamps.  With such small currents running through the traces, a small change in resistance could change the current by a relatively large factor.  A system designed to respond to small currents may not react so well to the error.  Another unwanted effect of parasitic resistance is the voltage drop.  If two neighboring traces experience a large enough voltage drop, the coupling capacitance could cause problems as well.

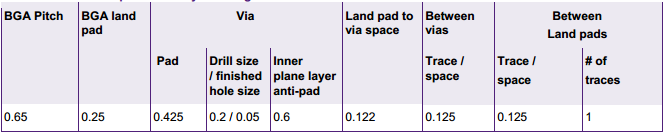
This unwanted capacitance is called parasitic capacitance.  One of the main problems with parasitic capacitance has to do with amplification.  If the capacitance it near an amplifier, the error could be largely increased.  A large enough parasitic capacitance could create a path of feedback and destabilize the electronic network.  At low frequencies, parasitic capacitance can usually be negated.  The problems mostly occur at higher frequencies.  Inductors are know to create parasitic capacitance within themselves.  While an inductor is charging or discharging, the windings have small voltage differences between each turn.  At high frequencies, an inductor experiences rapid changes, and therefore the potential between the windings may be greater, causing a larger capacitance.  This can slow down the performance of a network relying on the inductor.

PCB stacking is another aspect of the Printed Circuit Board that must be taken into consideration.  Supplying the board its ground and voltage source via traces is not recommended for designs that require good performance.  A few problems can arise due to varying lengths of trace between different components and the ground or supply.  As mentioned earlier, resistance in a trace gets larger as the trace becomes longer.  So if two pins should be grounded, but one is connected to ground by a significantly longer trace than the other, there may actually be a voltage difference between the two.  In effect, one of the pins won’t actually be grounded.  And the operation of the system may suffer because of this.  By stacking the PCB, we can create uniformly charged layers.  The vias extending from these layers to the component pins on the top layer of the board will be about the same distance, minimizing the potential difference between two pins stemming from the same source.  
  
A few common stacking configurations are the two layered, four layered, and six layered PCBs.  The two-layered board is usually set up with a signal layer above a ground layer, which are both considered conductive layers.  The conductive layers are separated by a dielectric layer made of material called prepreg.  Prepreg is basically fibreglass mixed with epoxy resin.  Below the ground layer is the core, which is made of fibreglass epoxy resin also.  In many cases, another dielectric layer, made of soldermask, will be added above the signal layer.  All in all, a 2 layer PCB has two conductive layers as well as 2 or 3 non-conductive layers.  In a 2 layer board, the parasitic resistances due to bad grounding can be pretty much eliminated.  However, the supply voltage still has to be distributed at the signal layer, through copper traces. This leaves some improvement to be desired.  
  
The four layered PCB is set up with the first conductive layer being the signal, the second layer as the ground, the third layer is the supply voltage, and the fourth layer is another signal layer.  The supply and ground can reach the signal layers through vias, just like in the two-layered boards.  The signal layers can reach each other via through-holes.  Through-hole technology allows for connections to be soldered on the bottom side of the board, between components that are sitting on the top layer.  Similar to the two-layered PCB, the four-layered PCB has dielectric layers separating the conductive layers, as well as soldermask on the outermost layers.  This PCB design resolves a lot more issues than the two-layered design.  
  
The six-layered PCB adds two signal layers between the voltage supply plane and the ground plane.  More layers can be used in a PCB to reduce parasitic problems produced within the signal layers.  
  
3.12.1 PCB for BGA  
  
In order to accommodate our pin-heavy Ball Grid Array (BGA) microprocessor, research into fan out techniques was necessary. The use of BGAs has become quite common for applications requiring medium to high pin-count IC packaging. In researching fan out techniques it was also discovered that we must pay close attention to additional areas such as solder paste chemistry, reflow solder profile and solder paste stencil etching, all of which can affect the outcome of our printed circuit board.

Since the BGA balls are arranged in matrix fashion, routing all of them on a single layer would be nearly impossible. Through the use of fan-out vias, or escape vias, this problem can be avoided by allowing the routing of the balls on other layers. Currently we have samples of the ZCE AM1808 package which contains a BGA with a ball pitch of .65mm while the alternative package, the ZWT, has a ball pitch of .80mm. These two packages create different specifications for our vias, land pad to via space, trace space between vias as well as trace space between land pads.   
  
For our purposes, the fan-out pattern for our package centers each via within the space between four adjacent BGA land pads. The vias are placed 1.3 mm apart, with every other location being skipped, and alternating them between adjacent rows. A partial example is shown below in figure 3.15.1, and the layout tool design rules are shown below in table 3.15.1.

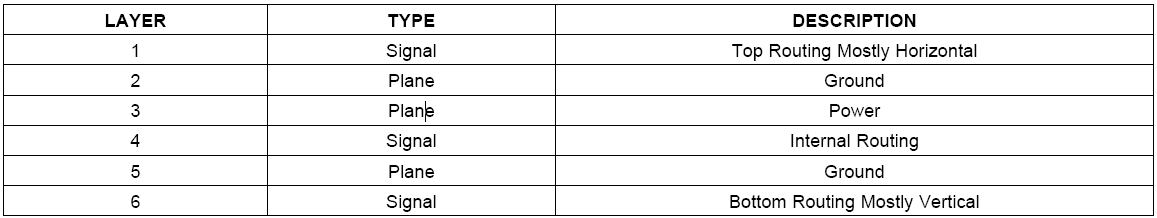


**Fig 3.12.1 - .65mm BGA Fan-out Technique**

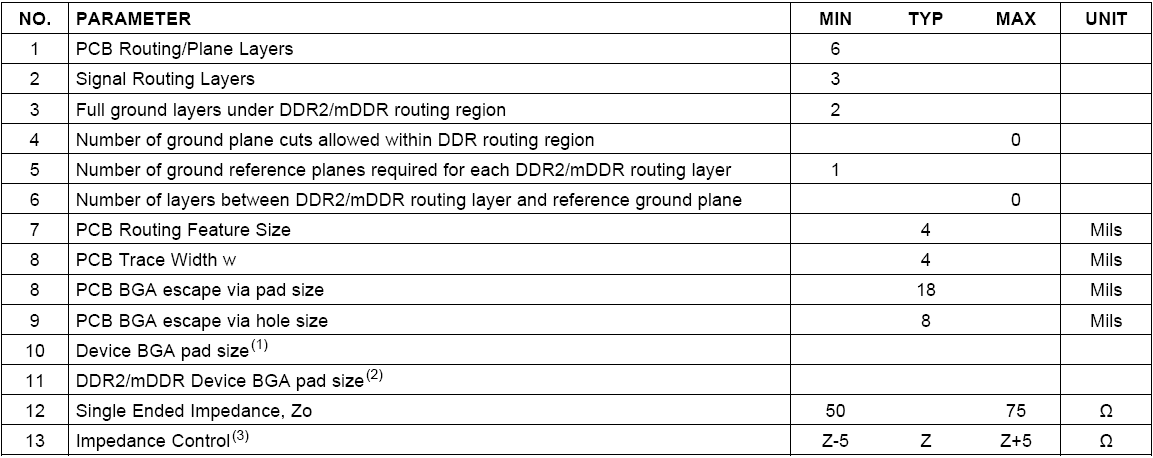


**Table 3.12.1 - .65mm BGA layout design rules**

Specifically for our application, Texas Instruments provides information on the minimim PCB stack up as well as the PCB stack up specifications. This information is shown below in tables 3.12.2 and 3.12.3. (AM1808 datasheet)



**Table 3.12.2 - Device Minimum PCB Stack Up**



**Table 3.12.3 - PCB Stack Up Specifications**

## 3.13 - Operating System

For the purposes of CyberChess, the operating system chosen for the final hardware must be lightweight, flexible, and readily usable by the development team. The OS needs to be as bloat-free as possible so as few things as possible get in the way of the system running smoothly. Stripping out such bloat is most easily done in an operating system for which the source code can be manually modified and compiled. As most software development will occur on an Intel architecture with ARM as the destination, compiling the operating system will have to make use of cross-compilation - the act of compiling software on one architecture so that it will execute on another architecture. Various programs are available to perform such cross-compilation, and while it will be a useful experience and learning opportunity, cross-compilation itself should not result in any significant hardships.

All modern, feature-full open source operating systems are UNIX-based, with the most powerful options being based on the Linux kernel. A selection of popular Linux distributions offer support for the ARM architecture – some are supported by the same team or company that produces the Intel offering, while others exist as ports by a separate team. Much of the information below is based on prior experience with the respective distributions, supplemented by research into their ARM counterparts.

### 3.13.1 - Arch Linux ARM

Arch Linux is a Linux distribution based on the idea of providing simple, accessible customizability to the end user. The majority of system customization is maintained in simple textual configuration files. Pacman, the main package manager of the distribution, provides simple, powerful functionality in locating and installing binary programs of any of thousands of packages available in the officially supported repositories. The Arch User Repository (AUR) further extends this by providing access to even more packages that are voluntarily maintained by users, and access to this repository is simplified by any of a number of programs that act as wrappers around Pacman, adding enhanced functionality to the already powerful package manager.

Arch Linux ARM is developed and maintained by a separate team from that which actively works on the main Arch Linux distribution. However, the ARM team cooperates with the main development team, resulting in Arch Linux ARM being a true port of the distribution to the ARM CPU architecture. Arch Linux ARM is fully supported on any processor that supports ARMv5TE or higher, including the OLinuXino iMX233.

Installation instructions using an SD Card are given on the Arch Linux website for our development board in addition to a few notes, one of which warns of the 64MB RAM limitation of our development board, stating we will need swap for system upgrades including such activities as generating locales for glibc. In Linux, RAM is divided into chunks called pages. Swapping is the idea of moving a page of memory out of RAM and into a preconfigured space on the hard disk, appropriately named swap space. This act frees up precious space in the physical RAM. This combination of the swap space and the physical RAM makes up the total virtual memory. This problem can be solved by using a partition manager to create a swap space on the SD card with a 1MB gap between the partitions. After booting with the new modified SD card, the user must activate swap in Linux using the terminal then run the problematic code or upgrade again. The “swappiness” of the swap can be set to 1 to reduce its usage to only when necessary, since the default value of 60 makes the system slow and wears out the flash memory.

### 3.13.2 - Debian ARM

Debian has always been known to support more hardware architectures than other Linux distributions and includes many packages specific to each ARM version. For our applications and usage, we would need to use *arm-linux-gnueabi* in Lenny in the *armel*section, which supports ARMv4t and up. The most advanced version of Debian supports ARMv7 and up as well as the Thumb-2 instruction set. (Shroder, Carla)

### 3.13.3 - Ubuntu ARM

Ubuntu is one of the most popular desktop Linux distributions, being designed to be readily accessible to new users with no previous experience with POSIX environments. It is built upon Debian, and thus bears a certain underlying resemblance to that distribution. Being designed with ease-of-use in mind, Ubuntu hides much of the system configuration from the user, giving them ready access to less powerful GUI-based controls while restricting further access. Such hindrances of direct control over the system would only result in frustration, and the additional bloat associated with being user-friendly makes Ubuntu less than the best choice.

Ubuntu ARM is a complete Linux ARM distribution with thousands of packages available for both desktop and server. The Ubuntu Netbook Remix (UNR) brought about a very promising feature that made use of the Enlightenment Foundation Libraries to power a 2D user interface with features similar to that of a 3D interface. UNR went away with the release of Ubuntu 11.04 as Ubuntu ARM and Desktop were merged. The current Ubuntu ARM port supports ARMv7 and up which excludes our current target microprocessor.

### 3.13.4 - Gentoo

Gentoo is one of the most flexible Linux distributions, officially supporting at least 10 CPU architectures, including ARM. The distribution is based on a philosophy of giving the user extreme control over optimizing their environment. While binary packages do exist and can be installed on a Gentoo system, the most common method of program installation involves using the Portage software distribution system to download sources and compile them client-side. These programs are optimized by the user’s use of various compilation flags to indicate to the compiler what features are and are not necessary. In this manner, the user can optimize individual programs to contain nothing more than what is needed for their purposes. While this gives the user explicit and powerful control over optimizing the system, it comes at the price of relatively time-consuming upkeep - every update to a program requires complete recompilation of that program, any change of USE flags (universally applied compilation flags that can a user can override as needed) requires recompilation of all affected programs. This excessive upkeep would be unlikely to directly affect CyberChess, due to its single, specific purpose.

### 3.13.5 - Android

Android is a Linux-based operating system designed for use in devices like smartphones and tablet computers. The main platform for Android is the ARM architecture and would be an excellent fit for CyberChess if we were using a more advanced processor such as the ARM Cortex-A or Cortex-A8. While Android could definitely run on our microprocessor, performance would be impacted. If we were to go with Android it would probably be necessary to modify our hardware to include a DSP since we could offload all of our multimedia processing to this processor. (Etheridge, Darren)

## 3.14 - Shells

Shells provide user access to the functionality of the operating system’s kernel, allowing for direct control over the system. A number of shells exist for interacting with the Linux kernel, including the historically beloved Bourne Shell, the Bourne Again Shell (Bash) which builds upon it, and the recently popular Z shell (zsh). While CyberChess will ultimately operate without the users having access to whichever shell is selected, picking a strong shell will help facilitate better development, and the system will still make use of shell scripts to automate behavior at boot up and shut down.

### 3.14.1 - Bourne Shell

The Bourne Shell is effectively the canonical shell of all UNIX-based operating systems, having been the primary shell of UNIX Version 7. Any modern \*NIX machine can be expected to have support for executing shell scripts written for the Bourne Shell, and many modern shells, including the other two being investigated for CyberChess, provide full support for executing these scripts with little modification. The Bourne Shell does show its age with comparatively limited functionality and availability of abstractions, so while the historical value of the shell is undeniable, CyberChess will likely instead make use of one of its successors.

### 3.14.2 - Bourne Again Shell (Bash)

The Bourne Again Shell, more commonly simply called Bash, was created as a replacement for the Bourne Shell and has become the standard shell included on most Linux distributions. With support for a greater number of features, Bash is effective not only for interaction with the kernel, but also for basic prototyping of program ideas. As the default shell of most Linux distributions (in fact, most will simply use a symbolic link to have the path for the Bourne Shell instead use Bash), Bash is a strong and obvious choice for the shell to use.

### 3.14.3 - Z Shell (zsh)

One final consideration, mainly due to favorable prior experience, is zsh, another shell designed as an extension of the Bourne Shell taking inspiration from various shells that have historically built upon it. Zsh offers complete compatibility with Bourne Shell scripts when invoked by way of symbolic link as /bin/sh, as well as extreme customizability throughout all of its features. While this makes it a wonderful tool to play with, none of these features are likely to provide any advantage substantial enough to offset the additional weight in comparison to that of Bash.

## 3.15 - Speech Recognition

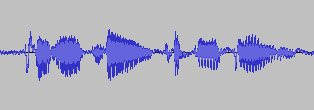
The main component of the project that makes CyberChess stand out from other automated chess games is the speech recognition that serves as input to our other engines. An independent voice recognition system will be needed for CyberChess to ensure that all users will be able to give voice commands that can be recognized. It was decided that PocketSphinx software from Carnegie Mellon would be most efficient for CyberChess. PocketSphinx is programmed entirely in C, which in recent years has been claimed to compile almost as efficiently as assembly.

### 3.15.1 - How Speech Recognition Works

In order to understand how Speech Recognition is done we first must take a look at the structure of speech. Speech is a sequence of stable states mixed with dynamically changed states. Throughout this sequence, similar classes of sounds called phones and diphones (parts of phones between two consecutive phones) create words. From a software perspective, sub phonetic units, different substates of a phone, are often emphasized. Usually three or more regions per phone can be found. Phones create subwords like syllables and subwords form words. Words and non-linguistic sounds like breathing, “um” and “uh”, cough, etc form utterances, the audio chunks between the pauses.

To recognize what was said, the audio waveform is split into utterances and paired with the closest match. An example audio waveform of “PB2 to B3” is shown below in Figure 3.9.1. First a feature vector, typically taken from 10 milliseconds of audio and composed of 39 numbers, represents the speech in a comparable form. Then a model, a mathematical object that gathers common attributes of the spoken word is compared against and the best model used.

Three models are used to find the match: an acoustic model, a phonetic dictionary, and a language model. The acoustic model is the acoustic properties of each senone. The phonetic dictionary contains a mapping from words to phones. The language model is used to restrict word search. PocketSphinx was chosen specifically for the reason that it provides us with the acoustic model for the English Language as well as the tools to setup a language model and phonetic dictionary for our specific application.



**Fig 3.15.1 - Audio Waveform of “PB2 to B3” recorded using Audacity**

The language model and dictionary files are created using an online converter provided by the CMU Sphinx Toolkit. To create them, one must first create a *Corpus.txt* file which will contain every possible phrase or command, one per line. This file is uploaded via a web browser and converted into a randomly generated 4 digit .lm and .dic file. These files are then referenced at runtime to compare inputs from the microphone.

### 3.15.2 - Problems with PocketSphinx

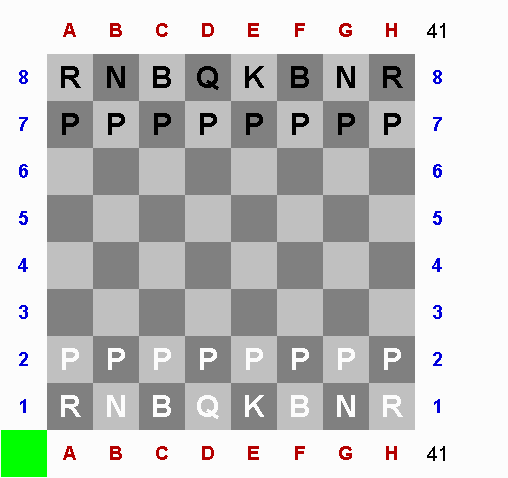
In testing our own language model and dictionary files it was discovered that PocketSphinx has some difficulty in recognizing the difference between the letters D and B, almost always defaulting to B. To fix this problem, we could retrain PocketSphinx’s acoustic model with more accurate recordings for those letters, change our coordinate system into words like Dog and Bear for D and B respectively, or to implement a more complex naming system (as shown in Fig 3.i.1).

Retraining the acoustic model would require us to record our commands and words at a sampling rate of 16KHz in mono with single channel. These recordings would then generate acoustic feature files and we could tweak them until our program was better at recognizing the different letters. Since PocketSphinx works near perfectly for our applications out of the box besides this small problem, adapting the acoustic model would require a lot of unexpected work.

Using words in place of letters is a simple solution that would alleviate our problems, but would require us to teach these names to our users. This could be done through images on the board or a pre-recorded audio clip before the game starts, but it may become troublesome for users to remember 8 random words after hearing them once.

The third solution involves a more complex naming system that includes the piece initial as well as its location. For example, a pawn at A2 would be fully called “PA2.” This system, like the words, would require the user to know the proper name of every piece and the exception of Knight represented with N, with K being reserved for King. This solution does not fix the problem fully though as it is still possible for two pieces of the same type to exist on a D and B cell at the same time. For example, at the beginning of the game there are pawns PB2 and PD2. This would require an additional system to this solution in the code that would check if a similar type piece exists on both B and D cells with the same number, then ask the player for confirmation of letter, if it is not possible to determine their input through their move-to location.

Although this naming system also has a small learning curve and a few conditions where additional steps or actions must be taken, it is currently the best solution to our problem.



**Fig. 3.i.1 - Complex Naming System**

***Permission Pending***

### 3.15.3 - Further Speech Recognition Research

A hurdle to overcome when using speech recognition is loud or noisy environments. Thankfully games of chess usually occur in a quiet setting, CyberChess still needs to function with apparent background noise. To help counter this problem we looked into Bluetooth Headsets, Cell Phones, and simple wired headsets.

Bluetooth headsets would require us to incorporate a Bluetooth module into the board but they come with built in background noise reduction. A cell phone based solution could be done wired or wirelessly, either through an auxiliary cable, WiFi, or bluetooth. A wireless implementation would again require us to add more hardware and a wired implementation would require us to write software for the cell phone in order to be able to process the data.

For these reasons, it was decided standard wired microphone headsets would suffice. The close-to-mouth microphone makes sure we are hearing the correct source and a simple command start word could be used to make sure background noises would not confuse the system.

### 3.15.4 - Sphinx4

Sphinx4 comes from the same open source project at Carnegie Mellon University that created PocketSphinx. Sphinx4 is written entirely in Java and serves cloud-based systems with deep interaction with NLP modules, web services and cloud computing best. Since CyberChess relies on live speech recognition embedded into the Linux platform, Sphinx4 is not a good choice. (Carnegie Mellon University)

### 3.15.5 - eSpeak

eSpeak is an open source speech synthesizer that was developed to handle multiple languages and run in either a Linux or Windows environment. One of the main features of eSpeak is the small amount of memory it requires –all of the relevant software and its data can be stored in just 1.4MB. eSpeak is available as either a command line program or in a shared library format to be used in another program. It is written entirely in the C programming language, leading to its small size. Not only does eSpeak handle text to speech but it also supports text to phoneme codes as well, allowing it to be integrated into another speech synthesizer engine’s acoustic modeling tools.

