Creating an Autonomous Chess Robot

A Technical Report submitted to the Department of Electrical and Computer Engineering

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Keenan Alchaar

Spring, 2023

Technical Project Team Members

Nick Cooney

Eli Jelesko

Gabriela Portillo

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Harry C. Powell, Department of Electrical and Computer Engineering

Statement of work:

Keenan Alchaar

My primary roles for this project were developing the custom serial communication protocol and integrating the chess engine into the system. Developing the communication protocol involved defining a custom UART instruction set and writing the code to parse data using these instructions on both the MSP432 and Raspberry Pi. This also involved implementing transmission error handling with Fletcher-16 checksums on both platforms. Integrating the chess engine Stockfish involved developing a Python script on the Raspberry Pi. I developed the Python script which the Raspberry Pi uses to communicate with MSP432 and carry out all its primary functions: move legality checking, game state checking, and move generation. Additionally, I completed some smaller software modules, including the chessboard and PWM modules.

Nick Cooney

I was responsible for the hardware interface. I designed the schematics for the stepper driver, sensor network, switches, buttons, limit switches, debugging LED, and MSP432 pinout. I helped Eli with the PCB layout by resolving the design review errors we had prior to our board sendouts. I also drove the PCBs to and from WWW Electronics for assembly. I wrote the embedded software to read the switches, read the sensor network, and engaged the electromagnet. I also wrote the initial stepper control code, which Eli came back to later to add motion profiling. At the end of the project, I refactored our entire codebase to meet our coding standards. I also wrote nearly all of gantry module, which orchestrates how the robot interprets board readings and messages form the Raspberry Pi to move the motors and play chess.

Eli Jelesko

I designed all the mechanical systems, the circuitry for the power supply, electromagnet, UART, and the signal light in addition to most of the PCB layout. I also designed and implemented the command queue software architecture, much of the stepper motor control and UART on the MSP432 and worked closely with Nick on the rest of the embedded software. The mechanical work involved creating a full robot CAD (Computer Aided Design) assembly. This integrated COTS (Commercial Off The Shelf) parts and custom designed components. From this CAD, I manufactured and assembled the robot's frame, gantry system, supports for sensors, and the sensor board. I also soldered most of the components on both PCBs that WWW Electronics did not.

Gabriela Portillo

I mainly worked on preliminary electrical design work. I started out by designing the general format of how our PCB and microcontroller would connect. When the schematics were complete, I designed a PCB layout and sensor board design. Though these designs were not used as the final design. I later helped with electrical debugging and manufacturing of the sensor network. I soldered the reed switches to the board and made all the connections on the board. I made sure that the PCB documentation for assembly for WWW Electronics was correct including Bill of Materials and some assembly graphics. Later on, I helped with the debugging of the sensor network.

Table of Contents

Table of Figures

Table of Tables

Abstract

This project is a robotic system capable of autonomously playing chess against a human opponent. The key components of the system are a gantry-based structure for retrieving and moving chess pieces around the board, a *Raspberry Pi 3 Model A+* (Raspberry Pi) [1] on which runs an open-source chess engine, an *MSP432E401Y* (MSP432) [2] microcontroller that orchestrates the system, and a custom-designed printed circuit board (PCB) for routing power/data and providing a hardware interface to the player. The system uses a network of reed switches, electromechanical devices that close in the presence of a magnet, (henceforth called the "sensor network") embedded below each tile on the chess board to detect the presence of pieces. In conjunction with a memory-managed record of the board state, kept since the beginning of the game, this allows the robot to identify which pieces are located in which tiles at any point in time, the "board state". Each time the human makes a move, the system scans the sensor network to determine the current board state. This is compared to the previous board state to determine the move that was made, which is then transmitted to our chess engine of choice, Stockfish [3], over a Universal Asynchronous Receiver/Transmitter (UART) data bus. Stockfish responds with a move for the robot to make, which is then performed as a series of motor commands on the MSP432. The notion of this chess robot extends the human condition in that it puts a technological spin on a game traditionally focused on human interaction and competition. That is, it lays a framework of possibilities for the design of autonomous systems that can be expanded upon for generations to come.

Background

Chess is a two-player strategy game dating back to the 7th century A.D [4]. Originating in India, many groups over its long history have become fascinated with its gameplay, leading to the development of variations of the game through time and place. Chess as we know it today was largely shaped during the medieval period, when the game's introduction to Europe resulted in several changes to piece names and movement patterns [4]. One of the first documented references to a "chess robot," a machine capable of playing a human in a game of chess, was a device called the Mechanical Turk in the late 18th century [5]. This feat proved to be a scam, with the device being controlled by a human chess master inside the machine. Nonetheless, the Mechanical Turk marked the first milestone on the road to a fully autonomous chess robot. Since then, numerous advancements in computational complexity, software development techniques, and mechanical design have led to the advent of chess engines, systems capable of engaging in a complete game of chess. While a chess robot might use a chess engine for move generation, the engine itself need not be capable of physically moving the pieces. Much like the Mechanical Turk, we propose a robot capable of engaging in a physical game of chess, as directed by a chess master; however, our master of choice will be a digital chess engine.

In conceiving this project, we were originally focused on the development of a general-purpose articulated robot arm, i.e., a robot arm capable of a 360^{o} range of motion. Over time, we narrowed down our scope to a robot that could physically interact with its environment and its observers. While the articulated design, with its several ball joints allowing for a full range of

motion, would have a more "human" feel to it, we found ourselves limited by the torque capacity of motors within our price range. As we discussed specific applications of the robot arm, we chose the game of chess, and thus adopted a gantry-based approach for simplicity and structural integrity. This gives rise to the final product: an autonomous robotic arm that will use a series of carts on a gantry system to pick up and put down chess pieces and play a full game of chess against a human opponent.

One design consideration for our chess robot was how to handle piece recognition. For the system to be truly autonomous, there must be some mechanism of detecting which pieces are in what location on the board at any point in time. We considered several approaches to this issue, such as an array of reed switches, Hall effect sensors, or near-field communication (NFC) tags embedded in the physical board and chess pieces. There has already been significant work done in detecting chess pieces for robotic applications. One group at Gunadarma University used a computer vision framework to detect piece locations; however, while their system was effective at determining the location of a piece, it required additional information and context clues to differentiate pieces [6]. Another group combined several image processing algorithms to create a robot that could correctly interpret the board and play full-length games [7]. Similarly, their vision system was only able to detect that a piece occupied a square and had to rely on memory to identify which piece it was. While these approaches are potentially more modular (i.e., not necessarily dependent on a specific board/piece set), we felt that similar, if not more robust, functionality could be achieved with an array of sensors embedded in the board. Moreover, we expected that developing a complex image processing or computer vision model, in addition to constructing the robot, would be beyond the scope of a single semester project.

One source of inspiration we pulled from was a robotic art piece, *Can't Help Myself* by Sun Yan [8]. This robotic art piece is a breathtaking intersection of science and art that asks the question "what is the meaning of life itself?" Our original design resembled the robotic art piece until we shifted to the more stable gantry approach after we decided to design the robot to play chess. Other groups who have developed chess-related robots, and from whom we have pulled inspiration, include: Andrew Jakab's group, who made a chess board capable of piece recognition using NFC technology [9], and Josh Eckels from Rose-Hulman Institute of Technology, who built a gantry-based chess robot that required the user manually enter moves [10]. Each of these examples has guided our vision for our autonomous chess robot in terms of their positive and negative features. For instance, we were concerned that an NFC-based sensor network, like that of Jakab's group, would require us to balance the impedances across an antenna array to maintain a clear signal between the chips and the readers. Additionally, since large radio frequency multiplexers do not seem to be common order, we would likely have needed 64 NFC chips and antennae to make such a solution work. Based on preliminary calculations, this would consume too large of a proportion of our budget to make such an approach feasible. Therefore, we decided against the NFC approach. In contrast, Josh Eckels' robot largely resembled our system's design in terms of form; he used a gantry system with an electromagnet-based arm for piece movement, similar to what we had envisioned. However, his approach required the manual entry of moves, and due to the lack of a rigid arm support, led to excessive shaking during movement in the vertical

direction. One goal of our robot was to maintain clean motions, so this research gave use inspiration on how to improve upon previous work.

What makes our robot unique from other projects and previous designs is that our design includes both sensing capabilities for piece detection and recognition, and physical movement of the robotic arm, the goal being a fully autonomous robot. It is worth mentioning that most chess engines can only be run on a device with a fully-fledged operating system and require computational power proportional to the level of performance that is expected of them. For this reason, we chose to use a Raspberry Pi to run our chess engine, as well as to interface with our MSP432 microcontroller and sensor network. The Raspberry Pi's small form factor and access to well-documented chess libraries in Python make it ideal for this application.