### 3.15.6 - Android

An alternative to PocketSphinx could be to use Android and its built in library for Speech Recognition. The Android SDK can integrate speech input with an application that we would create using the RecognizerIntent. This intent prompts the user with a “Speak Now” input and sends the data to Google’s servers to be processed. Currently, Google can provide speech recognition for English, Mandarin Chinese, and Japanese.

Using the Android SDK to provide speech recognition would require us to run Android as our Linux distribution in addition to adding a WiFi module to connect to Google’s servers. Alternatively, we could use Android Smartphones and their built in connectivity to process commands, but this would require us to write software for the phone.

### 3.15.7 - Windows

Both Windows Vista and Windows 7 come with built in Speech Recognition software. It has support for dictation of documents and emails as well as voice commands for nearly every operation on the computer including starting and switching between applications, controlling the operating system, and filling out forms on the web.

Windows Speech Recognition comes with support for English (US), English (UK), German, French, Spanish, Japanese, Chinese (Traditional), and Chinese (Simplified). It also modifies itself to accommodate both headset and desktop microphones. The use of Windows Speech Recognition would first obviously require us to have a Windows Environment tied into CyberChess. This could be done using a laptop, but would take away from the beautiful embedded system we have designed. For this reason, Windows Speech Recognition was not chosen.

## 3.16 - Board Representation (Logical)

### 3.10.1 - Language

Software production always requires the programmer to pick the best tools for the job. The most important tool to the project is the programming language itself - each language provides different benefits and, unfortunately, limitations. Each software element of CyberChess to be produced could potentially benefit from a different language, and thus inspection of various languages is imperative. Looking towards being able to use benefits of previous knowledge, C, Common Lisp, Java, and Python were most heavily investigated. Cursory research of other alternatives included looking into C++ and JavaScript, two of the most popular languages for software development and web development respectively.

All languages that were looked into are relatively well known throughout the world of software, and likely need little introduction. C, as one of the oldest and most prolific system programming languages, is universally recognizable as a staple of programming. It offers the benefit of speed, with code compiled directly down to native machine code and control over the lower levels of execution placed directly in the hands of the programmer. Operating at this low level allows for heavy optimization, but comes with the responsibility of manual memory management - unlike many of its successors, C expects the programmer to allocate and free memory as needed, having no concept of garbage collection.

C’s widespread use has also resulted in plentiful documentation of the various features of the language, as well as a ready supply of examples of implementations of various data structures and algorithms. Use of C for large projects, however, can quickly become a challenging task, as the language is still very low level - it compiles to efficient machine code because it is quite close to it, acting effectively almost as shorthand. This means that many concepts that other languages have abstracted away - a simple example being iterating through and acting upon the elements of a linked list - must be implemented by hand. While other libraries beyond C’s standard library do exist to fill in missing functionality, having such additional external dependencies is not seen as desirable for a project intended to be lightweight.

In direct relation to CyberChess, C provides little in the way of abstractions to aid in creation of the software, but it does give the programmer extensive control over low-level operations. Many efficient implementations of chess programs make use of such operations, playing around at the bit level to minimize memory usage. As a procedural language, C has no built-in support for object-oriented programming - a paradigm which seems fitting for creating a virtualization of chess. However, such a paradigm can be mimicked with creative use of structs, data structures that can contain any combination of data types, including other structs, and function pointers, variables that refer to functions by their memory location. This involves much more effort and intentionality than a language with this functionality built in, but the benefits of efficiency and having unhindered access to low-level mechanics could well be worth the added complexity. With its direct access to memory, C is an obvious choice for direct control over the hardware, such as the motors and L.E.D.s. It is a strong possibility for the implementation of the game as well.

Common Lisp sits on the other end of the spectrum as a language that has a plethora of abstractions ready for the programmer’s use. In its interactive form, Common Lisp makes use of a rather advanced version of a Read-Eval-Print Loop, often referred to as a REPL, in which code is first read, then evaluated, then the output is printed. While this does not sound overly revolutionary, and a number of other languages have such a construct, it proves to be quite powerful in Common Lisp. Programmers can implement reader macros to alter the behavior of the Read part of the REPL, literally changing the syntax of the language to meet their needs. Before code is evaluated, user-defined macros expand to write entire sections of code for the programmer based on a template they previously defined. As a Lisp dialect, the code is composed entirely of S-expressions, nests of lists infamously enclosed in parentheses.

This uniform representation of code and data allows macros to flawlessly nest within the code itself with no worry of the code breaking. Mark Jason Dominus, a prominent Perl programmer writes about Common Lisp’s use of macros as the main form of assignment - setf, the assignment operator, is itself a macro that not only expands to a call to the appropriate function, but can be readily updated to reflect assignment to a data type constructed by the programmer. (Tilly, Ben) This flexibility and the accompanying expressiveness is the core strength of Common Lisp - Peter Seibel’s *Practical Common Lisp* demonstrates the creation of a unit testing framework in a total of 26 lines of readily understandable code. Unfortunately, this power comes with some serious drawbacks - compiled Common Lisp applications tend to be extremely large, as the entire language must be present within the application. (Seibel, Peter) Their performance is not up to par with most compiled languages. Common Lisp also suffers from its own flexibility - the community surrounding it sees frequent mentions of the fact that the triviality of implementing something that is good enough for a single user’s purposes has resulted in very few libraries being produced that fit any more generalized purpose.

Common Lisp does not offer simple access to lower level operations and hides all memory management in its abstractions. As such, the efficiency of any program written for CyberChess would effectively fall to the whims of the compiler, tempered by those suggestions the programmer is permitted to make. The expressive power of the language, which includes a fully formed object-oriented suite, known as the Common Lisp Object System (CLOS), is certainly a desirable feature. Considering the final product will be in a relatively limited embedded system, this power will likely not be worth the inability to easily influence the space efficiency of the program.

Java is often considered to be in the same family of languages as C, with heavy influence visible in its syntax. While it supports various paradigms, it is often used for its solid support of object-oriented programming. Java is a memory-managed language, with a relatively advanced Garbage Collector (GC) that progresses objects from a young generation to an old generation which is checked for collection less often to improve efficiency. Most of the memory management is hidden from the programmer and low-level operations are not as readily available as in C, nor quite so obscured as in Common Lisp.

While some compilers can compile Java down to native machine code, the standard practice is to compile to an intermediate bytecode which is run on the Java Virtual Machine (JVM) as a means to make these compiled files universally usable. While this is useful for applications that are expected to proliferate throughout a number of machines, CyberChess is being designed with the software and hardware hand in hand. Thus, the benefit of universal usability is effectively moot - it may be nice to be certain that the program will act exactly the same regardless of the production environment not being identical to the final deliverable, but it is not a chief concern. Unfortunately, as virtual machines can never run quite up to speed with their physical counterparts, the speed loss this universality requires is still present. Just-In-Time (JIT) compilers attempt to offset this by caching code and compiling it to native machine code right before it is to be executed, and good JIT compilers have begun to approach the speeds of natively compiled code. Java does offer an extensive set of libraries, providing excessive functionality to the programmer. The object-oriented nature of Java is certainly appealing for the design of a chess engine, but the speed lost to running on the JVM without Just-In-Time compilation may not be sufficiently offset.

Python is another multi-paradigm language that has steadily been gaining popularity. While it is most often used as an interpreted language, it can, like Java, be compiled to intermediate bytecode and executed on a virtual machine. Python was designed to be highly expressive and powerful, with simple, English-like syntax representing large abstractions. It is often informally described as “batteries included” - it features an extensive standard library that provides functionality beyond that of any of the other three languages described above. Despite *The Zen of Python*’s (PEP 20) advice that “There should be one - and preferably only one - obvious way to do it,” multiple implementations of chess engines in Python can readily be found. The expressivity and ready supply of resources makes Python a strong contender for use in designing the virtual implementation of the rules. The main drawbacks are again speed and the lack of access to lower level operations.

C++ is popular for software design due to its ability to compile down to native machine code, like C, with support for numerous paradigms, similar to Java and Python. It is effectively, though not actually, a superset of C, and thus most valid C programs compile and execute the same in C++. Programs written in C++ are roughly equivalent in execution time to their C counterparts and C++ is often used as the standard to be compared to in benchmarks between languages. The main detractor for use of C++ in CyberChess is lack of prior experience - while learning a new language is always desirable, it can be a trying activity in the midst of creating a project.

With every modern web browser supporting a full implementation, JavaScript is considered to be the native language of the internet. It offers many of the same features that Java, Python, and C++ provide. The main advantage of its use would be easy potential to bring CyberChess’s engine to the web. As this is not a major design goal, this is not a strong deciding factor, and while JavaScript is certainly a full-featured and growing language, it does not offer anything important the other considered languages lack.

### 3.10.2 - Board Data Structure

Research into board representation was initially in the form of attempting to implement the board in various ways to find those which were the easiest and most flexible to work with. A simple array implementation was investigated and, while relatively easy to deal with, was ultimately dismissed as being unnecessarily inefficient. Keeping track of a set of piece positions without an actual virtual representation of the board was attempted as well and found to work decently, but having a virtual board to interact with was found to be more natural.

CyberChess aims to have compact and efficient processing of the chess game itself. The engine should be a minor processing concern, and as such an efficient data structure for board storage is required. The data structure should make checking the validity of generated moves direct and non-intensive. Various data structures have been widely used in other chess engine projects, but two implementations stand out due to usability and efficiency - the 0x88 method and bitboards.

The 0x88 represents each piece location as a single byte, with file and rank corresponding to the first and second nybble (or vice versa, as the programmer desires). This effectively allows reference to four adjacent boards, with only one being the valid board. While this may seem to be wasteful, simply logical-anding a location with the hexadecimal value 0x88 will indicate whether a test location exists on the valid board - if either of those bits are set, the location is in a file or rank on one of the adjacent boards. This simplifies both programmatic generation of moves and checking the validity of moves being requested by a player.

Bitboards store a current board state in a collection of 64-bit integers, using each bit as a flag for the presence of a piece on a one-dimensional representation of the board. This provides the benefit of being able to perform a wide variety of operations with simple, efficient bit-shifting. Unfortunately, the benefits are reduced in non-64-bit processors, due to their inability to handle entire bitboards each clock cycle. Bitboards also have a high time versus space tradeoff - the most time-efficient usage relies on making use of a static set of pre-generated bitboards rather than dynamically creating them as needed. Simply having bitboards representing all possible movement vectors takes 32 kibibytes of storage, with numerous other possible sets adding additional speed benefits at the cost of more memory. The benefits of bitboards are seen most in applications that need to generate many moves, such as chess AI systems.

### 3.10.3 - Algorithms

The final software implementation of the rules of the game will be responsible for checking that any given move is both valid and legal. There are various means of performing these checks, but a proper implementation should seek to minimize time spent on them.

Checking move validity is simply a question of whether the destination exists on the board. Determination of this is effectively built into the board representations addressed above - 0x88 board representation requires only a simple logical-and operation to determine a location’s validity, while bitboards simply do not permit invalid moves by nature of only ever operating on the valid board. As such, there are no real clever algorithms to further enhance the offerings of these representations.

Checking move legality is a more involved process, requiring knowledge that the desired move obeys the piece’s movement patterns, is not in any way obscured, and that it does not result in breaking any rules of the game. Verifying movement patterns is fortunately simple for many pieces - means of doing so are addressed in table 3.10.1 below. All listed methods require very little processing time, relying heavily on checking equality of values and simple mathematical operations, two things processors are designed to be very efficient with. These methods are what would be used in conjunction with the 0x88 board representation. In order to ensure the game progresses properly, an attempt to move to the same space the piece currently occupies must register as illegal.

Bitboard representation would simply logical-and the requested move with a representation of valid moves for the respective piece type at its current location (an array of such bitboards would take 3 kibibytes of memory). The resultant bitboard would reflect the validity of the requested move, effectively reducing the check for move validity to a constant time operation. This is where the benefits of bitboards can be seen quite readily in systems which must process large quantities of move requests rapidly – every request can be handled in a single memory access and single bit operation, as compared to the variable functions required by the 0x88 board representation.

|  |  |  |
| --- | --- | --- |
| **Piece** | **Movement Pattern - Patterns assume all spaces are unblocked unless otherwise noted.** | **Programmatic Method** |
| Pawn | The Pawn has the most complicated movement pattern to define, being dependent on its location and the type of move:   * At any spot, the pawn may move forward one space in its file. * At its initial spot, the pawn may move forward two spaces in its file. * A pawn may only capture a piece that is in one of its adjacent files, one rank ahead of it.   Additionally, what is considered to be forward differs based on color, and thus color must also be addressed. | The need to know a Pawn’s color will require use of conditional statements or clever combination of bit shifting and multiplication involving the internal piece representation to determine if rank should be increasing or decreasing. With this knowledge, the system will then simply run through a series of equality checks based on adding appropriate values to the rank and file. |
| Rook | The Rook moves only in straight horizontal or vertical lines. | Simply XORing the results of checking that the file and rank of the source and destination are equal will return a truth value corresponding to the validity of the move. |
| Knight | The Knight moves in an L-shaped pattern, advancing two spaces in one direction and one space perpendicularly. | A naive approach might sum the absolute values of the differences between the destination and source file and rank and equate this to the value 3. This would allow the Knight to move |
| **Piece** | **Movement Pattern - Patterns assume all spaces are unblocked unless otherwise noted.** | **Programmatic Method** |
|  |  | three spaces in straight lines. Instead, equating the sum of the squares of these distances to 5 will ensure that only moves matching the Knight’s movement pattern register as valid, while also removing the need for the slightly more intensive operation of finding absolute value. |
| Bishop | The Bishop moves only in straight diagonal lines. | Equating the squares of the differences between destination and source files and ranks returns the validity of the move. This must be supplemented by a check that these distances are not 0 to ensure the piece is actually moving. |
| Queen | The Queen moves in straight lines in any direction. | ORing the results of the move validity for a Rook and Bishop will properly determine the validity for the Queen. |
| King | The King moves a single space in any direction. | Checking that the square of the file difference and the square of the rank difference are each less than or equal to 1 will determine the validity of the move. This must, like the Bishop, be supplemented by a check that the distances are not both 0. |

**Table 3.10.1 - Piece Movement Patterns and Algorithms**

Checking that a move is not obscured requires additional computation. With a bitboard representation, bitboards containing 1s at each spot between two points and 0s elsewhere would be available, taking up a total of 32 kibibytes. Using a logical-and operation with the current board state would indicate whether the requested move is blocked. In a 0x88 representation, the spaces between the two spots must be iterated through in a simple loop, checked one by one. While this is a comparatively slow process, the loop will never need to check more than 6 spots, which is still a rather inconsequential processing load.

Finally, the move’s legality relies on its obedience of the more abstract rules. Availability of an en passant capture must be maintained in some form, either in the board representation or alongside it, to allow the engine to properly handle an attempt at such a capture. The validity of castling requires that the King and the castling Rook have never moved and that the spaces between them be empty. Knowledge of the movement of one of the two pieces can be maintained by simply having a 4-bit word that acts as flags for the availability of each possible castling move (white and black, king and queen side), mimicking the storage of this knowledge in Forsyth-Edwards Notation (FEN). Castling, like any movement of the King, also requires that no space the King would be moving through be in check, the most complicated of the abstract rules to address.

Verifying that a space is not threatened requires ensuring that no enemy piece can reach that square in one move. For bitboards, this requires a series of logical-and operations, performed on the bitboards representing the current location of each piece type of the opposite color and that type’s movement pattern starting from the square in question. In 0x88 representation, this can be accomplished for any arbitrary spot in two main ways, and can be checked after each move in one additional, less processor-intensive way. The most demanding method starts at the space in question and attempts every movement pattern available from that spot - up to 35 positions, each of which must be dynamically generated for 0x88 representation. If knowledge of the location and type of all enemy pieces is maintained, it is instead possible to iterate through this list and simply attempt the move using each piece as the start location. This requires additional information storage - namely, the list being iterated through must be maintained and updated after each move.

While movement of the King requires a thorough check like those described above, movement of any other piece can be simplified to checking along the vector that continues in the direction from the King to the other piece’s start location. If this vector is not a straight line on the board, the move does not have the potential to put the King in check. Otherwise, simply proceeding along that line will encounter any piece that could potentially render the move illegal. Similarly, it must be determined if a player’s move leaves their opponent in check. This is readily handled by attempting to move from the destination square to the opponent’s King’s square as well as following a similar vector, from the opponent’s King through the source square, to look for pieces that now threaten the King.

# 4 - CyberChess Hardware

## 4.1 - ARM9 AM1808 Processor

The ARM9 AM1808 is the microprocessor we chose to be the core of CyberChess. There were many reasons we needed a processor like this one.  First of all, it is an inexpensive processor.  The processing power in the AM1808 is plenty for CyberChess, without being too over the top (like the Cortex-A8 in respect to this project).  The AM1808 contains 37 registers total, status registers occupy 7 of them leaving 30 general purpose registers.  It has a total of 144 GPIOs, which (along with our various drivers) should be enough to support all of our systems (L.E.D.s, motors, switches, etc.).

Twenty of the pins (PINMUX0-PINMUX19) are capable of multiplexing, each with a 4-bit field.  This may allow for the control of more outputs (such as numerous L.E.D. drivers) without the requirement for a large amount of pins.  As mentioned in Chapter 3, the AM1808 has auPP to interface with the DAC and ADC converters.  It’s internal Direct-Memory-Access controller allows the uPP to operate without much interference to the microprocessor’s other operations.  Figure 4.1.1 (below) shows the AM1808 Functional Block Diagram.



**Figure 4.1.1 - AM1808 Functional Block Diagram**

### 4.1.1 - Accessing GPIO

The ARM9 AM1808 used for CyberChess comes with 144 GPIO pins which will be responsible for handling the input and output to the microprocessor. The GPIO can be accessed through the linuxuserspace or it may be implemented into the C source code. The gpiosysfs interface must be supported by the linux kernel used in order to manage the GPIO from the linux shell. Linux kernels 2.6.30 and above come with the GPIO support already included. The next step in handling the GPIO is determining how many pins are available on the given SoC. Before handling a gpio signal, the pin must be exported to the interface using the following prompt with N being the number of the pin accessed.

echo N > /sys/class/gpio/export

The next step in accessing the GPIO is determining the direction of the pin. Enter out for an output pin and enter in for an input pin.

echo out > /sys/class/gpio/gpioN/direction

echo in > /sys/class/gpio/gpioN/direction

The final command for handling the gpio pins is for writing a value of either high or low to an output pin. To perform the write function the following is entered with a “0” for a low value and “1” to write a high value to the pin.

echo 1 > /sys/class/gpio/gpioN/value

echo 0 > /sys/class/gpio/gpioN/value

For the GPIO to be accessed throughout the source code it must contain the following header.

#include <linux/gpio.h>

Then the pin must be allocated and released after use using the following functions respectively. Neither function will work when provided an invalid GPIO pin.

intgpio\_request(unsigned gpio, const char \*label);

voidgpio\_free(unsigned gpio);

The direction and value of the GPIO may be altered using the functions given below.

intgpio\_direction\_input(unsigned gpio);  
 intgpio\_direction\_output(unsigned gpio, int value);

### 4.1.2 - PLL

The PLL clocks on the AM1808 are divided into PLL0 and PLL1.  PLL0 is used as a reference clock for external devices (such as the data converters).  The PLL1 is used in the mDDR/DDR2 controller, separate from the PLL0.  The two types of clock outputs on the microprocessor are the domain clocks (SYSCLK) and the auxiliary clock (AUXCLK).  The AM1808 provides two serial clocks (pins D19 and G19) for Serial Peripherals 1 and 2, DDR positive and negative clocks (pins W8 and W7, respectively), external memory interface (EMA) clock (pin B7), UPP channel A and channel B clocks (pins U17 and G1, respectively), and a video display port clock (pin K4).  Internal clock division and multiplication will allow the AM1808 to interface with different devices of varying clock input requirements.  PLL0 and PLL1 both require a supply voltage of 1.2 V through pins L15 (PLL0\_VDDA) and N15 (PLL1\_VDDA), grounded at pins M15 and M17.

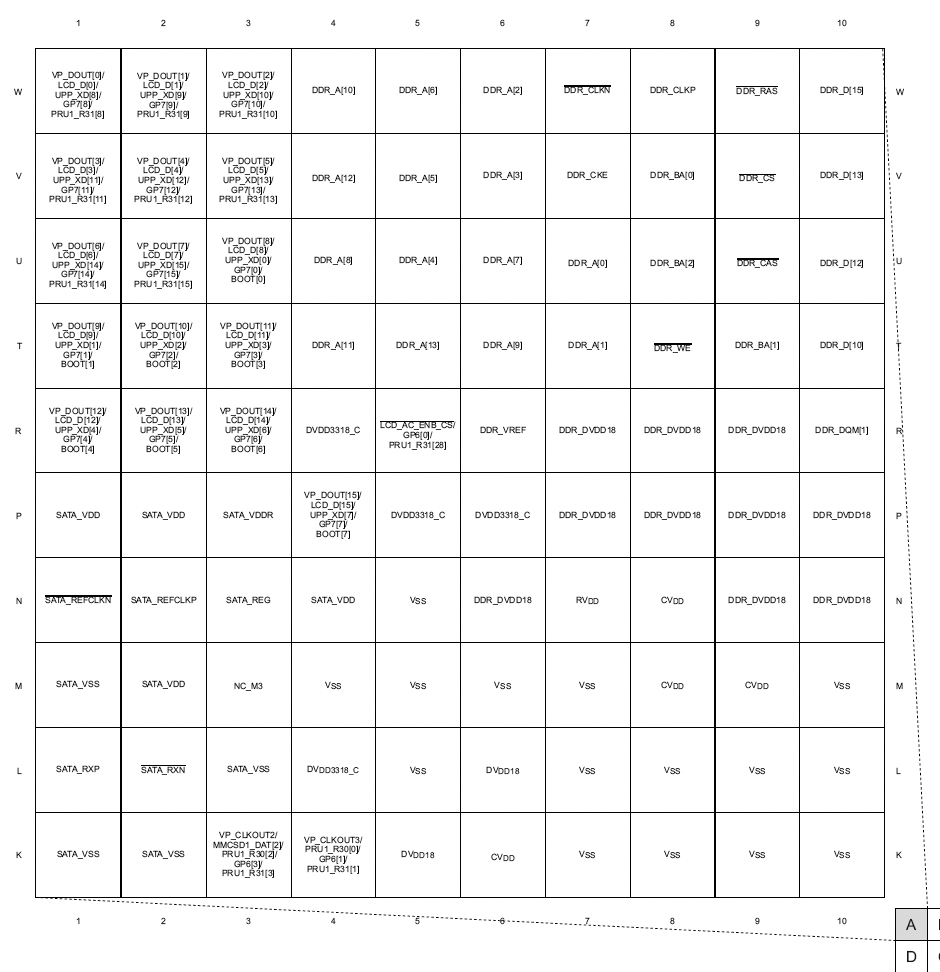
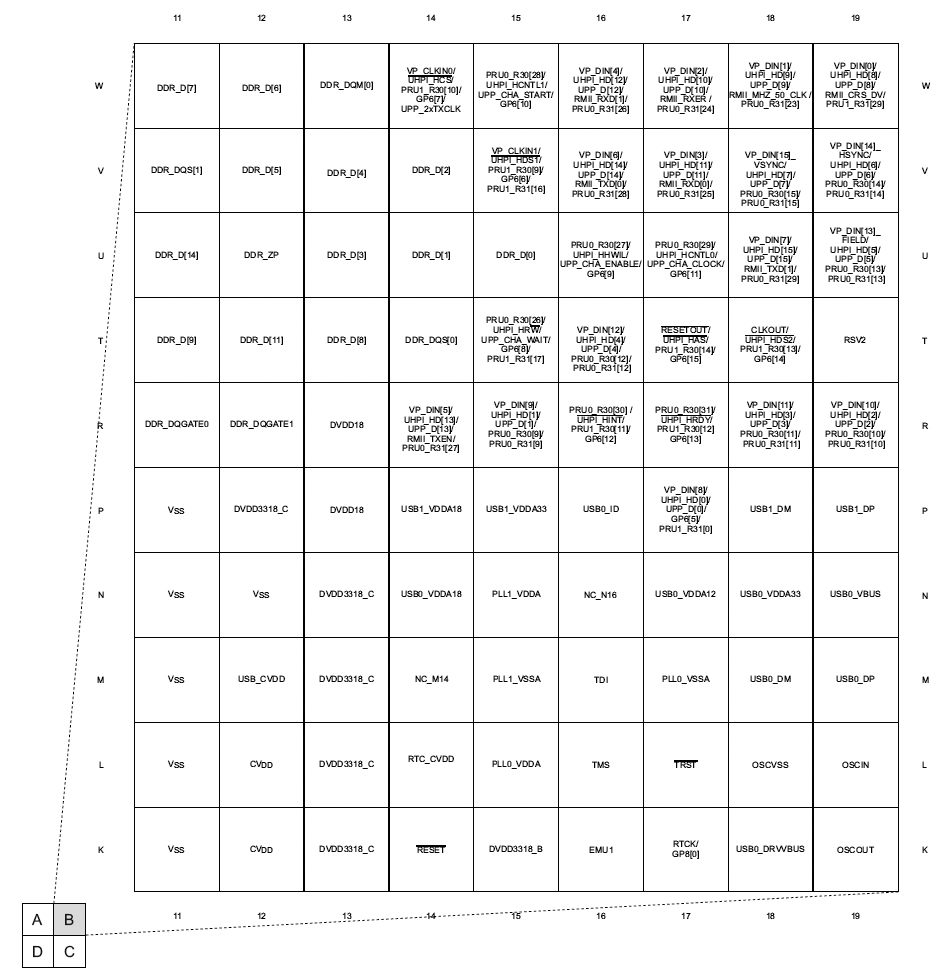
The following graphics show the view of the AM1808 pin assignments in four quadrants (A, B, C, and D).  The ZCE/ZWT pin package is set up as a ball grid array (BGA), which will effect the design of the circuit board.

### 4.1.3 - UPP

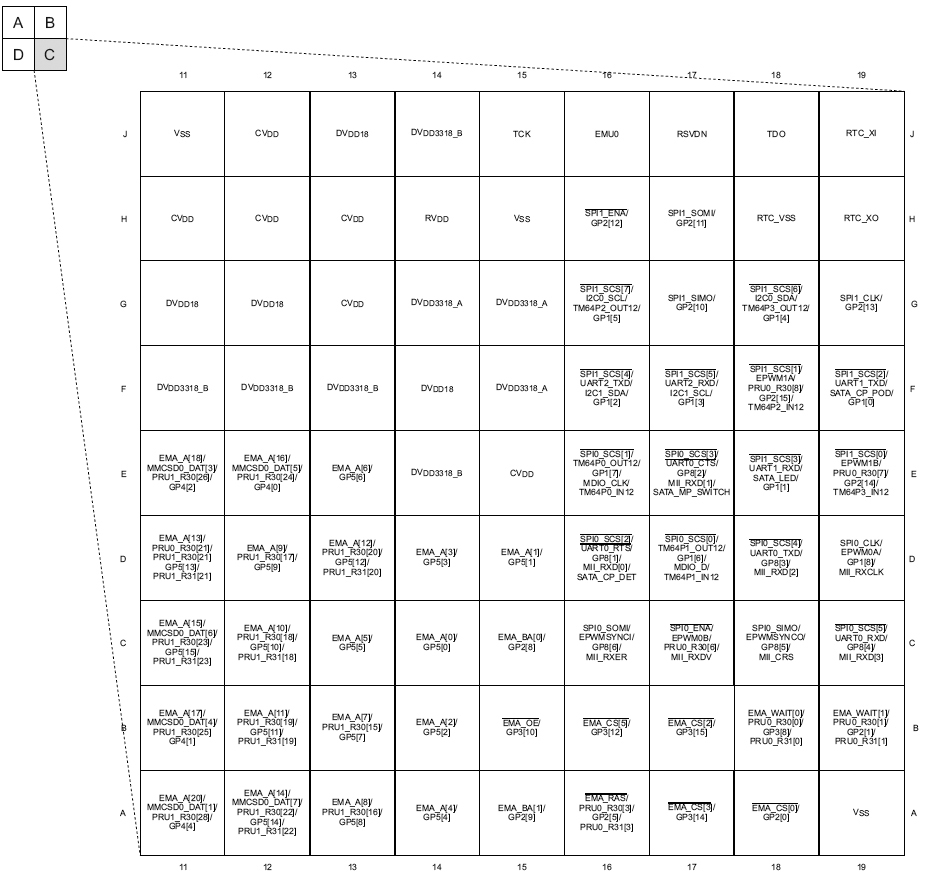
The Universal Parallel Port on the AM1808 will be used to interface with the two data converters in the system.  The AM1808 comes with two channels (A and B) uPP pins.  Channel A will be assigned to the ADC and channel B to the DAC.  Pins U17 and G1 will be linked to the bit clock pins on the ADC and DAC, respectively.  The data from both devices will be sent/received via two of the UPP\_D pins (out of 16).

**Other Peripherals**

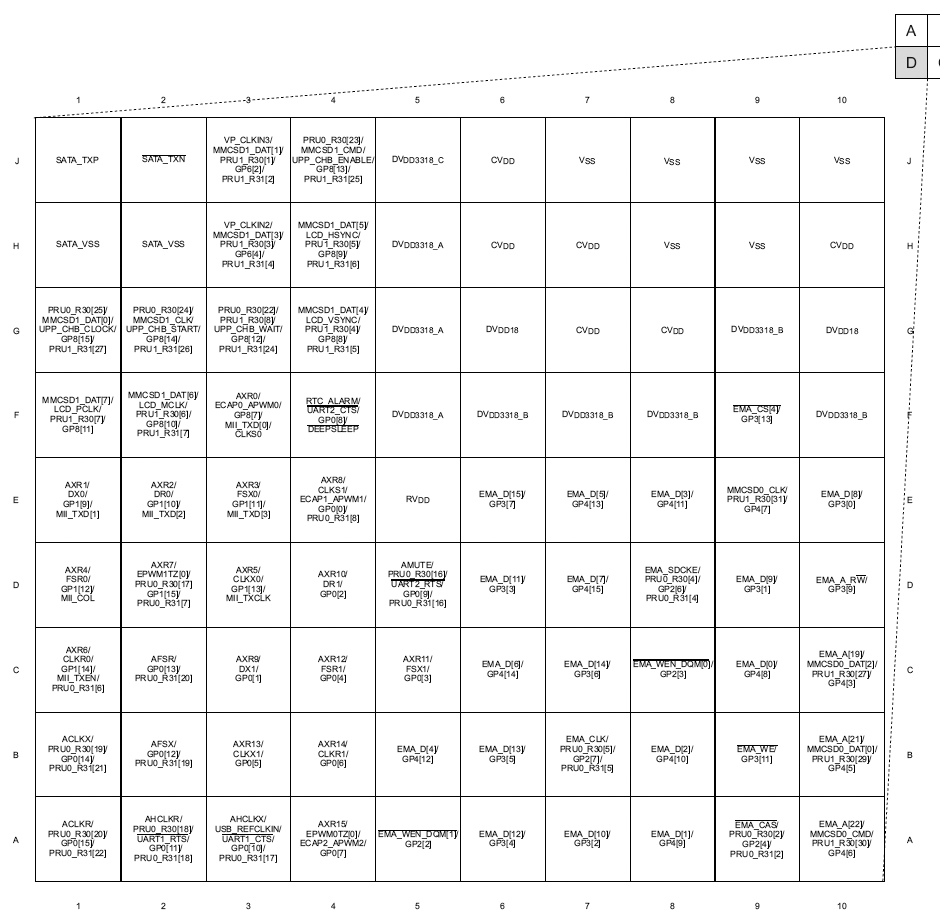
The AM1808 also comes with DDR data bus pins. These will be essential in connecting SDRAM to the board. This will also add to the requirements of the PCB design, mandating that an extra layer be reserved for the DDR ground plane.

**Figure 4.1.2 - Pin Map (Quad A)**

**Figure 4.1.3 - Pin Map (Quad B)**



**Figure 4.1.4 - Pin Map (Quad C)**

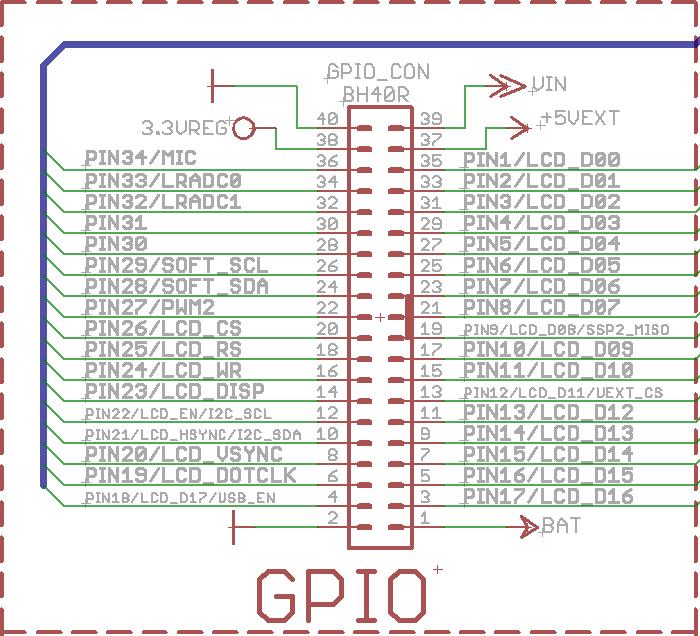


**Figure 4.1.5 - Pin Map (Quad D)**

## 4.2 - OLinuXino iMX233 Development Board

The OLinuXino iMX233 is an open source single board computer that will be used as the main development board during the design phase of CyberChess. It was decided that a development board that contained a ARM9 processor would be best suited for prototyping due to the final PCB environment of CyberChess containing an ARM9 AM1808 processor. The iMX233 fits the needs of this project, containing an ARM926 processor that runs at 454 MHz with 32 KB of on-chip RAM and 64 KB of ROM. The OLinuXino was also chosen as the development board for CyberChess due to its compatibility with a Linux OS environment and its relatively low price compared to other platforms in consideration.

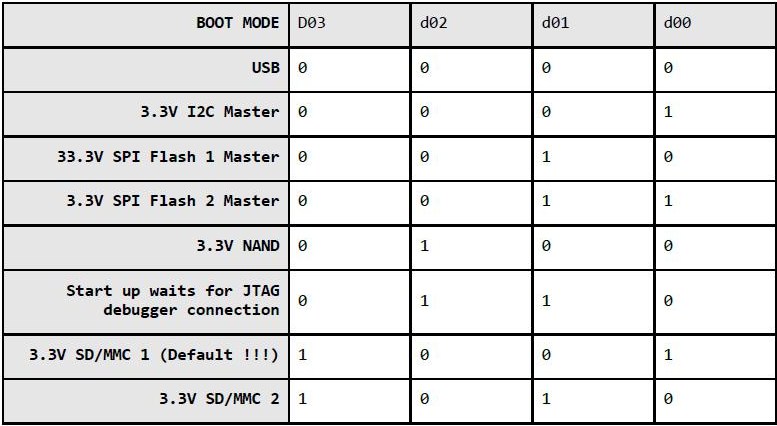
There are two main debugging options available for the iMX233. The first is using the UART interface for debugging and can be done using a USB to serial cable. The second option is the JTAG interface, which will not be considered due to the pins of the JTAG interface being multiplexed with the microSD signal. This leaves the SD slot of the OLinuXino inoperative. The iMX233 comes with 40 general purpose I/O pins that allow the user to add additional hardware as well as help debugging. Each pin can be accessed individually through a female to male jumper or a 40 pin ribbon cable can be attached to access all pins.



**Fig. 4.2.1 - OLinuXino iMX233 GPIO Pin Layout**

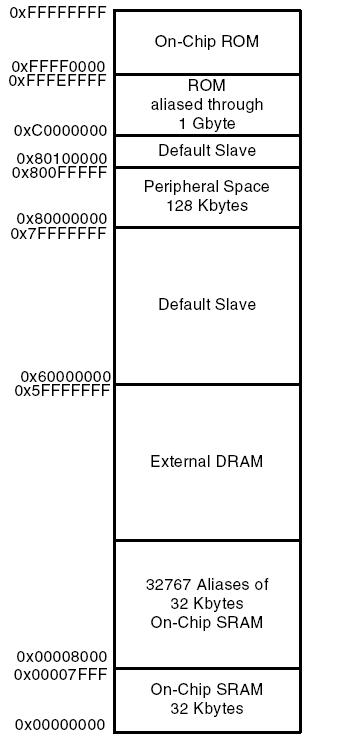
***Permission Pending***

The OLinuXino iMX233 is capable of booting the operating system from up to eight different locations based on the connection of four jumpers located on the bottom of the platform. The default location is set to read from an SD/MMC. Since an SD card was already chosen to be used as memory, no additional soldering will be needed for the iMX233 to boot properly. Below are the boot locations based on active jumpers on the development board.



**Fig. 4.2.2 - OLinuXino iMX233 Boot Locations Based on Memory Used**

***Permission Pending***

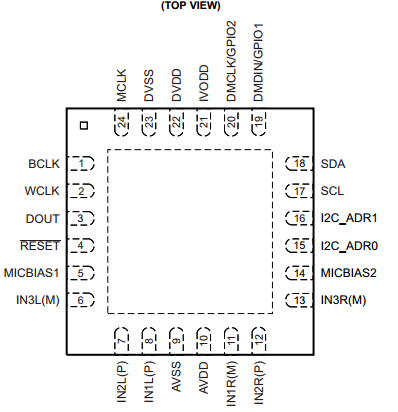


**Fig. 4.2.3 - OLinuXino iMX233 Memory Map**

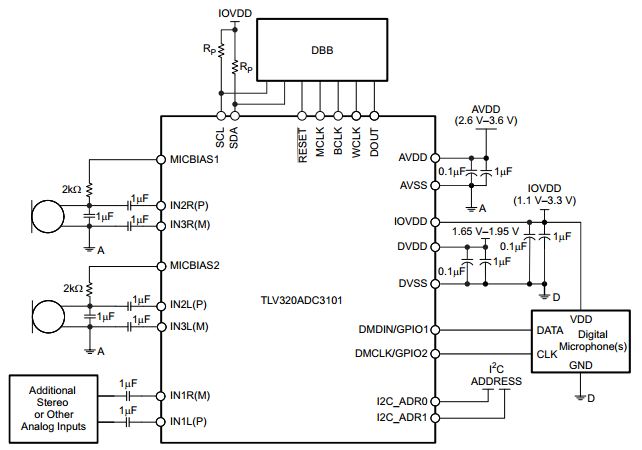
***Permission Pending***

## 4.3 - TLV320ADC3101 ADC

The TLV320ADC3101 is the analog-to-digital converter that will be used in the hardware for CyberChess.  We chose this device because one of the main applications is voice and audio processing.  This is an essential tool in implementing speech recognition into CyberChess.  The TLV320ADC3101 can convert stereo audio signals into digital through six audio inputs.  With this many inputs, and sampling rates ranging from 8 to 96 kHz, this ADC should be suffice for the sole purpose of converting the human voice into computer language.  The TLV320ADC3101 is packed with many digital filters and an analog gain control that allows the user to adjust the EQ and signal strength of the voice input.  Figure 4.3.1 (below) shows the pin layout for the ADC.



**Figure 4.3.1 - TLV320ADC3101 Pin Layout**



**Figure 4.3.2 - Typical Circuitry**

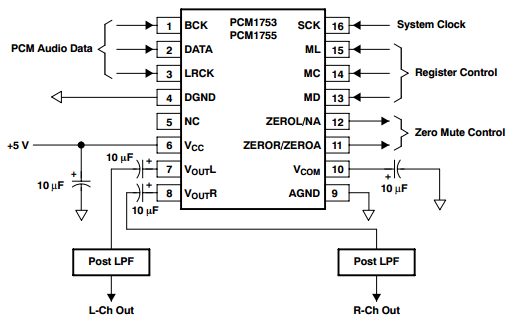
Pins 7 and 8 will be connected through the PCB to a mono phone connector used for audio input.  Pins 11 and 12 will be hooked up to a second phone connector.  These two pairs of pins will be the destination of the voice signals going through microphones white and black (separately) before the signal is converted to digital.  The CPU will be programmed to ignore one channel while listening to the other, and vice versa.  All pins associated with digital microphones or GPIOs will be unused.  This includes pins 19, 20, 21, 22, and 23.  The clock inputs (pins 1,2, and 24) will be connected to SYSCLK outputs on the AM1808.