This project made heavy use of our knowledge of circuitry and PCB design from the Electrical Fundamentals sequence (ECE 2630, 2660, and 3750), where in each course we designed, printed, wired, and tested a PCB as the final project. These final projects helped us develop the ability to think like engineers as we worked through the designs and debugging processes. Embedded programming from the Embedded Computing and Robotics I and II (ECE 3501 and 3502) and Introduction to Embedded Computer Systems (ECE 3430) courses was also critical to the design of this project, as the entire system was orchestrated through embedded code running on the MSP432. The mechanical design of the project drew from Computer Aided Design (CAD) (MAE 2040) in which members of our group were introduced to and improved upon their CAD skills. Networking protocols from Computer Networks (CS 4457) were also relevant for this project, as we remotely interacted with our Raspberry Pi to prepare the chess engine's environment.

Physical Constraints

Design Constraints

We balanced several constraints to realize the design of our final product. On the PCB side, the manufacturer enforced a limit of 50 vias per square inch. We discovered after contacting the manufacturer that this was imposed as a one inch by one inch moving window, in which no section of the board could contain more than 50 vias, but also no two vias could be less than 0.01 inches apart. Due to the compact nature of our PCB, we had to pay close attention to this limitation in several areas, specifically near integrated circuit (IC) chips and thermal dissipation pads. On the computational side, our MSP432 came equipped with 256 kB of ram. While this is sufficient for our embedded code, it would have been incapable of running the chess engine. This was why we have the separate Raspberry Pi for move generation. Several times during development, the MSP432's default stack and heap size constraints led us to hard-fault our program due to memory collisions. Our approach to modifying these settings was dependent of the development environment we used (see below). Finally, parts availability was a recurring issue throughout this project. In particular, the limited supply of certain IC's like stepper motor and electromagnet driver chips all but forced our hand on which components to use.

Cost Constraints

The original budget of $500 was the cost constraint for this project. Budgeting required preliminary research into mechanical and electrical parts, as well as an assessment of materials already owned. Shipping was not a concern for this project and was not factored into the budget. When designing the PCB, the main design consideration was minimizing size, which in theory would also minimize cost. Therefore, we chose to use predominantly surface mount components to obfuscate the minimum board area necessary. What we failed to account for was the magnitude of cost for PCB assembly. Each component that was to be assembled cost 50 cents, but with over 200 components on our board, assembly cost much more than we expected. Cost was also a major consideration when choosing the board's components, as we aimed to use the cheapest components that would be sufficient for the job at hand. Cost, in tandem with ease of use, was also a consideration when designing the sensor network. There were multiple options that we could have gone with, such as NFC tags, Hall effect sensors, or reed switches. Each chessboard square needed some sort of sensor device, so the notion of purchasing 64 sensors (at minimum) lead us to choose reed switches (of which we could obtain 100 at a unit price of $0.4772 each) over the other options. Hall effect sensors generally had a high unit price and would have required more magnets for piece recognition. NFC tags would have required up to 64 readers as well, which would have been too expensive for our overall budget.

Tools Employed

Completion of this project relied on several useful tools for software development, mechanical CAD, and electrical CAD. Each of the tools are described below, grouped by function.

General

GitHub [11] was used as version control during development for all software, electrical, and mechanical files. There were four primary repositories: electrical designs, mechanical designs, MSP432 software, and Raspberry Pi software. The developers were already familiar with GitHub, so its integration and use were quite straightforward.

Electrical

The main tool used for the electrical design was KiCad [12]. KiCad was used to design the electrical schematics and the PCB layout. To better view the electrical files on GitHub, CADLab [13] was used and integrated with GitHub. KiCad was a new tool for the electrical designers. The main reason KiCad was used was that it is open source, and therefore does not require purchase of a license. KiCad is also well developed and integrated with electrical part distribution sites like Digi-Key and Mouser for easy importing of part-specific footprints. KiCad's calculator tool was also used in aiding the design of the PCB. In particular, the trace width calculator helped us determine the minimum trace width that could handle our current and power projections without overheating.

Mechanical

All mechanical designs were made using Solidworks CAD (Computer Aided Design) software [14]. The aluminum frame was cut to shape using a horizontal band saw and the sensor board was cut using a water jet. Many of the specialty parts were made using a 3D printer in Polyactic Acid (PLA) plastic using a Prusa i3 MK3S+ [15] or an Ultimaker S3 [16]. All parts for

the Prusa were sliced with PrusaSlicer 2.5.0 [17] while those on the Ultimaker were sliced with Ultimaker Cura 5.1.1 [18].