The TLV320ADC3101 can operate in I2S mode, where all the data is sent through a single pin (DOUT), and the bit and word clocks determine what channel information is being sent.  In standard I2S mode, a word clock low corresponds to information coming from the left channel, and a word clock high corresponds to the right channel.  The period of the word clock is equal to the sampling period.  The audio clocks can operate at frequencies of 8 kHz, 11.025 kHz, 12 kHz, 16 kHz, 22.05 kHz, 24 kHz, 32 kHz, 44.1 kHz, 48 kHz, 88.2 kHz, and 96 kHz.  A reference clock must be provided through either the BCLK or MCLK pins.  The TLV320ADC3101 has an internal PLL clock to generate the appropriate audio clocks from a wide range of MCLK inputs.  The MCLK inputs accepted can range from 512 kHz to 50 MHz.  The following functions show how the sampling frequency is determined based in the PLLCLK\_IN (MCLK or BCLK).

fS = (PLLCLK\_IN × K × R) / (NADC×MADC×AOSR × P)  
P = 1, 2, 3,…, 8  
R = 1, 2, …, 16  
K = J.D  
J = 1, 2, 3, …, 63  
D = 0000, 0001, 0002, 0003, …, 9998, 9999

## 4.4 - PCM1753 DAC

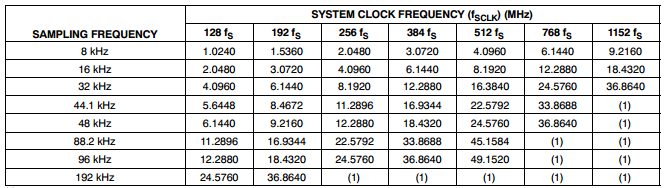
The PCM1753 is a CMOS DAC with 16- to 24-bit audio processing capabilities.  This converter can reconstruct digital data sampled at up to 200 kHz.  With a dynamic range of 106 dB and a three pin serial control port that allows the user access to some internal functions, the PCM1753 is ideal for the sound system that will be implemented in CyberChess.



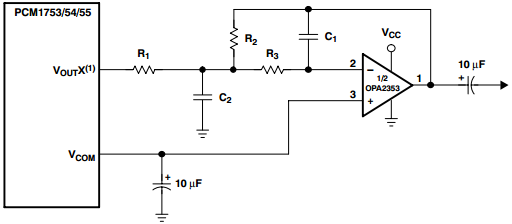
**Figure 4.4.1 - PCM1753 Pin Connections**

The serial interface on the PCM1753 includes the BCK, DATA, and LRCK pins (1, 2, and 3, respectively).  The system clock (SCK, pin 16) should be synchronous with the BCK and LRCK pins. The DATA pin receives the mp3 (or other digital audio format) information from the processor.  The BCK determines the rate at which the bits of data are streamed, and the LRCK determines which channel the data belongs to.  In standard data format, the left channel corresponds to LRCK high, and the right channel corresponds to LRCK low.  The BCK may operate at 32, 48, or 64 times the sampling frequency (LRCK frequency) to determine how much information comes through in one cycle.

The amount of information that can be sent in one sampling period is directly related to the frequency of the bit clock.  However, due to the limitations of clock speeds (in this case, the bit clock), the sampling frequency can only go so high before the amount of information sent per cycle reaches a limit.  Table 4.d.i (below) shows the frequency capabilities of the PCM1753 for various sampling frequencies.



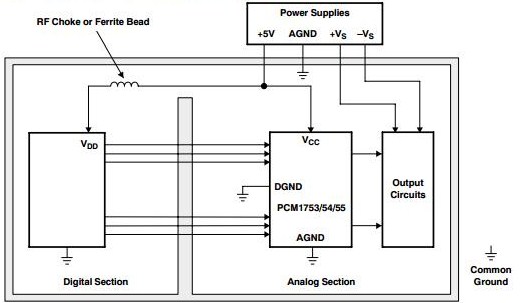
**Table 4.4.1 - PCM1753 System Clock vs. Sampling Frequencies**



**Figure 4.4.2 - PCM1753 Output Filters**

Figure 4.4.2 suggests an output lowpass filter that can be used to amplify and filter each channel while the analog output is sent to the speakers.  The values of the capacitors and resistors in this basic filter are dependent on the electrical ratings of the audio speakers as well as the desired range of frequencies that would optimize the sound quality.

It is recommended that the digital and analog portions of the printed circuit board be powered by different power sources to keep digital switching noise from interfering with analog components performance. If a 5V digital power supply must be used for both the analog and digital sections of the PCB, an inductance may be placed in between the digital supply and analog supply connection to prevent coupling. A RF choke or ferrite bead will work as an inductor in this case. Figure 4.4.3 shows the preferred layout of the PCM1753 on the PCB.

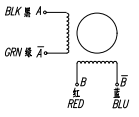


**Figure 4.4.3 - PCM1753 PCB Layout**

## 4.5 - ROB10846 Stepper Motor

The ROB10846 stepper motor was selected to be used as the primary motors to drive the XY-plotter in the movement system of CyberChess. This motor was chosen because it provided the highest torque for the lowest cost. The torque of the motors were taken into consideration due to the additional weight of the Y axis being placed onto the X axis motors. At a max input voltage of 3V and a current of 1.7A, the ROB10846 provides its highest level of torque at 48N·cm. The current required to drive the motor was a key factor in which motor driver would be selected to control the stepper motors of the XY-plotter.

Figure 4.5.1 (below) shows the wiring diagram for the ROB10846 for reference when connecting the stepper motor to the driver.



**Figure 4.5.1 - Basic Stepper Motor Schematics**

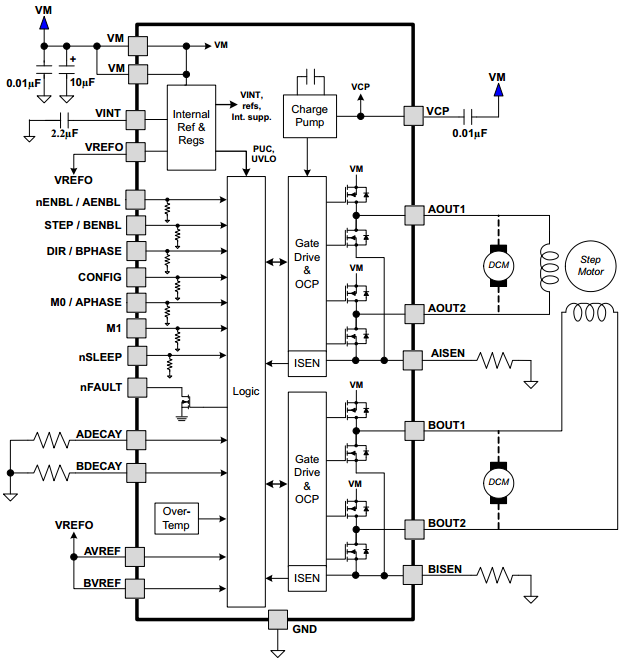
***Permission Pending***

In order to help determine how much space will need in the XY-plotter compartment, it is also helpful to know how much space the stepper motor will be taking up.  The long side of the stepper motor is 48 mm.  The side orthogonal to the rotary arm is 43x43 mm.  A sufficient amount of space will be allowed for the stepper motors in both directions. The drive shaft of this particular stepper motor is 5mm and will affect the decision on which circular gear will be chosen to drive the motors along the vex rack. Each stepper motor will be powered by its own DRV8834 motor driver. To connect the motors to the driver simply take the black and green cable of the stepper motor, they are labeled A and A respectively, to pin 4 and pin 6 of the motor driver. Two other wires must be connected to the DRV8834 in order to drive the steppers, they are the red and blue wires labeled B and B respectively, and must be attached to pin 9 and pin 7 of the motor driver.

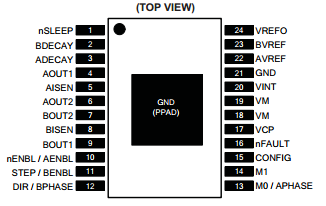
There is also a way to determine the correct wiring of the stepper motor if its wires are not labeled. Begin by plugging in all four wires of the stepper motor into the driver in any arbitrary order and and then send a pulse to drive the motor. If the motor is moving erratically or not at all, switch a wire from phase A with a wire from phase B. If the motor is rotating in the opposite direction than desired, switch the wires belonging to one of the motors phases. This method is applicable to the ROB10846 motor selected for CyberChess but will not be needed due to the motor being properly labeled.

## 4.6 - DRV8834 Stepper Motor Driver

The DVR8843 is a dual H-bridge motor driver with the ability to drive two DC motors or (Lone stepper motor.  Because we are interested in driving two stepper motors with this part, we will need two of these drivers. The driver has two control modes: Step/Direction, with up to 1/32-step microstepping, and Phase/Enable, which allows the driver to drive external references.  We  ofthare mainly interested in the Step/Direction mode.  With 1/32 stepping, we can get a fairly smooth motor movement from the DVR8843

.

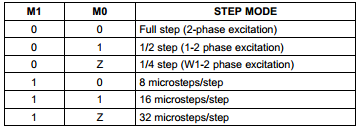
**Figure 4.6.1 - DRV8834 Schematics**



**Figure 4.6.2 - DRV8834 Pinout**

The two figures above show the basic wiring schemes (Figure 4.6.1) under standard operating conditions and pinouts (Figure 4.6.2) for the DRV8834.  In Phase/Enable mode, pins 10 and 11 can be turned on to enable output from AOUT1 (4), AOUT2 (6), BOUT1 (9), and BOUT2 (7).  These outputs go to the windings on the stepper motor.  By sending a high or low signal to BPHASE (12) and APHASE (13), a lag or lead can be created between the A and B outputs.  The switching of phases in windings A and B (see Figure 4.5.1) determine which direction the motor will rotate.  The M1 pin (14) is used to determine fast or slow decay of the stepper motor.  The Phase/Enable mode is intended for input from other devices that are basically pre-programmed to send high, low, and high impedance signals at a rate independent of the DRV8834 stepping capabilities.  In essence, the stepping of the motor is controlled by a source other than the driver, and the DRV8834 is simply a medium to convert logic high and low into driving currents or no currents.

In indexing mode, the DRV8834 operates on one of its stepping modes, determined by the CPU, in a specified direction, also determined by the CPU.  The M0 and M1 pins will each be connected to its own tri-state GPIO from the AM1808.  The digital signal sent through these connections will address one of the 6 stepping indexes in the driver (see Table 4.f.i, next page).  The DIR pin (12) will specify whether the output currents will increment or decrement to the next signal step (note that if the signal through phase A is incremented, the signal through phase B must be decremented; and vice versa).  The STEP pin (11) is used as a clock (not necessarily with a constant frequency).  The output steps are incremented or decremented on the rising edges of the STEP pin input.  The frequency of the stepper motor is directly related to the frequency of the input into the STEP pin.  In 1/32 microstepping mode, there are 238 steps per revolution of the motor.  So, the frequency of the stepper motor is equal to the frequency of STEP divided by 128.



**Table 4.6.1 - DRV8834 Step Mode Truth Table**

The DRV8834 takes a power supply voltage anywhere from 2.5 to 10.5 V (the absolute maximum is 11.8 V).  The maximum output current from the OUTA and OUTB pins is about 2.12 A, at 10.8 V power supply.  To match the rated current of 1.7 A and torque of 48 N·cm in the ROB-10846 Stepper Motor, the power supply voltage must be reduced to below maximum voltage.  The STEP input can be used to control the acceleration of the motor.  However, the motor must have a decent amount of torque to move the weight of the other pieces on the xy-plotter.  During the testing phase, optimal power supply conditions will be determined for each stepper driver (exclusively from each other, because each stepper motor will carry a different weight).

The digital input voltages must be below 0.5 V, and the input highs must be at least 2.5 V.  Therefore, pins 1 and 10-15 will all be linked to 3.3 V GPIOs on the microprocessor.  Pin 15 is the CONFIG pin, which switches the driver mode between Phase/Enable (low) and Indexing (high).

The stepper motors in CyberChess will be run by drivers in 1/32 indexing mode.  So the CONFIG pin on both drivers will constantly receive a logic high.  The nSLEEP input (pin 1) will be used to power up the device after other priority systems have turned on.  The nSLEEP and CONFIG pins on both drivers could possibly be routed from a single GPIO on the processor.  The nENBL pin (10) disables/enables the output stage of the stepper motor without stopping the internal logic processes from the M1, M0, DIR, and STEP pins.  When nENBL is high, the output stage is disabled; when it’s low, the output stage remains enabled.  Because there is no apparent reason to disable only the output stage for CyberChess applications, the nENBL pin should remain low.  Therefore, nENBL will not be connected to anything.

## 4.7 - ROB09064 Servo Motor

The servo motor is a pretty simple part.  It comes with a 3 pin power and control cable. The individual wires/pins are brown (ground), red (power), and orange (control).  The power cable can be operated from 4.8 to 6 V directly from the power supply.  To control the servo, different length pulses must be sent through the control wire.  The ROB09064 will turn 60 degrees in time periods ranging from 160 milliseconds (operating at 6 V) to 200 milliseconds (operating at 4.8 V).  The stall torques at 4.8 and 6 V are 5.2 and 6.5 kg\*cm, respectively.  These operating conditions should be sufficient to support the weight of our primary magnet and the small length of wood that holds it.

The ROB09064 can be operated by sending 5 V pulses through the control cable.  The servo motor changes its angle based on the length of the pulse it receives.  A pulse time of 1 ms will cause the servo motor to turn to its 0 degree (minimum) position.  A pulse time of 2 ms will turn the servo to its 180 degree (maximum) position.  Any pulse time between 1 and 2 ms will turn the servo to an angle proportional to that time.  For the purposes of CyberChess, we are only interested in its minimum and maximum positions.

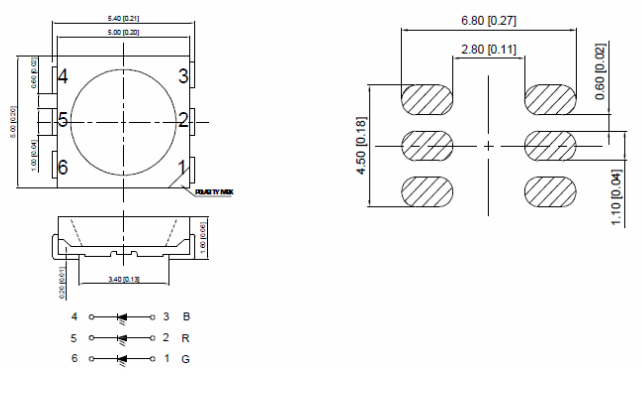
The servo has to be given a pulse every 20 ms to either retain its position or move to another position.  For example, to keep the servo at minimum position for a long time, a pulse of 1 ms must be sent once every 20 ms.  This pulse width control can be accomplished through software run by the microprocessor.  No driver is needed for the servo motor.  The control wire will be linked directly to a single output pin on the CPU.

It is probably safe to assume that CyberChess players will spend a lot more time thinking about their moves than the XY-plotter will spend executing the moves.  Therefore it is also safe to assume that the servo motor will spend most of the gameplay in the release position (the position where the magnet is facing down so that no chess piece can be grabbed) as opposed to the hold position (where the magnet is facing up to grab a piece).  Because the servo motor must be constantly fed pulses, the release position should correspond with the minimum position of the servo motor in order to save power by using the least amount of pulse time as possible.

The ROB09064 dimensions are 41x20x38 mm, which is important in determining the amount of space needed in the xy-plotter compartment.  It weighs 41 g, which is light enough to be supported by the stepper motors which will be controlling its location.

## 4.8 - COM-10866 RGB L.E.D.s

We chose the COM10866 L.E.D.s for their small size and color variety.  The RGB L.E.D. consists of three diodes, one for each of the three main colors.  The forward voltages for each of these colors are as follows: 2-2.5 V for red, and 3.1-3.8 V for green and blue.  The maximum current rating is 30 mA, so the L.E.D.s should be operated at around 20 mA.



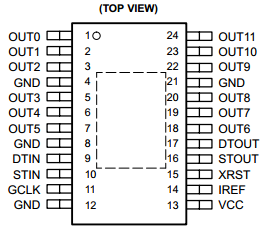
**Figure 4.8.1 - COM10866 Dimensions and Pin Layout**

***Permission Pending***

Figure 4.8.1 (above) shows the pin layout and chip dimensions.  The L.E.D.s are small enough that there will be no problem fitting them underneath our chessboard.  The height of the L.E.D. is important in determining the crawl space between the two glass panes we plan to use.  In order for the L.E.D.s to operate in forward bias, pins 4, 5, and 6 will be grounded, and the outputs from the drivers will be appropriated to pins 1, 2, and 3.  The final CyberChess design will have 96 of these RGB L.E.D.s, one under the center of each square on the top glass pane. This layout will allow the board to direct player attention to any individual square on the board.

## 4.9 - TLC5930 L.E.D. Driver

The TLC5930 is a constant-current sink driver that can operate currents between 0.2 to 40 mA.  This means the driver is capable of running our COM10866 L.E.D.s at the desired current of 20 mA.  The driver has 12 output pins, which are divided into four groups of R, G, and B outputs.  This means that four RGB L.E.D.s can be operated from one driver.  Since we need 81 L.E.D.s, we will need 21 TLC5930 drivers.  The DS-link input allows packet operation so that the TLC5930 can operate with minimal pin use from the CPU.  Three types brightness adjustment functions allow the TLC5930 to change the brightness of all diodes at once or individually.  The device utilizes pulse width modulation to allow 10 bits of gray scale data to go to each output.  This allows for a very wide selection of intensities in each L.E.D., and therefore a lot of color choices in our RGB L.E.D.s.  The gray-scale clock can be operated at up to 25 MHz, internally or externally.



**Figure 4.9.1 - TLC5930 Pin Layout**

The output currents that pass through the L.E.D.s are about 168 times the magnitude of Iref.  Therefore, we can control the constant outputs simply by adding one resistor between the IREF pin and ground.  This resistance is calculated by taking the reference voltage Viref (1.23 V) increased by a factor of 168 and dividing it by the desired output current (20 mA for our L.E.D.s).  This yields anRiref of about 10.3 kΩ. That is the size of the resistor we will be connecting between pin 14 and ground.

## 4.10 - COM-08642 Reed Switches

The COM-08642 is a simple reed switch that shorts a circuit when coming in contact with a magnetic field, and leaves a circuit open otherwise.  There will be 96 reed switches in CyberChess (one for each of the 64 squares on the play space and 16 in each of the two capture zones).  The common end of each reed switch will be connected all into a single 5 V input from the power supply.  The other end on each of the reed switches will be connected to separate GPIOs on the microprocessor.  The rated current of the COM-08642 is 1.2 A.  Assuming the resistance in the switch is negligible, the rated current requires a resistor of at least 4.17 Ω to be connected between each reed switch and the voltage source.  A resistor of about 10 Ω will be used to keep the current well below rated value.

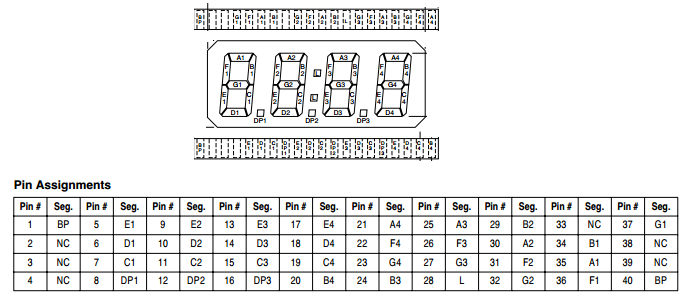
## 4.11 - Memory (SD Card | RAM)

The operating system as well as our code, speech recognition libraries, and additional data will be stored on a 2 GB Secure Digital Card.

The random access memory chip that will be used is a Dynamic 512Mb (32Mx16) 333 MHz DDR2 1.8v chip with an access time of 7.8 microseconds. (Mouser)

## 4.12 - FE0202 Panel Display

The FE0202 is a 4-digit, 7 segment LCD display panel.  All segments will be linked to the corresponding pins on the LCD panel driver.



**Figure 4.12.1 - Digital Clock Pinout**

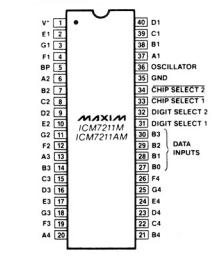
*Permission Pending*

## 4.13 - ICM7211M LCD Panel Driver

The ICM7211M is an LCD 4-digit panel driver designed to interface with a microprocessor.  Using multiplexing, the ICM7211M reads information sent from the processor to the pins (31 and 32) to select a digit (2 bits for a total of 4 digits) and the four data pins (27-30) to write the binary hexadecimal number desired.  The chip select pins (33 and 34) are used to enable the writing in data from the CPU. The ICM7211M has an internal RC oscillator with an oscillation frequency of 19 kHz, which is usually an acceptable frequency for most applications.

An external clock can be connected through pin 36, but is probably unnecessary. The Backplane pin (5) will be connected to four 200 pF capacitors, each connected to four parallel groups of pins; 2-4, 6-20, 21-26, and 37-40.

The pinout is shown on the next page in Figure 4.13.1.  Table 4.13.1 shows how the pins will be linked between the driver and the panel display.  The driver is rated at 5 V supply voltage, supplied to pin 1, grounded through pin 35.  Two of these drivers will run independently of each other in the CyberChess design. Each one will be assigned to one of the two LCD panel displays.



**Figure 4.13.1 - ICM7211M Pinout**

***Permission Pending***

### 4.13.1 - Powering the Colon

The ICM7211 does not have an output to support the colon (pin 28 on the FE0202) on our panel display.  To solve this problem, one extra GPIO will be linked from the AM1808 to pin 28 in both displays when timed games have been selected.

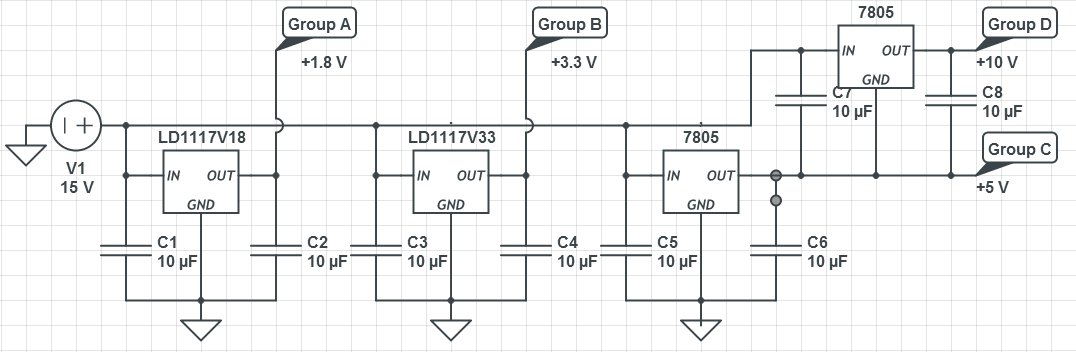
## 4.14 - Power Supply/Voltage Regulation

Table 4.14.i (below) shows the power ratings for all of the active devices on the final PCB.  Devices with ranges of acceptable power ratings have been assigned specific voltages which are consistent with their respective ratings and should be optimal for the performance of the whole system.

|  |  |  |
| --- | --- | --- |
| **Device** | **Voltage (V)** | **Current (mA)** |
| AM1808 | 1.8/3.3 | N/A |
| TLV320ADC3101 | 3.3 | N/A |
| PCM1753 | 5 | 16 |
| DRV8834 | 10 | 4 |
| ROB-09064 | 5 | N/A |
| TLC5930 | 3.3 | N/A |
| COM-08642 | 3.3 | N/A |
| SDRAM | 1.8 | N/A |
| ICM7211M | 5 | N/A |

**Table 4.14.1 - PCB Device Supply Ratings**

In order to accommodate all of the voltage requirements on the PCB, a power supply will be routed through three different voltage regulators: LM1117-1.8, LM1117-3.3, UA7810, and one UA7805.  The main power supply will be 15 V, and it will supply four different power groups: A-D.  The voltage regulators should significantly reduce the ripple from the main power source.  The circuit diagrams below shows how each group of devices will be connected to the power supply.



**Figure 4.14.1 - Voltage Regulation**

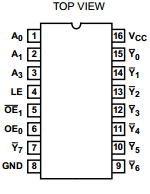
|  |  |  |  |
| --- | --- | --- | --- |
| **Group A** | **Group B** | **Group C** | **Group D** |
| AM1808 (1.8 V)  SDRAM | AM1808 (3.3 V)  TLV320ADC3101  TLC5930  COM-08642  CD54HCT237 | PCM1753  ROB-09064  ICM7211M | DRV8834 |

**Table 4.14.2 - Power Supply Groups**

## 4.15 - Other Hardware

### 4.15.1 - M74HC238 Decoder

The M74HC238 operates at a supply voltage between 2 and 6 V.  It will be supplied in power group B (see Table 4.14.2).  Operating with a supply of 3.3 V, the minimum voltage for input high is approximately 2.5 volts, and the maximum for input low is approximately 0.8 V.  The minimum voltage for output high is about 3.2 V, and the maximum voltage for output low is 0.1 V.



**Figure 4.15.1 - M74HC238 Decoder Pinout**

For the purposes of this project, only pins 1-3 and pin 6 will be connected to the processor.  OE0 (pin 6) is the Enable pin; and A0, A1, and A3 are the pins used to address a certain output (pins 7 and 9-15) to switch to high, while the other outputs remain low.

0 = Low  
1 = High  
X = Don’t Care

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OE0** | **A0** | **A1** | **A3** | **Y0** | **Y1** | **Y2** | **Y3** | **Y4** | **Y5** | **Y6** | **Y7** |
| 0 | X | X | X | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

**Table 4.15.2 - M74HC238 Decoder Truth Table**

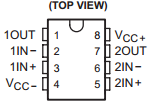
The output pins will be sent connected through each row of reed switches (with every reed switch in that row in parallel with each other, see Figure 6.1.2).  When a reed switch is activated, the output signal will be retrieved by one of twelve GPIOs connected to the other side of the reed switches.

### 4.15.2 - PJ-380 Phone Jack

The PJ-380 is for mono channel microphone input.  It will be directly connected to the input channels of the ADC.  The PJ-380 is rated for 500 V AC, with a minimum of 50 Hz, and also for 30 V DC.  The input resistance is less than 30 mΩ.  There will be two PJ-380 phone jacks in CyberChess, mounted in an accessible location on the enclosure, connected to the PCB by a long wire.

### 4.15.3 - TL082 Op-Amp

The TL082 is a double op-amp chip.  One op-amp will be used in each of the two low pass filters at the output of the DAC (see Figure 4.4.2).  The chip operates at a supply supply voltage of 15 V at Vcc+, with Vcc- being grounded.  Figure 4.15.2 (below) shows the TL082 pinout.



**Figure 4.15.2 - TL082 Pinout**

### 4.15.4 - SPK110 Hobby Speaker

The SPK110 is a 23 mm 8 ohm speaker that will be used to provide audio feedback to the user. This speaker will be connected to the low pass filter that is in turn attached to the PCM1753.  The values of the resistors and capacitors in the low pass filter will be based on the 0.2W rating of the SPK110.

# 5 - CyberChess Software

The software is responsible for making all of the hardware work. A Chess Engine will virtualize the game of chess, allow the computer to know where pieces are at all times, which pieces can move where, and how to know if a player is in check or lost the game. The Board and Movement Engine is comprised of the XY-plotter, L.E.D. Control Systems, and sound systems. This engine is tightly integrated into the Chess Engine and allows for the virtualization to become a reality. Finally the Speech Recognition ties them together allowing a user to give a command to the Chess Engine and having the Board Engine initiate and finish the move.

The vast majority of software within the CyberChess system will be coded in the C programming language. C was chosen over the other investigated options for its fine control over memory management and low-level operations, allowing simple, direct access to hardware without much overhead. Well-written C code tests extremely well in benchmarks, especially for functionality that will be highly important to CyberChess, such as bit manipulation and, again, memory management. This will add a certain degree of additional complexity to the code, as C natively supports very few helpful abstractions, but this challenge will serve only to improve the knowledge of the programmer.

## 5.1 - Operating System

The Operating System (OS) will be Arch Linux ARM. This is a port of the Arch Linux distribution to the ARM architecture which stays true to the Arch philosophy of a simple, controllable operating system. Arch is a full-featured rolling-release distribution of Linux that permits the end user to control the entire system as they see fit. This fits nicely with CyberChess, as we can treat it as a full production operating system during development and later dedicate it to its task of just running the CyberChess software with relative ease.

## 5.2 - Chess Engine

The Chess Engine is a programmatic implementation of the rules of the game to ensure user-requested moves are valid and the game proceeds properly. It maintains an internal representation of the current game state, including piece location, en passant availability, castling possibility, and which color next moves. The engine provides a single C header file, “chess.h”, which guarantees a simple API permitting access to knowledge about the game state and the ability to request a move. The API will additionally support a simple function to compare two board states in order to allow the Board Engine to ascertain whether the current board state is as it should be.

As the project does not require generation of excessive numbers of possible moves, the engine does not necessarily need to be designed with efficiency in this area as a key priority. Even a particularly poorly implemented engine would easily outpace the speed at which users could issue new commands. However, the engine is still designed with the goal of high efficiency, as a learning exercise and to permit flexibility in any future use of the engine for other purposes or projects.

### 5.2.1 - Data Structures

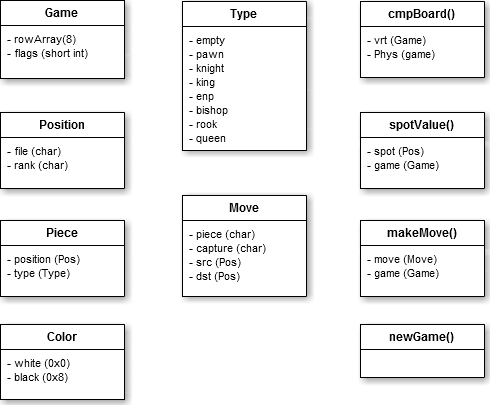
The engine makes use of two key data structures. Game state is contained in a structure composed of a representation of the board and a word made up of various bit flags corresponding to game information. At this time, the board has a space-efficient representation in which each row is contained in a single 32-bit integer value. This gives each spot in the row 4 bits to store piece information, split into one color bit and three piece-identification bits. With this implementation, much of the general control of the board can be accomplished entirely with bit shifting, taking advantage of the relevant speed efficiency. This does have the drawback of having to search the entire board space to determine piece location for such functions as check detection. This will be addressed either with a refactoring of the current data structure, possibly with a complete dismissal of the use of integers for rows, or by compounding the current structure with piece information maintained outside of the board.

Moves are a basic data structure containing a reference to the starting location, the ending location, and information about the pieces in each. This is sufficient information to readily test both the legality and the validity of the move while also slightly improving the efficiency of move execution by having piece information available for manipulation.

Each of these abstractions can readily be transformed to or from a string of text, allowing simple text-based interfacing with the other software within CyberChess. This will allow the various programs to communicate directly, making use of UNIX pipes to directly pass their textual output to the input of another program, reducing internal communication to extreme simplicity.

The board makes use of the 0x88 method for determining valid positions. This was chosen over bitboards for simplicity in writing the engine - the benefits of bitboards will be lost to the fact that large numbers of board states will not often be generated. For the purposes of further experience and experimentation, the engine may at some point be re-implemented with bitboards. If this does come to pass, the two engines will be run through a set of identical tests to determine which really is a better choice for the CyberChess system.

A summary of data structures and methods is shown below in figure 5.2.1.1.



**Fig. 5.2.1.1 - Summary of Chess Engine Data Structures and Methods**

### 5.2.2 - Game Saving and Loading

Upon request, CyberChess will save the current game state to a file which can later be reloaded or read by an interfacing system. The game state will be written to a simple text file stored at a consistent location within the system. Subsequent game saves will overwrite this file, a simple way to ensure that system meets the requirement of being able to load one previous game. The output will be a modified version of Forsyth-Edwards Notation (FEN) which stores the positions of the pieces on the board, the color of the next player, castling availability, en passant availability, and move counts. This notation is quite readable and easy to edit both manually and programmatically, allowing for simple game analysis and design of tests for the system. This output will only be generated and saved at the request of the users, not unnecessarily wasting any processing power on keeping it up to date. Upon a successful loading of the game, the file will be deleted to reflect that there is no longer a saved game to be loaded. The save file will persist beyond the creation of a new game in order to allow players to effectively put one game on hold and start another.

The progress of the current game will make use of a condensed form of Portable Game Notation (PGN), recording each move with Coordinate notation, slightly modified to differentiate between movement and capture, rather than the usual Standard Algebraic Notation. This modification to PGN makes code for future parsing of the file simple to implement as each move is recorded in an incredibly consistent form. This PGN movetext will be updated after each move by simply appending to the file until the game is completed, at which point it will be saved with a time-based identifier allowing for future inspection. By simple parsing of this file, it will be readily possible to have the system replay a completed game as it was initially played.

### 5.2.3 - Timer

CyberChess will implement a simple chess clock that permits users to play either with a time limit on their individual moves, on each player, or on the entire game. With an individual move timer, each turn a player will be allotted a certain amount of time; if they fail to make a move in this time, they will lose the game. Giving each player a set amount of time for the entire game is in line with tournament-style time controls; here, each player must ration their time as they see fit, again losing when they run out of time. The final timer option is a simple timed game, in which the game will end after a set amount of time. This option is mainly for when users want to play only for a set time due to external restrictions and does not necessarily end the game when time runs out - the system will offer to save the game to be returned to later.

Due to the speed difference between human interaction with a chess board and CyberChess’s movement of pieces, the timer will not function quite like a normal chess clock. In a game with timed moves, it will pause during the movement of any piece and then decrement a set amount of time before resuming countdown. For example, in a game where each move is supposed to be given 20 seconds, 5 seconds may be deducted at the conclusion of a piece’s movement and 15 seconds left on the clock when it resumes. This is an attempt to balance the benefits the next player would receive from extra time observing the board during movement.

When each player has a set amount of time for the entire game, the clocks will use a compensation delay, either a simple delay or a Fischer delay, to make up for the time spent moving the piece. A simple delay waits a set amount of time before reducing the player’s clock while a Fischer delay adds to their clock at the beginning of the turn, allowing them to potentially increase their total time by calling out a move quickly. To better facilitate blitz (4 to 15 minutes per player) and lightning (under 3 minutes per player) games, the system may increase the time allotted to the simple/Fischer delay or permit a player to call out a move before the previous move has been physically completed, halting their timer and switching turns at that point.

## 5.3 - Board, Movement and Sound Engine (BMSE)

The Board and Movement Engine is responsible for providing the low-level software interface between the virtual game of chess and the physical game of chess. It contains the functions for moving the XY-plotter, adjusting the L.E.D.s on the board, triggering the use of audio files, and maintaining knowledge of the physical state of the board. It takes input from the Chess Engine and instructs the physical hardware to react appropriately - visually, audibly, and most importantly, with the proper physical movement. The engine can be broken down into three smaller, specialized engines addressing these three main areas.

The L.E.D. lighting system is encompassed inside the Board and Movement Engine and contains all of the functions for manipulating the L.E.D.s. As CyberChess will make use of RGB L.E.D.s, there will need to be support for creation of a wide range of effects. This aspect of the engine will allow for control of the appropriate values for each individual L.E.D., requiring interfacing with the entire array of 96 L.E.D.s.

The audio engine will select the appropriate audio file to be played as necessary. Most audio support will be simply utilize the functionality of the operating system, so the audio engine will simply need to contain a set of conditional statements or a switch-case to determine which file is to be played. To create a more seamless play experience, the engine may be expanded to queue audio files that are relevant to current play, expiring files that remain in the queue for longer than they are relevant to what is going on in the game and prioritizing their placement in the queue based on their importance to gameplay.

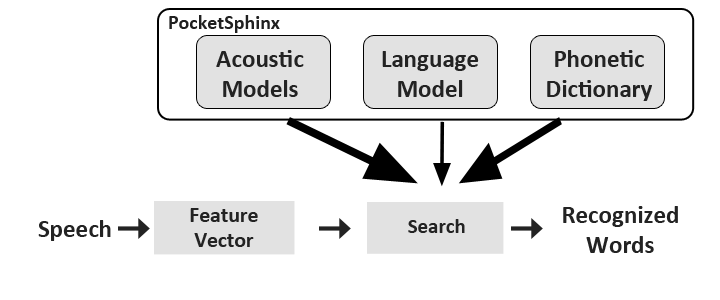
The control of the physical movement of the pieces is the most important area the engine has power over. This portion of the engine will determine the most direct path for movement of a piece from its starting location to its destination and then issue the appropriate signals to have the stepper motors actually execute the move. The movement segment of the engine will also read the state of the reed switches, checking the board state against that stored within the chess engine any time this state changes to confirm that the board has changed from how it should be. Fortunately there are very few cases where human intervention is expected, and they are all at explicit request of the system, so the engine can be designed with the expectation that this check will almost always determine that the board is no longer in its proper state.

## 5.4 - Speech Recognition Engine

The Speech Recognition Engine will make heavy use of PocketSphinx, allowing all spoken commands to be converted parsed to text based on a dictionary of recognized words. This output will be piped to the appropriate program, usually the chess engine, by the main control system.

Our Speech Recognition Engine will be composed of several C files as well as a language model and dictionary file that adapt PocketSphinx to work for our needs.

The main C file for PocketSphinx is called PocketSphinx\_Continuous.c and is shown in block form in Figure 5.4.1. This code will first parse the input speech for model-specific features from the built in feat.params library. The code will then read in a binary model definition file and break down the input into CI-phones, CD-Phones, estimate phones, CI-sen, SEN, and Sen-Seq. These sequences and phones are then compared using memory-mapped I/O for senones and a dict.c file. The possible words and cross words are found and then narrowed down using the language model (.LM) and dictionary (.dic) files.



**Fig 5.4.1 - Speech Recognition Engine**

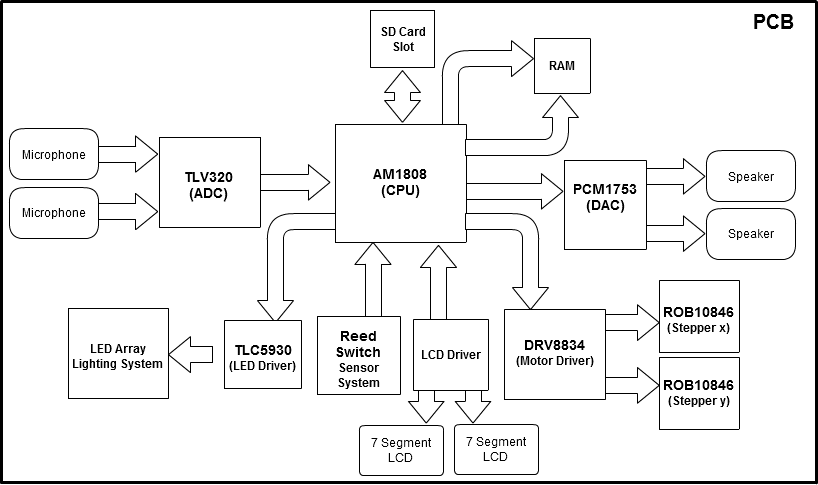
|  |  |
| --- | --- |
| **Command** | **Action** |
| **“[Piece Location] to [New Location]**” | Moves the piece at *PIECE LOCATION* to the stated *NEW LOCATION*. |
| **“[Piece Location]possible moves”** | Lights up the squares that the piece at *PIECE LOCATION* can move to. |
| **“Pause Game”** | Asks other player “Would you like to save the game for later as well?” |
| **“End Game”** | Asks other player “Would you like to end the game unsaved as well?” |
| **Command** | **Action** |
| **“New Game”** | Asks other player “Would you like to start a new game as well?” |
| **“Resign”** | Asks that player “Are you sure you would like to resign?” |
| **“Draw”** | Asks other player “Do you accept a draw?” |

**Table 5.4.1 - Speech Commands**

# 6 - Design Summary

## 6.1 - Hardware

Figure 6.1.1 below is a basic block diagram demonstrating how CyberChess’s hardware will interact.



**Fig 6.1.1 - Hardware Block Diagram**

### 6.1.1 - Movement System

The main goal of cyber chess was to create an automated chessboard that responds to voice commands. The users of cyber chess therefore do not need to move the chess pieces manually. Instead each chess piece will have a thin neodymium magnet attached to its base to be used in both the movement and sensor system. In the movement system the magnets on the chess pieces will interact with another stronger, cylindrical neodymium magnet, 0.5 inch diameter by 0.3 inch height, that lies beneath the chess board and sensor board. This larger magnet is attached to the movement system which is a basic XY-plotter.

The XY-plotter will consist of the following:

* + 2 - 68 oz\*in  (400 steps/rev) Bipolar Stepper Motor
  + 1 - 6.5 kg\*cm Servo Motor
  + 2 - Sets of Toothed Rack Gear
  + 1 - 1.5” Diameter 36 Tooth Circular Gear
  + 3 - 24” Drawer Slides
  + 3 - 1.5”x 4”x 3.5” Pine Wood
  + 1 - 4”x 30” x 1.5” Pine Wood
  + 1 - 22” x 30” x 1.5” Pine Wood
  + 1 - Gorilla Glue

The first step in design of the movement system is to attach two 24” drawer slides parallel to each other on both sides of the 22” x 30” x 1.5” playing board. The slides will provide the y axis direction of motion. This first step is simple but crucial to the project overall. If the tracks are not exactly parallel or if the board is warped the accuracy of the XY-plotter will be significantly lowered and the magnet might not be able to reach all desired boundaries of the playing board.

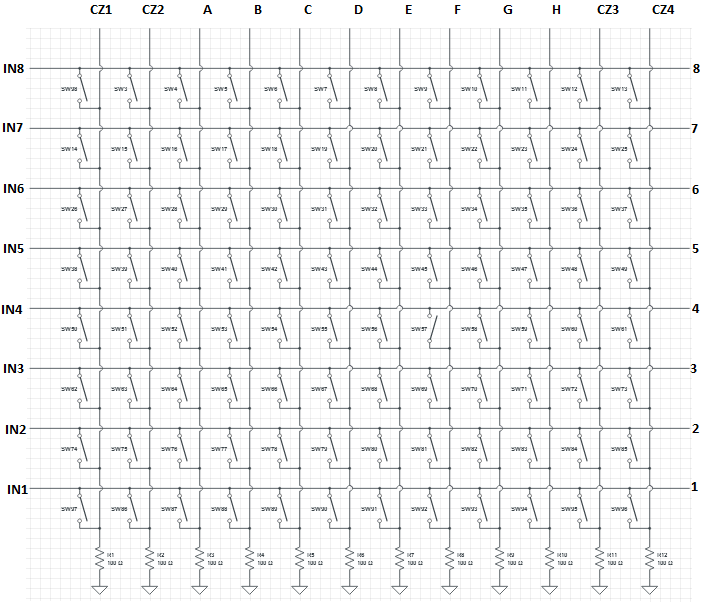
Next the vex rack will be aligned along the inside of one the drawer slides and will be glued or fixed in place. Then a 5mm hole will be drilled in the center of the 1.5” diameter circular gear so it can be placed tightly onto the shaft of one of the stepper motors. The Stepper motor in turn will be glued into place into a cutout section equal to the size of the motor on a piece of 1.5”x 4”x 3.5” wood that will be mounted on top of the drawer slides. The height the motor will be placed is important because we need the circular gear to be properly mated with the vex rack so there is no complications. Next a block of wood equal to the size of the piece the stepper motor is mounted in, is placed on the opposite drawer track to complete the y axis of the plotter.

The design of the x axis is identical to the y axis described before. A piece of 4”x 30” x 1.5” wood and one 24” drawer track  will serve as the range for the x axis of motion. Place more vex rack alongside the drawer track and glue it in place. Mount the second stepper motor in a block of wood as before and secure it to the x axis drawer track at the right height to where the gears mesh as before. A servo motor is also attached on top of the 1.5”x 4”x 3.5”  piece of wood the x axis stepper motor is attached to. This servo motor controls the range of motion for our larger neodymium magnet. The x axis is then connected to the y axis by attaching the 4”x 30” x 1.5” x axis on top of the 1.5”x 4”x 3.5” wood secured to the drawer slides on both ends of the playing board.  The main neodymium magnet will be attached to the wooden arm magnetically by attaching to a metal cap fitted to the end of the wood.  The magnetic attachment (as opposed to glue) will make it easier to remove the magnet if it does not work well with the system. This will complete the basic design of the plotter.

The two bipolar stepper motors will be driven by DRV8834 dual bridge motor drives acquired through Texas Instruments. These motor drivers will be connected to our AM1808 processor and will control the motion and direction of the magnet base. The servo motor does not need a driver and will be connected to the microprocessor directly to control the rotation of the magnet when it is needed.

### 6.1.2 -  Sensor System

To help the detection of illegal moves it is important to know where the pieces are located on the chessboard at all times. As previously stated, each chess piece will be fitted with a thin magnet underneath the unit. This magnet will trigger a reed switch that can sense the magnetic field of the magnet. These switches are placed under the glass playing board onto the sensor board. The sensor board will use the same layout as the 22” x 30” x 1.5” playing board, but have a reed switch wired across each 2.5’’ playing square as well as the squares of the capture zone. When a chess unit is placed in any location on the playing board it will interact with the reed switch below it, producing an on state that can be used to determine the location of the chess piece on the board.



**Figure 6.1.2 - Multiplexed Reed Switch Array**

The setup of the reed switches is shown in Figure 6.1.2 (above).  To reduce the number of pins used, a basic multiplexing technique will be used to read in information from the reed switches.  The input pins (IN1-IN8) on the left of the schematic will be linked to the outputs of a 3-to-8 decoder, which will be linked to 3 GPIO pins off the processor.  All of the outputs (C1-C4 and A-H) will be linked directly back into GPIOs in the AM1808.

To read the position chess pieces on the board, the processor will send a signal through each input, one at a time, and check which of the output pins reads a signal as well.  If the output produces a signal, that means a piece is in that column.  Because only a certain row is sending a signal at a given time, the processor knows exactly what squares are being occupied during that time period.  The reed switch program will run of the clock cycle at a certain frequency.  For each new period of the clock, the internal function will increment to the next input, starting from IN1 and going to IN8, and starting over again.  The reason for using this technique is to save on processor outputs, and to deter players from manually moving chess pieces.

The efficiency of this multiplexing technique is low, because each row can only be monitored one-eighth of the time.  However, it doesn’t matter, because if a piece is moved from its correct position, it will only take a maximum of 7 clock cycles for the processor to see the error.  If a piece is misplaced and then put back into the proper place before the processor can see the error (which is unlikely considering how fast the clock moves in respect to a human), then, effectively, the piece never moved.  Because the reed switch system can not differentiate between chess pieces, a human player could switch the position of chess pieces in such a way that all of the same squares are occupied/unoccupied, and the program will not be able to tell.  However, because the program still thinks the setup is correct, it will assume the physical placement of the pieces is equivalent to the virtual placement.  In effect, all the pieces that are on occupied squares will take on the moving capabilities of whatever piece is supposed to be on that square.

For example, if the program says the white player should have his/her King on E1 and Queen on D1 (both initial positions), however a player manually places the Queen on E1 and the King on D1; it will not affect the virtual gameplay.  Ultimately, the Queen and King will switch roles with each other.  A player can only hope to confuse the other player by physically moving pieces around.  A player can find out if his/her opponent is playing with switched pieces by asking the computer what his/her opponent’s possible moves are off of a particular square.

### 6.1.3 - Sound System

The primary function of the sound system in CyberChess is to provide the users of the game feedback based on voice or manual commands given. It is important that correct feedback is given so this system will depend heavily on the speech recognition function of CyberChess for efficiency. The sounds system will tie in mainly with the chess rules engine and respond when a user makes an illegal move, there is interference with the playing board, or the match is won. The sound system will consist of both input taken from the user and output relayed to the speakers for feedback.

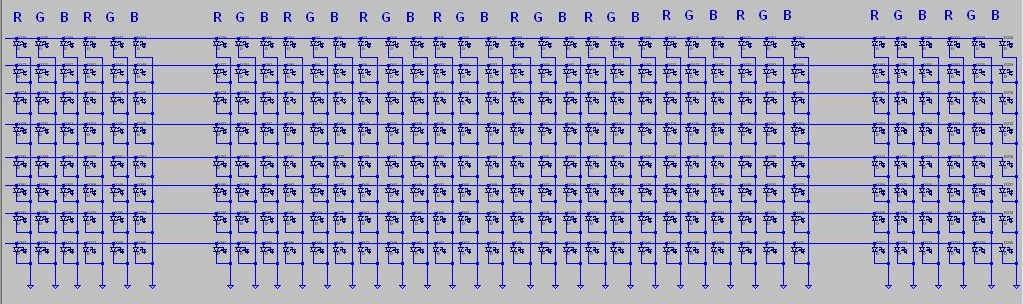
The main input of CyberChess’s sound system is an analog signal produced by the users voice when giving a command. This analog signal is then passed through our A/D convertor, which then gives us a nice digital form of our analog signal that can be parsed into string format to be used in the speech recognition.

The output of the sound system follows a reversed process of the input to the sound system. The microprocessor will take the digital string given from the input and break off to the appropriate feedback that is to be given to the user based on the command. Once the feedback is selected the string is passed through the D/A convertor to be converted back into an analog waveform that can be used to send to the speakers to produce speech.

### 6.1.4 - Lighting system

In order to stay in theme of the project CyberChess will be laced with L.E.D’s around the playing board to give the game a more futuristic feel. The L.E.D.’s are also there for more than just pleasing the eye. They also serve as a guideline how to play chess if the user is a beginner at the game and unfamiliar with the rules. Using the speech recognition, a user will call out a chess piece at a given location ( i.e. PA2 ). The board will then light up locations on the board that mark a valid move for the chess piece. There will be L.E.D’s aligned along the corners of every playing square to make sure the playing board is properly lit. To do this 96 L.E.D’s will  be needed. All of the diodes will be connected to an L.E.D. driver which in turn will connect with the microprocessor.

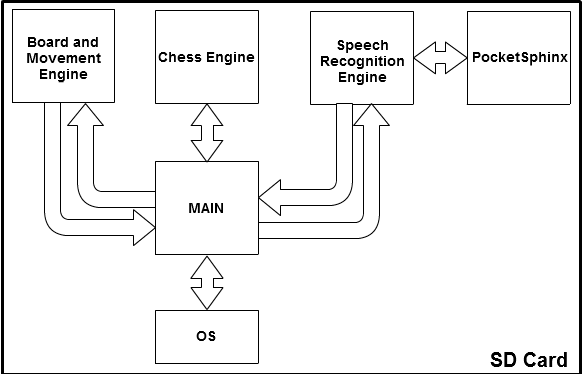
The L.E.D’s will be arranged in a row/column multiplexed array to cut down the amount of drivers needed and in turn the amount of GPIO used by the microprocessor. Figure 6.1.3 shows the array of L.E.D’s in the CyberChess initial design.



**Fig. 6.1.3 - multiplexed L.E.D. array**

## 6.2 - Software

The software for CyberChess is broken down into four main components: main, the board and movement engine, the chess engine, and the speech recognition engine. The board and movement engine oversees all of the physical actions taking place by the hardware. The chess engine is responsible for all of the virtual chess logic and rules. The speech recognition engine is used to control the other two engines, and the main file is used to connect all the engines together into a cohesive software system. The breakdown of this software system can be seen below in fig 6.2.1. Also shown in this figure is the Operating System (OS) block.



**Fig. 6.2.1 - Software Block Diagram**

The main program facilitates communication between the other pieces of the CyberChess software, taking output from each block and routing it to the appropriate destination to maintain a smooth gameplay experience. As the largest part of this program’s facilities involve functionality that can be handled entirely by file redirection and sequential piping of program output, this overarching control may be implemented as a shell script, beginning execution at startup by way of being included in the sequence that is executed each time the system is booted up. The program will utilize a simple infinite loop, constantly checking for output from the various blocks to be redirected to its appropriate destination. This design allows for system shutdown to be handled cleanly and simply as well - the appropriate code can simply sequentially follow the infinite loop, automatically executing upon exit from the loop.

Should it be determined that use of a shell script is too much weight due to the fact that shell scripts have the overhead of being, at the most basic level, sequential calls to built-in functions and external programs, this same structure will be rewritten in C. Helpfully, shell scripts generally closely follow the structure that would be needed to implement the same functionality in a true programming language, and so this rewrite would not be a huge undertaking and, in fact, may be done to improve the system even if it is not deemed strictly necessary.

### 6.2.1 - Chess Engine

The chess engine is responsible for all of rules, moves, and logic involved with the game of chess. This engine takes all of these abstraction and applies them to a virtual space where moves can be made on a simulated board. The important input for the chess engine comes from the speech recognition engine in the form of a textual representation of a move. After a vocalized command is given, it is parsed and passed through the main program and into the chess engine to be computed. If the move is not possible the user will be notified, otherwise the chess engine will carry out the logic and give outputs back to the main, to be sent to the board and movement engine where the physical changes can be controlled.

The chess engine additionally handles saving and loading games, maintaining a game record in one file that will ultimately be archived and writing a textual representation of the game state on demand to another file. This latter file is what will be consistently loaded from any time the system is asked to load a game, allowing for the possibility of allowing users to load custom games by overwriting this file.

The timer will make use of a preemptible infinite loop executing in its own thread to constantly check the time remaining on the clock. This will make use of the time functions available in the C standard library, recording the time at which the clock begins counting down and decrementing the timer based on the difference between the current time and this recorded time. Due to the main program’s similar use of an infinite loop, the timer responsibilities may be pushed up into this main program. If the main program remains as a shell script, this will simply make use of the results of calling the date binary present within any modern Linux distribution. Should it become a C program, it will again use the time functions from the standard library.

### 6.2.2 - Board, Movement, and Sound Engine (BSME)

The board, movement, and sound engine controls all of the hardware-based systems including the movement system, the lighting system, sensor system, and the sound system. The board and movement engine takes inputs from the main program, which interacts with the chess engine and speech recognition, to determine what moves to make, which lights to turn on, and which sounds to send out through the speaker. This engine is also responsible for monitoring the sensor system composed of the reed switches. This information is always being parsed to make sure no pieces have been removed from play by a user.

When the board is turned on, an activation and welcome sequence will initiate with sounds and lights. All of the L.E.D.s will flash and rotate through the available colors while a Welcome Message is played through the speakers.

When it is a player’s turn, their half of the board will light up with their corresponding team color, optionally selected during setup of the game. If a player does not vocalize a move command within the time allotted in a timed game, an audible and visual countdown will occur. The lighting system will light up with their team color five rows up from their first rank and darken one row each second during the last five seconds of their turn. The sound system will also play 5 different tones or a spoken countdown as well.

When a player puts another player in check, a pulsating lighting sequence around the threatening piece will occur and a “check sound” will play through the sound system. The path between this piece and the threatened King may also be lit to aid the player in deciding how to react to the threat.

When a game ends, a specific lighting and sound sequence will initiate. The respective sequence will differ based on how the game ended - checkmate will result in the winning player being congratulated and the lights being focused on their side of the board. A resignation will similarly signal which side has won. In the event of a draw, a more balanced sequence will play, indicating that neither side successfully won the game.

|  |  |
| --- | --- |
| **Output Voice** | **Event** |
| “Would you like to play a timed game?” | At the beginning of a new game, the computer will ask this to the white player. |
| “Please choose the timer mode you would like to use.  Time moves, timed sides or timed game?” | If the player responds “Yes” to a timed game, now the player must state the mode. |
| “Please select the desired time limit.” | Player vocally sets the clock countdown starting point. |
| “Are you sure you would like to move from [Current Square] to [Desired Square]?” | If the move called is valid, the computer double-checks with the player to make sure the player would like to proceed. |
| **Output Voice** | **Event** |
| “Please restate your move.” | The computer says this if the program could not decipher what the player said. |
| “[Current Square] to [Desired Square] is not a valid move” | The computer says this if a player calls a move that is illegal or impossible. |
| “Check.” | Player is put in check. |
| “Checkmate.” | Player is put in checkmate. |
| “Draw.” | Players have a draw. |
| “Are you sure you would like to pause the game?” | Computer asks this if player says “Pause Game.” |
| “Are you sure you would like to end the game unsaved?” | Computer asks this if player says “End Game.” or “New Game.” |
| “Are you sure you would like to resign?” | Computer asks this if player says “Resign.” |
| “Do you accept this draw?” | Computer asks this to a player when the other player says “Draw.” |

**Table 6.2.1 - Audio Out Commands**

### 6.2.3 - Speech Recognition Engine

First the speech will be taken in through the wired microphone provided for the player. This speech will be converted into a digital signal and converted to a feature vector. PocketSphinx and its libraries will take over from this point, providing us with the acoustic models to compare against during the search, to find out which words were spoken. These recognized words will then be plugged into the Speech Recognition Engine’s Move Definer to finalize the move to be done by the Chess Engine. The Chess Engine will take this move and pass its converted instructions to the movement engine and the move will be done.

# 7 - Prototype, Construction, and Coding

## 7.1 - Parts Acquisition and Bill of Materials

The parts for CyberChess will be acquired over the course of senior design I up until the start of the spring semester on January 7, 2013. All of the components needed are in stock and will not require any longer than a week to ship. Below is the list of all products needed to create CyberChess along with the name of the supplier of the component as well as the model number. The majority of these products have already been ordered, and any that have not yet been ordered will be within a month of the completion of this document.

|  |  |  |  |
| --- | --- | --- | --- |
| **Qty** | **Description** | **Supplier** | **Part Number** |
| 2 | 68 oz.in (400steps/rev) Bipolar Stepper Motor | Sparkfun | ROB10846 |
| 2 | Dual Bridge Stepper Driver | Texas Instruments | DRV8834 |
| 1 | Servo Motor | Sparkfun | ROB09064 |
| 1 | 24 Bit Analog to Digital Converter with 92dB SNR | Texas Instruments | TLV320ADC3101 |
| 1 | 24 Bit Digital to Analog Converter with 106dB SNR | Texas Instruments | PCM1753 |
| 1 | 456 MHz ARM9 Microprocessor | Texas Instruments | AM1808 |
| 1 | Single Board Linux Computer | Olimex | OLinuxino-iMX233 |
| 1 | 2 GB Secure Digital memory card | SanDisk | SDSDQM002GB35A |
| 2 | (16 pack) Rack Gear | Vex Robotics | 276-1957 |
| 100 | RGB L.E.D.s | Sparkfun | COM-10866 |
| 24 | L.E.D. Driver | Texas Instruments | TLC5930 |
| 96 | Reed Switches | Sparkfun | COM-08642 |
| 1 | DRAM 512MB 333MHz | Digi-Key | IS43DR16320B3DBLI |
| 2 | 7 Segment LCD display | Purdy | FE0202 |
| **Qty** | **Description** | **Supplier** | **Part Number** |
| 2 | LCD Panel Driver | Intersil | ICM7211M |
| 2 | Phone Jack for Audio Input | Shenzhen Lanho | PJ-380 |
| 3 | 24” Drawer Slide | Gatehouse | LS12224 |
| 1 | 1.5” Diameter 36 Tooth Circular Gear | Vex Robotics | 279-2169 |
| 1 | 36” x 30” x .093” Acrylic Sheet | Optix | MS-06 |
| 1 | 5V Low-Dropout Linear Regulator | Texas Instruments | UA7805 |
| 1 | 10V Low-Dropout Linear Regulator | Texas Instruments | UA7810 |
| 1 | 3.3V Low-Dropout Linear Regulator | Texas Instruments | LM1117-3.3 |
| 1 | 1.8V Low-Dropout Linear Regulator | Texas Instruments | LM1117-1.8 |
| 1 | 3 to 8 Decoder | Sparkfun | M74HC238 |
| 2 | JFET Op-amp | Texas Instruments | TL082 |
| 2 | 23mm Hobby Speaker | Abra | SPK110 |

**Table 7.1.1 - Parts acquisition**

### 7.1.1 Bill OfMaterials

The bill of materials for CyberChess is displayed below in table 7.a.ii. Each component is listed below along with its model, unit price, quantity, and total price. The total estimated budget for CyberChess is XXX.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Qty** | **Description** | **Part Number** | **Unit Price** | **Total Price** |
| 2 | 68 oz.in (400steps/rev) Bipolar Stepper Motor | ROB10846 | $16.95 | $33.90 |
| 2 | Dual Bridge Stepper Driver | DRV8834 | N/A | N/A |
| 1 | Servo Motor | ROB09064 | $12.95 | $12.95 |
| 1 | 24 Bit Analog to Digital Converter with 92dB SNR | TLV320ADC3101 | N/A | N/A |
| 1 | 24 Bit Digital to Analog Converter with 106dB SNR | PCM1753 | N/A | N/A |
| 1 | 456 MHz ARM9 Microprocessor | AM1808 | N/A | N/A |
| 1 | Single Board Linux Computer | OLinuxino-iMX233 | $56.50 | $56.50 |
| 1 | 2 GB Secure Digital memory card | SDSDQM002GB35A | N/A | N/A |
| 2 | (16 pack) Vex Rack Gear | 276-1957 | $19.99 | $19.99 |
| 100 | RGB L.E.D’s | COM-10866 | $0.76 | $76.00 |
| 24 | L.E.D. Driver | TLC5930 | N/A | N/A |
| 96 | Reed Switches | COM-08642 | $1.50 | $144.00 |
| 1 | DRAM 512MB 333MHz | IS43DR16320B3DBLI | $8.47 | $8.47 |
| 2 | 7 Segment LCD display | FE0202 | N/A | N/A |
| 2 | LCD Panel Driver | ICM7211M | N/A | N/A |
| 2 | Phone Jack for Audio Input | PJ-380 |  |  |
| 3 | 24” Drawer Slide | LS12224 | $8.23 | $24.69 |
| 1 | 1.5” Diameter 36 Tooth Circular Gear | 279-2169 | $12.99 | $12.99 |
| 1 | 36” x 30” x .093” Acrylic Sheet | MS-06 | $19.98 | $19.98 |
| **Qty** | **Description** | **Part Number** | **Unit Price** | **Total Price** |
| 1 | 5V Low-Dropout Linear Regulator | UA7805 | N/A | N/A |
| 1 | 10V Low-Dropout Linear Regulator | UA7810 | N/A | N/A |
| 1 | 3.3V Low-Dropout Linear Regulator | LM1117-3.3 | N/A | N/A |
| 1 | 1.8V Low-Dropout Linear Regulator | LM1117-1.8 | N/A | N/A |
| 1 | 3 to 8 Decoder | M74HC238 | $0.95 | $0.95 |
| 2 | JFET Op-amp | TL082 | N/A | N/A |
| 2 | 23mm Hobby Speaker | SPK110 | $0.99 | $1.98 |
|  |  |  | **Total** | $412.40 |

**Table 7.1.2 - Bill of Materials**

## 7.2 - PlexiGlass Chess Board Construction

One of the first things we will have to do is to make the chessboard.  The board will consist of two plexiglass panels of equal area and thickness, one above the other, separated vertically by .75 inches. The design of the encasing (which is described in the next section) will allow for two plexiglass panels to sit in fixed locations, one above another. The plexiglass on top will be the actual chess board, designed with the chessboard pattern, where the chess pieces make contact.  The plexiglass on the bottom is simply there for the reed switches, L.E.D.s, and wiring to sit on.

All of our dimensions were determined by first embedding the magnets in our chess pieces and testing how distance between the chess pieces affected the magnetic forces between them.  We determined the minimum center-to-center distance at which two chess pieces did not cause each other to move to be 1.25 inches, which we decided should be half the distance of one chess square.  No chess piece should ever come closer than half a square to another chess piece during the course of a CyberChess game.  Using our half square distance, we were able to find out that the plexiglass panels need to be about 29x24x0.15 in3.  Since the size of each individual square on the top pane will be 2.5, the total play space should be 20x20 in2.

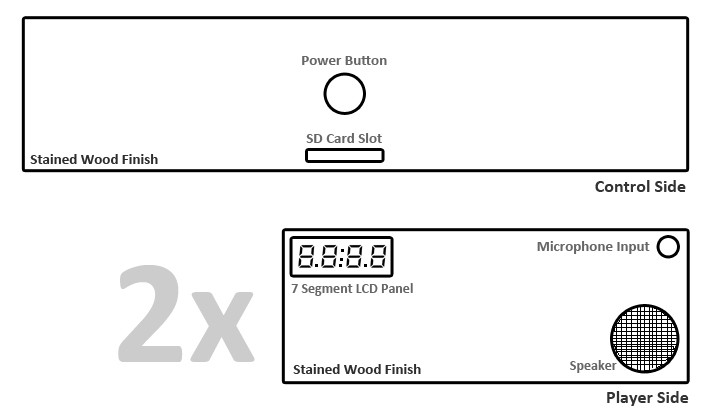
On the left and right side of the play space will be capture zones for captured pieces to move to the side.  We designated enough space for two columns of eight squares (although these squares will not be marked) for each capture zone, which allowed for 5x20 in2 on each side of the play space.  An extra 3 inches will be needed on each side of the plexiglass panel to allow space for the motors beneath the board to move along the edges without touching the walls.  Also one inch on each side will be held by the encasing.  The thickness of each plexiglass panel will be about 0.15 inches, making the plexiglass durable enough without being too expensive.

Both panel of plexiglass will be purchased, with no markings.  We will make the chessboard design on one of the panes using clear acrylic paint. The black squares will be painted, and the white ones will not.  Tape will be laid down along the edges of the intended black squares to stop the stain from spreading into the desired white squares.  Once the paint is applied, the plexiglass panel will be left to dry.

## 7.3 - Encasing Construction

The wood enclosure for CyberChess will need to accommodate our movement system’s hardware so we will begin by measuring the final board play area, capture zones, as well as the movement hardware to be encased. It is important to note that our movement system will be incapable of reaching certain areas, so the board and encasing must be designed with enough space to create an outer buffer of areas unreachable. Once these final volume dimensions are determined wood will be purchased to be cut into the proper sizes. Only the 4 outer walls  and the bottom of the encasing will need to be created as the top of the encasing will be provided by the chess boards.

Measuring a quarter inch from the top of the walls, two half-inch notches will be made across the length of two walls to allow for chess board insertion. These same two walls will then have another notch created a quarter inch from the bottom for hardware insertion and removal. Three, including the two notched walls, of the four walls will be attached in a U shape using angled cuts and nails. The bottom of the enclosure will then be fixed to the U shape using nails and the fourth wall will be attached to the base using hinges. This wall will be easily droppable using a latch locking mechanism at the top sides of the walls. The Player Side panel and the Control Panel are shown below in Fig 7.3.1.



**Fig 7.3.1 - Control Side panel and Player Side panel layouts**

## 7.4 - PCB Vendor and Assembly

The printed circuit board (PCB) will house and electrically connect all of our hardware components. Besides for all of the interconnectivity our board will need an output audio port to be connected to our speaker as well as two audio inputs for the wired microphones. Additionally our board will need a Secure Digital card slot so it can boot and run our code.

Eagle will be the software used to create our PCB design as it has good support from the industry and many tutorials available. It is still being determined whether we will order the board with the components attached or if we will solder the components on ourselves. The main motivation behind having the components soldered on for us is their microscopic size, as it would be very difficult to properly solder our components without the proper equipment. Having them attached by our PCB manufacturer would increase the cost of our board assembly and since we do not have funding, we are going to attempt to solder the parts ourselves.

In addition to Eagle, OrCad and Allegro may also have to be used since we currently can not locate all part layouts for Eagle. OrCad and Allegro will be used to export these layouts into Eagle where the rest of the design will take place. Alternatively we may look into using PCB Artist as it is completely free and is supported fully by our manufacturer.

To actually have the board printed we will use 4PCB.com because they offer special discounts for engineering students. Additionally, they offer 24 hour tech support which could serve to be helpful if we run into any complications.

## 7.5 - Final Coding Plan

Coding the software for CyberChess will be an involved process, ensuring each program operates properly individually and in concert with each other piece it needs to interact with. As such, the coding process will be intimately tied to the test plan detailed below in Section 8.3.

### 7.5.1 - Chess Engine

The chess engine will likely go through a number of iterations (in fact, it has already gone through a few) as the data structures for the game state and moves are refined and relevant code is refactored. As many elements of the engine, such as checking movement validity and executing captures, are rather straightforward to implement, a basic skeleton of the engine will be created for each conceived data structure. These skeletons will then be fleshed out with attempts at the more complicated functions, such as check detection, and the data structures may be further refined as issues arise with these functions. When the engine and relevant data structures are solidly defined in such a way as to minimize such issues, the engine will start undergoing more thorough testing as outlined in the referenced test plan.

The functionality required for saving and loading games should be implemented with very basic code which will need only minor changes based on refining the data structures used. These functions will simply need to respectively create and parse a textual representation of the current game state, using the previously established modified Forsyth-Edwards Notation. Selecting file location to write from and read to is trivial, as is actually writing and reading from this file.

### 7.5.2 - Board and Movement Engine

The board and movement engine must interact with various hardware which may or may not have pre-existing drivers that offer the support required for CyberChess. As such, design and coding of the engine will likely be in parallel with design and production of appropriate drivers for interfacing with the CyberChess hardware. Fortunately, the Linux kernel has a plethora of drivers built into its source code, so, in combination with documentation for the hardware, there should be no shortage of material to aid in designing new drivers to make up for any current lack.

This engine, being responsible for a considerable variety of functions, will be constructed in a piecemeal fashion, with support for lighting, motors, and audio all addressed individually. There is considerable possibility that each subsection will be divided out into its own engine, at which point the board and movement engine will act more as an interface between the rest of the software and these engines than as an engine itself.

# 8 - Implementing the Design

**Figure 7.5.2.1 – Chess Engine and BMECommunication**

**Chess**

**Engine**

**Board and Movement Engine**

## 8.1 - Prototype Plan

Prototyping can serve to be very important in determining what design decisions may need to change. Our entire prototype will be run using our development board and the same Arch Linux ARM distribution we will be using in the final build. The hardware components will be connected via breadboard and for prototyping purposes only, a monitor, keyboard, and mouse will be connected to the development board to allow for interaction.

The prototype will lack the enclosure and will instead be setup similar to a table, with the plexiglass boards on top of a support structure allowing easy access to the movement system and hardware for adjustment and control.

## 8.2 - Build Plan

**Hardware Specific Prototyping**

An area that will require the most tweaking and optimization is our Movement System. This system is comprised mainly of our stepper motors, vex racks, and our motorized magnet. To ensure that this system is as quiet and fast as possible, the team will prototype this system first. Since CyberChess contains two layers of plexiglas in the play area, these two layers must be mimicked in the prototype. These sheets of plexiglass can even be the sheets used for the final project, but they do not need any styling or design. These two layers will be separated using rubber or plastic spacers to mimic the spacing created by our lighting system. Once this play area is acquired it must be secured in place using braces and mounts. The movement system will then be created below it and wired into the breadboard which will be connected to our development boards general purpose I/O. This prototype will allow us observe our magnet strengths and their effects, providing the feedback necessary to tweak and modify the movement system for the better.

## 8.3 - Test Plan

CyberChess is a complex combination of hardware and software built from the ground up specifically for our needs. This means there could arise problems within the hardware or software that need testing or debugging. The environment in which all speech-related tests should occur should be as quiet as possible due to the fact that our input is the users voice. Lighting in the test environment is not very important but should be substantial enough to allow all of the hardware actions to be seen easily. The size of the test environment is also negligible but should be big enough to house the CyberChess unit as well as four individuals.

### 8.3.1 - Software

CyberChess contains many software components that will require individual tests. These tests must also be completed prior to testing the hardware because a large chunk of the hardware tests require working software. Each element of the software will first be tested individually to check that prepared input results in anticipated output. When two pieces that will be regularly interacting are both functional, they will be tested in conjunction with each other to make sure they cooperate smoothly. As each pair proves operational, longer chains of programs will be tested, building up to tests of the entire system as a whole.

**Chess Engine Tests**

As the chess engine operate independently of the speech functions, it can be tested entirely within the production environment alongside its development. An ncurses-based interface has been designed to interact with the chess engine in the production environment. This allows the testers to interact with a visualisation of the virtual chess board, testing alterations to the engine by playing with the game. Conveniently, the ability to save games to file and reload them allows simple testing by way of manually or programmatically generating test cases.

Due to the ease of programmatic generation of tests, the chess engine will be subjected to rigorous testing of checking the validity and legality of moves. The engine should be solidly constructed to the point that, as long as a piece acts properly in any given location, it will continue to act properly anywhere else. Even so, rigorous testing will ensure that this is in fact the case. Additionally, such testing will guarantee that Pawns and Kings, the only two pieces which are subject to different movement rules based on their current location, are thoroughly tested. This testing is simplified by the data structure abstraction of a move, which conveniently allows the test to simply loop through a set of values for each of its fields. Iterating through every possible value for a move structure’s starting position, end position, piece value, and captured piece value results in a total of 172,032 unique tests - significantly more than could ever be checked manually. As such, the expected value of these tests will also be programmatically generated using simple knowledge of the valid moves of any given piece. Smaller scale unit tests and manually created movement tests will be used to determine that the program generating expected values is in fact properly generating these values.

Unfortunately, due to special rules surrounding certain situations in chess, this still will be insufficient. Special cases including check detection, proper handling of en passant, and the validity of castling will all have to be addressed with manually created unit tests. These tests will be performed by way of loading in custom board configurations and short sequences of moves that involve the special rule in question.

**Board and Movement Engine Tests**

The Board and Movement Engine will be tested first in a terminal, ensuring that expected output is being obtained, and then in conjunction with the hardware. Expectations will be determined based on experimentation with the hardware and the engine will be run through unit tests seeking these anticipated results. When the engine seems to be up to par and the hardware is deemed operational, the two will be tested together.

Testing of the Board Engine along with the hardware will address the individual pieces of the engine first on their own, and then together. The lighting portion of the engine will first programmatically be instructed to light each L.E.D. in sequence to ensure that output is handled as expected. Each L.E.D. will be directed to cycle through the set of colors that will be required for the game in a set order to again check that every L.E.D. is being accurately controlled. Once this control has been properly established, more complicated sequences of lighting the L.E.D.s will be tested, culminating in tests of each of the sequences to be used during actual gameplay.

The audio control of the engine will be tested by simply having the system play each of the audio files in sequence. Should the engine be supplemented with the ability to create a priority queue of audio files, various queues will be manually designed and tested to check that files timeout to reflect that gameplay has moved on. Once this system has been shown to be reliable, the testing will advance to having the audio play at the same time as the corresponding L.E.D. sequences. Testing will be deemed successful when all combinations that will be used in regular gameplay have proven to be working properly.

The movement system will likely require the most rigorous testing. Various paths will be manually created and provided as input to the engine in order to check that the system properly moves as required. Following confirmation of this, actual moves will be generated and used as input, checking that the engine properly parses a move into an efficient path. Once this proves successful, testing will move on to bringing the entire engine together - test input will involve moves that the entire system must participate in, such as castling, piece capture, pawn promotion, and a threat to the King. These will all be tested to make sure the proper audio is used, proper lighting sequence is displayed, and these more complicated moves are executed properly.

**Speech Recognition Engine Tests**

The Speech Recognition Engine will be tested by having a test user speak loud and clear into a microphone. To ensure every component of the engine is working properly, the tests will take place in two distinct phases. The first will cover all of the vocal commands using piece location place holders to ensure the command is understood. The next test will cover every possible piece and location configuration to ensure the system understands every possible move.

The commands will be operated in a terminal and documented in the table below with a simple yes if the command was recognized and no if the command was not recognized. The Recognized As column serves as a place to document what was actually recognized by the engine in the case where the output is not what the test user spoke.

|  |  |  |
| --- | --- | --- |
| **Command** | **Recognized?** | **Recognized As (If No <-)** |
| *command speech* | *yes/no* | *if no what was it recognized as?* |

All possible piece and location configurations will then be tested to make sure the engine can decipher every possible combination. The table below shows how this information will be documented.

|  |  |  |
| --- | --- | --- |
| **Piece/Location** | **Recognized?** | **Recognized As (If No <-)** |
| PA2 |  |  |
| PA3 |  |  |
| etc. |  |  |

Upon completion of these tables and tests it will be obvious which commands work and which do not. Using this data we can tweak the Speech Recognition Engine to better recognize the missed commands or the team can implement software specific cases to help remedy the problems.

### 8.3.2 - Hardware

**Basic Hardware and Specific to Voice Commands**

The actions below are initiated via software but use hardware to carry out the actions. These actions make up the core components of chess as well as our project  and will thus require individual testing. Each command will be given to CyberChess and the results will be documented using the table below. A yes or no will be placed into the action complete field, the total time taken will be placed in time, and any additional comments or concerns will be documented in the last column.

The time column of the table is the most important behind “Action Complete?”. From the research done into alternative products and projects, it was glaringly obvious that most people did not enjoy these types of games because they are slow and loud. By emphasizing the time it takes to complete our game actions and working to optimize and minimize them, we can create a better product.

|  |  |  |  |
| --- | --- | --- | --- |
| **Action** | **Action complete?** | **Time** | **Additional Comments** |
| **Start Game** |  |  |  |
| **Move Piece** |  |  |  |
| **Capture Piece** |  |  |  |
| **Castle** |  |  |  |

**Additional Hardware requiring software oriented tests**

While these tests are testing the hardware, the only way some can be done is by writing custom code to run the action. The Lighting system will be tested using a L.E.D. Array Sequence. This code will run through every L.E.D. in the array and turn it on and off. Alternatively, the code could power them all on at once to allow a quick view of which L.E.D.s work and which do not.

The Microphone input will be tested using our Speech Recognition Engine since the microphone is the only thing it relies on. Currently our code when ran will check the hardware components for a microphone and display whether or not it has found it. This code will be sufficient for testing whether our microphone is working or not.

Our sound system, or speaker out, could have custom code written to help with the testing a debugging phase, but could be easily circumvented by merely playing an audio file within linux.

|  |  |  |
| --- | --- | --- |
| **Action** | **Action complete?** | **Additional Comments** |
| **L.E.D. Array Sequence** |  |  |
| **MIC Input** |  |  |
| **Speaker OUT** |  |  |
| **Castle** |  |  |

Testing of our power source will also have to take place and will be done using a multimeter. This information will be documented using the table below.

|  |  |  |
| --- | --- | --- |
|  | **Does it pass?** | **Additional Comments** |
| Power Source Voltage |  |  |

## 8.4 - Extending the Design

### 8.4.1 - LCD Display/Video Out

Currently CyberChess is only able to provide feedback via the lighting system as well as the sound system. Although we currently have LCD screens on both sides of the game, these are only 7 segment displays used for timing options. Throughout the research and design phases we have discussed the inclusion of video options either through the inclusion of an actual LCD display on the control side of the enclosure or through a video out connection like hdmi or composite video. This video display would allow us another form of visual feedback for the user, although it would require the design of a user interface. Deciding to go this route would require us to upgrade our processor or think about the inclusion of a digital signal processor to take a majority of the processing load off of our microprocessor.

### 8.4.2 - Wifi

Adding WiFi connectivity to CyberChess would open up a wide range of additional features. Users would be able to upload game saves to be played on another CyberChess unit, share win and loss data through Facebook or other various social networks, as well as the potential for online multiplayer between two geographically distant units.

### 8.4.3 - Camera

A camera system would allow CyberChess to not only monitor its surroundings, but would provide additional functionality such as win and loss photos. By mounting cameras on both player sides of the unit, the software could be setup in such a way as to take a photo of both parties at the end of the game. These photos would capture the joyful or sad moments created by winning and losing. Using camera system may also used to detect which

### 8.4.4 - AI

An artificial intelligence component of CyberChess would allow the game to be played by one player. Currently, CyberChess is designed for two people. The AI component would add a computer controlled player and would mimic a full software implementation of chess. There are many implementations available for chess artificial intelligence, both open source and commercial. For our purposes, an open source engine would best fit our situation. Chesstools available on Google code is an open source chess engine and AI framework. The AI framework is a compact minimax design that utilizes alpha-beta pruning and transposition tables. It is recommended to use Psyco, a Python extension module, to significantly speed up bot performance.

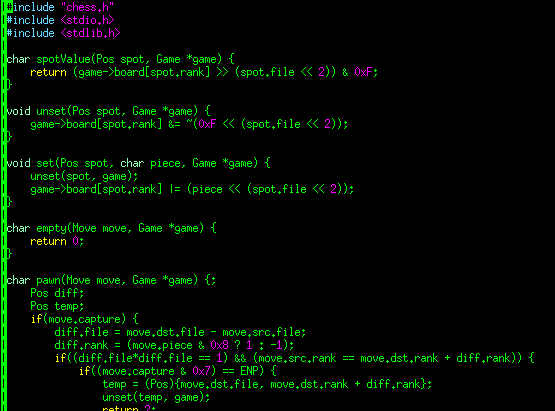
### 8.4.5 - Multiple Game Saves

A simple idea to extend CyberChess would be allowing for the presence of multiple save files stored within the unit. Making such a system feasible would require some manner of easily indicating which game should be loaded using spoken commands. The main issue stopping this from being part of the initial project is determining the design of such a system. Possible solutions involve tying games to individual board locations, lighting up those locations that currently correspond to a saved game on request. In this way, the players could select a saved game without necessitating any expansion of the speech recognition program’s vocabulary.

# 9 - Facilities and Equipment

## 9.1 - IDE and SDK

Basic low-level coding, such as the chess engine, will be conducted within the team’s text editor of choice. Much of the coding has already been done in Vim, a well-known and powerful text editor that has been in constant development for decades now. A screenshot of the Vim environment as well as code specifc to CyberChess can be seen below in Figure 9.1.1. C code is being compiled by the GNU Compiler Collection in a Linux environment and tested therein.

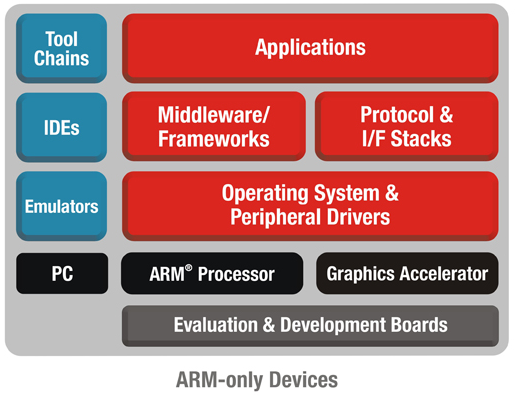


**Figure 9.1.1 - Vim Text Editor Screenshot with CyberChess C Code**

Another One of the main production environments is an existing 64-bit Intel Arch Linux installation, which will allow for a relatively smooth transition to the working environment of the ARM Arch Linux.

Code Composer Studio (CCStudio) is the integrated development environment that the team will be using since it is designed to be used with Texas Instruments embedded processor families. CCStudio is based on the Eclipse open source software framework and provides easy access to compilers for each of TI’s device families as well as a source code editor, project build environment, debugger, profiler, simulators, and real-time operating system. CCStudio has support for both Windows and Linux operating systems.

The Linux EZ Software Development Kit (EZSDK) for Sitara ARM Microprocessors provides developers with an easy way to setup and experience the features of TI’s ARM microprocessors. The EZ SDK includes the following: linux kernel and bootloaders, file system, Qt/WebKit application framework, application launcher, 3D graphics support, integrated WLAN and Bluetooth support, Dhrystone ARM Benchmark, Linpack ARM Benchmark, Whetstone ARM Benchmark, a flash tool, pin mux utility, as well as Code Composer Studio IDE v5. These components are summarized below in Figure 9.1.2.



**Figure 9.1.2 - Summary of Linux EZSDK Components and Features**

## 9.2 - Development Board

The OLinuXino iMX233 requires additional components to power and maintain a proper environment for the development of CyberChess. The power supply for the development board will be a 5V wall wart with a max output current of 1A. The debugging interface of the iMX233 will require a SJTAG interface programmer as well as a monitor and keyboard. In order to be usable as practical production environment, the board will require a Secure Digital card with a Linux image flashed upon it.

## 9.3 - Chess Board

The chess board for CyberChess will consist of two main parts, the plexiglass playing board and the enclosure that will hold the board in place. The playing board will start as a 36” x 30” plexiglass sheet that will be cut down to the playing board size at 30” x 22”. This will either be done by someone employed at a hardware store or will require a special acrylic cutting tool. Next, the chessboard design will be painted onto the playing board using clear acrylic paint. The board will then be placed in a safe location to dry. After drying thoroughly, the plexiglass chessboard is complete. The enclosure for the playing board will be made from wood and will need nails, a hammer, and saw in order to be constructed.

# 10 - Administrative Content

## 10.1 - Milestones

Below are specific deadlines intended to keep progress on design and implementation of CyberChess moving steadily forward such that a working prototype is available for presentation by mid-April. All future dates are, of course, tentative, but will be treated as binding so as to encourage constant work. Past dates are included to demonstrate progress already made.

**October 15, 2012**

*Initial Design and Initial Research*

The overarching idea of the project has been firmly established and initial problems and decisions are being addressed. While no final decisions have been made about parts, hardware or software, the basic ideas of what will be needed and how they will interact are known. Investigation and comparison of the various products available to meet these needs has begun.

**November 15, 2012**

*Research Phase Complete*

While it is recognized that design ideas may yet change to deal with problems or opportunities for improvement that arise during implementation, a basic design has been drawn up. Based on conclusions drawn from research in the previous phase, decisions about specific hardware and software for this design have been made. Both research and the decisions based on it are prepared for documentation.

**December 6, 2012**

*Research and Design (Senior Design 1 Documentation) Finalized*

The documentation is to be turned in at the end of the Fall 2012 semester which concludes on December 6, 2012. Having our Senior Design 1  Documentation completed a few days before gives us time to proofread, make changes, and print the final copy to be professionally bound. Each member of the team is expected to contribute their fair share of the total 120 pages of documentation, which equates to 30 pages per team member.

**January 7th 2013**

*All Parts Acquired*

This date marks the beginning of the Spring 2013 semester when the team will be in Senior Design II. The remainder of this semester will be dedicated to developing the working prototype of CyberChess, and thus materials and relevant equipment should be available so that work can immediately begin. The initial parts, which it is recognized may be replaced as better alternatives are discovered, are outlined in the hardware section of this document.

**January 9, 2013**

*Testing of Individual Parts Completed*

Each part must be tested individually to ensure that it runs properly on its own - should it fail as an individual, there is no reason to expect it to work in concert with the other pieces of the project. This testing should not take a significant amount of time and should be completed rapidly so that any problems may be addressed immediately. Successful completion of this testing will allow for continuation of the build process.

**January 21, 2013**

*Prototype Created*

Using all of the initial parts and the development board the team will create the prototype. This prototype will be used to further develop the hardware and software and test their implementations.

**February 21, 2013**

*Testing Phase 1 Completed*

This milestone marks the completion of the testing phase using the prototype hardware. The ending of test phase 1 means that all of the tests have been passed for both hardware and software. Any test that is failed will be addressed by way of modifying the design and prototype until the test is passed. As such, at the conclusion of this phase, there will effectively be a working development prototype.

**March 7, 2013**

*CyberChess Final Product Created*

After creating and testing the prototype, the final hardware will be created and implemented with the software uploaded to it. This milestone includes the creation of the printed circuit board, the soldering of our parts, as well as construction of the enclosure.

**March 21, 2013**

*Senior Design II Documentation and Testing Phase II*

Upon constructing the final CyberChess product, the Senior Design II documentation will be finalized in order to be presented to the review board alongside the product.

**TDB**

*Final Presentation*

This milestone marks the completion of the final presentation. This milestone can only be reached if every previous milestone in this section is completed. The team’s graduation is dependent on being successful during this presentation. Being successful requires that the product meets all of the specifications and requirements that were initially agreed upon as well as the requirements of the review board.

|  |  |  |
| --- | --- | --- |
| **Date** | **Milestone** | **Met?** |
| 10/15/2012 | Initial Design and Initial Research | Yes |
| 11/15/2012 | Research Phase Completed | Yes |
| 12/4/2012 | Research and Design Documentation (Senior Design 1 Documentation) Finalized | No |
| 1/7/2013 | All Parts Acquired | No |
| 1/9/2013 | All Parts Individually Tested | No |
| 1/21/2013 | Prototype Created | No |
| 2/21/2013 | Testing Phase 1 Completed | No |
| 3/7/2013 | CyberChess Final Product Created | No |
| 3/21/2013 | *Senior Design II Documentation and Testing Phase II* | No |
| TBD | Final Presentation | No |

**Fig 10.1.1 - Milestone Review Table**

## 10.2 Budget and Finance Information

All financial issues with CyberChess will be dealt with group members directly. The total cost of the project came out to $412.40 and will be split amongst all group members evenly.  Using the numbers given by the bill of materials we can determine if costs need to be cut in certain areas. The sensor board and lighting system alone account for over half the cost of CyberChess. One option to reduce cost is reducing the amount of reed switches and L.E.D’s used in the design. The majority of the hardware components used in CyberChess will be able to be sampled from the manufacturer free of cost. This will benefit the group if spare parts are needed. The prices calculated may vary going into the next semester as additional parts may be acquired.

## 10.3 - Summary and Conclusion

CyberChess is a new take on a relatively classical project idea, building on ideas seen in similar projects, supplementing past successes with the so far unique combination of voice control, autonomous movement, and player versus player gameplay. In conducting research to complete this project, the team has learned about a wide variety of topics, ranging from the simple - such as implementing a basic chess board representation - to the complicated - such as fan-out techniques for dealing with PCB data lines. The process has provided new insights about familiar products - experience with Linux has been extended to embedded Linux systems, knowledge about stepper motors compounded by thorough investigation of the various types available.

The team has also learned valuable lessons about the importance of time management in the midst of large projects. Without properly addressing time management, there is simply no way this project could ever come together in time to match the deadlines dictated above. Realizing this, the team intends to consistently commit time to creating the working prototype of CyberChess throughout the semester to come. With proper dedication, creating this system in a timely manner should be a straightforward and educational experience.

In addition to learning to learning to budget time, learning to budget financial resources has been key towards moving forward with CyberChess. As the project is entirely being funded by the team, it is quite important on a personal level to guarantee that associated costs remain low. Thorough research of available options has proven to lead to discovery of cost-effective solutions to meet the requirements and specificatoins of CyberChess.

All of this collaborative effort has had the positive impact of giving every member of the team significant experience with working on a team in which each participant brings a different skillset to the table. Learning to make use of each member’s strengths and cover for their weaknesses has allowed this project to evolve to a state that no individual member ever could have assembled alone. This is excellent preparation for entering a similar environment in the real working world where such collaboration is expected on a daily basis.

All of this learning and new experience is excellent itself, but it is made all the better by having something to show for it. The team is excited to realize the design that has been laid out in this document and looks forward to being able to present a working prototype in April. With a collection of elements that interests each member, CyberChess stands to become an excellent investment.

# 11 - Appendices

## 11.1 - Works Cited

"AM1808 ARM Microprocessor." Texas Instruments, Dec. 2011. Web. 18 Oct. 2012. <http://www.ti.com/lit/ds/sprs653c/sprs653c.pdf>.

"AM1808 Technical Reference Manual." *Texas Instruments*. N.p., Dec. 2011. Web. 1 Nov. 2012. <http://www.ti.com/lit/ug/spruh82a/spruh82a.pdf>.

Bdeakyne. "XY-Plotter." *XY-Plotter*. N.p., 6 May 2011. Web. 15 Oct. 2012. <http://www.instructables.com/id/X-Y-Plotter/>.

Bsrdjan. "OLINUXINO IMX233 Archlinux Upgrade." *Olimex*. N.p., n.d. Web. 18 Nov. 2012.word <https://www.olimex.com/forum/index.php?topic=31.5;wap2>.

"Chesstools - Open Chess Engine and AI Framework - Google Project Hosting." *Google Code*. N.p., n.d. Web. 04 Dec. 2012.

Coombs, Joseph. "Using the AM18xx Bootloader." *Texas Instruments*. Texas Instruments, Aug. 2011. Web. 20 Nov. 2012.

Dominus, Mark. "Why Lisp Macros Are Cool, a Perl Perspective." *Warhead*. N.p., 29 July 2005. Web. 2 Nov. 2012. <http://lists.warhead.org.uk/pipermail/iwe/2005-July/000130.html>.

Etheridge, Darren. "Using Android on ARM for Non-smartphone Applications." *Electronic Design*. N.p., 20 Apr. 2012. Web. 20 Oct. 2012. <http://electronicdesign.com/article/embedded/android-arm-nonsmartphone-applications-73782>.

Excalibur. "Excalibur Electronic Phantom Force Electronic Chess Set." *Amazon*. N.p., n.d. Web. 20 Oct. 2012.

"FPGAs for DSP Hardware." *Radio Eledctronics*. N.p., n.d. Web. 17 Nov. 2012. <http://www.radio-electronics.com/info/rf-technology-design/digital-signal-processing/fpga-dsp.php>.

Gruenstein, Alex. "Speech Input API for Android." *Android-Develoeprs Blog*. N.p., 5 Mar. 2010. Web. 25 Oct. 2012. <http://android-developers.blogspot.com/2010/03/speech-input-api-for-android.html>.

Grusin, Mike. "Retail Servo Landing Page." *Sparkfun*. N.p., 15 June 2011. Web. 15 Nov. 2012. <http://www.sparkfun.com/tutorials/283>.

"ISSI IS43DR16320B-3DBLI." *Mouser*. N.p., n.d. Web. 12 Nov. 2012. <http://www.mouser.com/ProductDetail/ISSI/IS43DR16320B-3DBLI/?qs=sGAEpiMZZMti5BT4iPSEnb3ITE3gp7AYnBuDtXeJRlI=>.

Justicz, Max. "How to Build an Arduino Powered Chess Playing Robot." *Instructables*. N.p., 20 Mar. 2011. Web. 20 Oct. 2012.

Lazaridis, Giorgos. "How Stepper Motors Work." *PCB Heaven*. N.p., 7 May 2010. Web. 13 Nov. 2012. <http://www.pcbheaven.com/wikipages/How\_Stepper\_Motors\_Work/>.

Matuszek, Cynthia, and Brian Mayton. "Gambit: A Robust Chess-Playing Robotic System." N.p., n.d. Web. 1 Nov. 2012.

Rankin, Brett, Paul Conboy, Samantha Lickteig, and Stephen Bryant. "Interactive Automated Chess Game." N.p., 3 Aug. 2012. Web. 2 Nov. 2012. <http://eecs.ucf.edu/seniordesign/su2012fa2012/g04/docs/eel4914.pdf>.

Rouse, Margaret. "NAND Flash Memory." *WhatIs*. N.p., Aug. 2009. Web. 15 Nov. 2012. <http://whatis.techtarget.com/definition/NAND-flash-memory>.

Schroder, Carla. "4 Fine Linux ARM Distros." *Linux.com*. N.p., 5 July 2012. Web. 5 Nov. 2012. <https://www.linux.com/learn/tutorials/598228-4-fine-linux-arm-distros->.

"SD Standards Overview." *SD Association*. N.p., 2012. Web. 10 Nov. 2012. <https://www.sdcard.org/developers/overview/>.

Seibel, Peter. "9. Practical: Building a Unit Test Framework." *Practical: Building a Unit Test Framework*. N.p., 2005. Web. 29 Nov. 2012.

Shust, Mike, and Jeffrey Cobb. "Understanding TI’s PCB Routing Rule-Based DDR Timing Specification." *Texas Instruments*. N.p., July 2008. Web. 29 Oct. 2012. <http://www.ti.com/lit/an/spraav0a/spraav0a.pdf>.

"Swap." *ArchLinux Wiki*. ArchLinux, 2 Dec. 2012. Web. 2 Dec. 2012. <https://wiki.archlinux.org/index.php/Swap>.

Torres, Gabriel. "Everything You Need to Know About Power Supplies." *Hardware Secrets*. N.p., 29 May 2008. Web. 23 Nov. 2012. <http://www.hardwaresecrets.com/article/181>.

"Versions of Decoders." *CMU Sphinx*. N.p., 26 Mar. 2012. Web. 10 Oct. 2012.

"What Is DSP + Analog." *Texas Instruments*. N.p., 2001. Web. 14 Nov. 2012. <http://www.ti.com/corp/docs/investor/dsp/dsp.htm>.

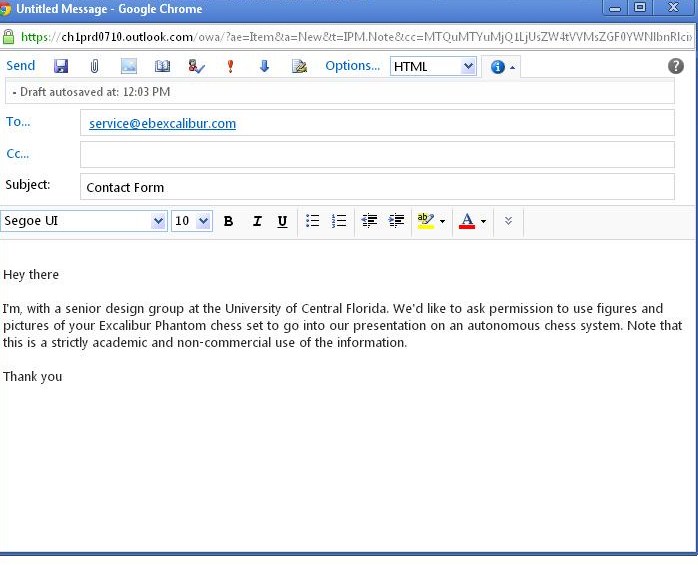
"Windows Speech Recognition." *Microsoft*. N.p., n.d. Web. 02 Dec. 2012. <http://www.microsoft.com/enable/products/windowsvista/speech.aspx>.

## 11.2 - Copyright Permissions

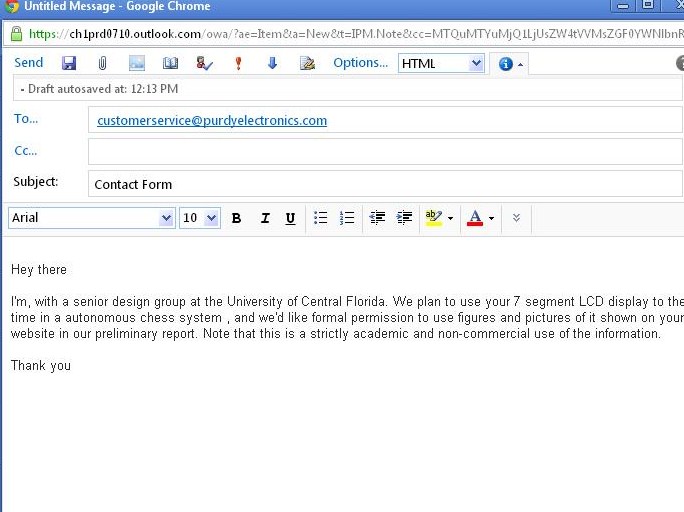
Authorization given by Texas Instruments

From <http://www.ti.com/corp/docs/legal/copyright.shtml?DCMP=TIFooterTracking&HQS=Other+OT+footer_copyright>, It states that “TI further grants permission to non-profit, educational institutions (specifically K-12, universities and community colleges) to download, reproduce, display and distribute the information on these pages solely for use in the classroom.” Since this project is for academic purposes only, we are granted permission.

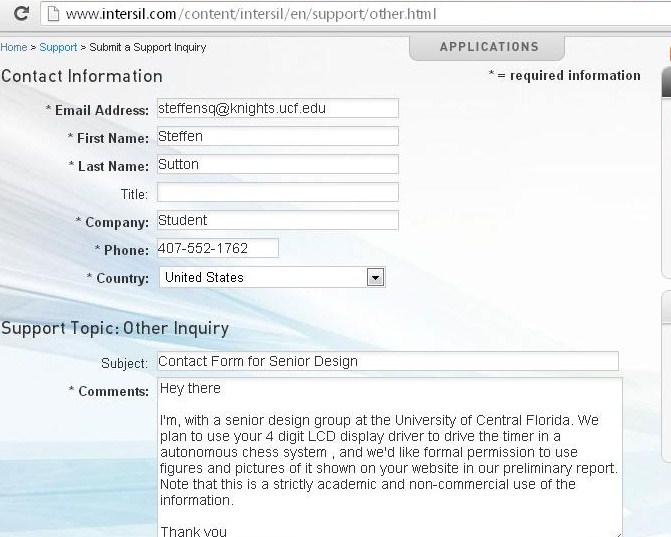
Authorization requested from Excalibur Electronics



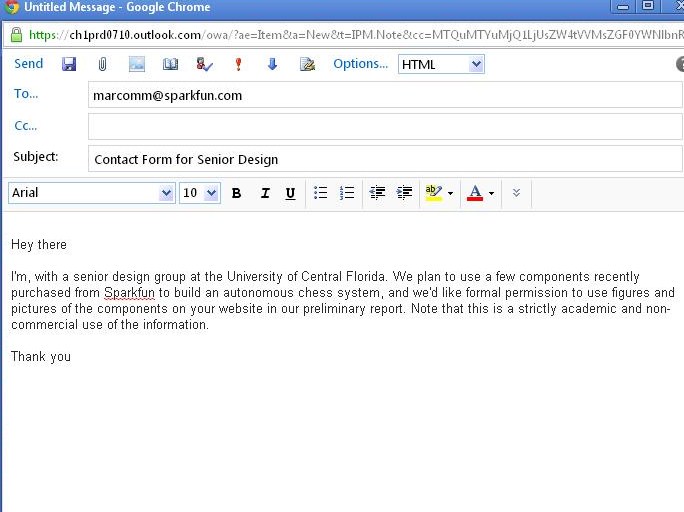
Authorization requested from Purdy Electronics



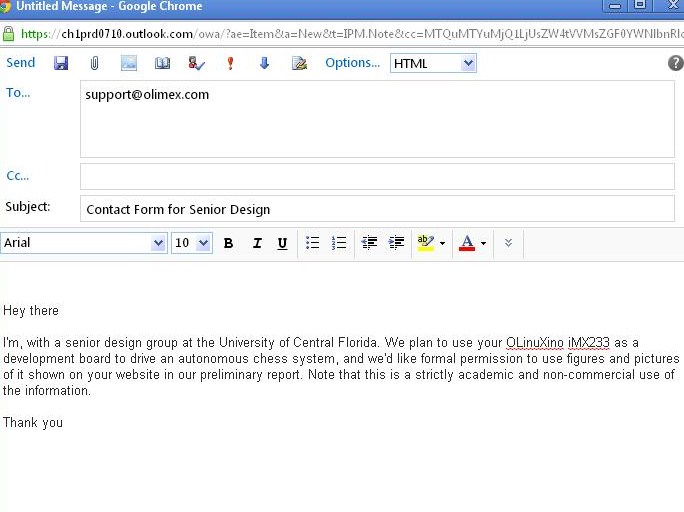
Authorization requested from Intersil Electronics



Authorization requested from Sparkfun



Authorization requested from Olimex



Authorization requested for Raspberry Pi photo