Software

Code Composer Studio (CCS) [19] and Visual Studio Code (VS Code) [20] were the two editors of choice for developing code for the MSP432. CCS was particularly useful for flashing code onto the MSP432 and debugging via its access to the MSP432's registers. The developers were familiar with both of these tools and encountered no problems during their use.

Societal Impact Constraints

Environmental Impact

The largest environmental impact of this project comes from producing raw materials, especially the aluminum and the silicon wafer of the PCB. Aluminum is extremely energy expensive to produce. Once the bauxite ore has been mined, it must be refined and then smelted, which requires about 68 MJ/kg of energy, usually in the form of electricity. Depending on the source of that electrical energy, this produces around 0.45 Gt of CO_2 [21]. Since aluminum is so costly to produce, it is often recycled, or in our case reused. Since we largely built our frame from parts off of previous years' capstone projects, our particular impact here is minimal. As for the PCB, both its manufacturing and recycling involve some rather toxic processes. When a PCB is made, it is subjected to several rounds of chemical baths, both acidic and alkaline. The waste products of these baths contain fluorides, phosphorus, cyanide, and heavy metals. These are usually treated on site, but some are released as sewage or put into landfills [22]. Not only are PCBs environmentally costly to manufacture, but they are also costly to recycle. Since most PCBs are single use, they will eventually become e-waste. When this happens, they are shredded, smashed, and then what metals can be reused are chemically separated from the dust. This dust contains mercury, cadmium, lead, and arsenic alongside the valuable copper and nickel. These toxic metals need to be appropriately disposed of to prevent poisoning the local ecology, but in lesser regulated countries like China this is not always the case. This process also releases several toxic gasses, such as dioxin and lead fumes, which are harmful to human health [23].

Sustainability

Defining sustainability to mean "the ability to reproduce for years to come," this project's design is dependent on several products to remain in production in order to feasibly manufacture without significant design changes. The PCB was designed for the general purpose input/ouput (GPIO) header on our specific MSP432. Therefore, if another microcontroller were to be used, a redesign of the PCB would be necessary. Other components of this project, like the Raspberry Pi, could be replaced with little to no effect on the design.

Health and Safety

Safety was, and still is, a primary concern. As our project involved large moving machinery, great care was taken to avoid hitting, crushing, pinching, or otherwise damaging human limbs. This involved designing the piece grabbing mechanism such that it could not forcibly drive into the playing surface and crush someone's fingers and ensuring the "arm" of the robot is out of the

way when the human player is making their moves. Potential pinch points are denoted by stickers on the frame, and nearby limit switches ensure the robot will stop moving if it hits an unexpected object, like a human limb. An emergency stop (e-stop) button can be pressed to immediately halt the robot's movement until a manual power cycle of the system has been performed. This is accomplished by purposely triggering a hard-fault in software so nothing is able to run unless manually restarted.

Ethical, Social, and Economic Concerns

With the target audience of this project being people who do not have a companion with whom to engage in a game of chess, the high price tag of mass manufacturability makes this robot a high-end product. This realistically only puts the machine in the hands of those with large disposable incomes, despite the envisioned audience of everyday consumers that wish to play chess in real life without a human opponent. Chess purists may have some concerns that turning the game robotic removes the beauty of the sport, but we argue that our design introduces a novel spin that can garner interest in the game. Another concern with our design could be cheating. Fortunately, the sheer size of our robot makes it impractical to cheat in tournaments, and online chess engines can already be used to cheat elsewhere.

A more general social concern would be the rise of automation. Historically, with the rise of automation, there could also be a risk of job loss for sectors of industry in which manual labor is no longer required to perform most tasks. Examples of this include the rise of automation in the automobile manufacturing industry, and store cashiers being replaced by self-checkout stations. Though with this specific application, it is unlikely that chess grandmasters would be out of a job due to this robot, as the chess engine software is freely available online. Instead, this robot is intended solely to serve as a fun and exciting spin on a classic game.

External Considerations

External Standards

Chess

The Universal Chess Interface (UCI) [24] standard is an open-source protocol for communication between a chess engine and chess interface. One benefit of the UCI standard is that any UCI-compliant chess engine can be interchanged with any UCI-compliant chess interface. In our case, we used Stockfish as the UCI-compliant chess engine, making our system itself a a UCI-compliant (robotic) chess interface.

Electrical

The main electrical external standards this project had to deal with were the Institute for Interconnecting and Packaging Electronic Circuits (IPC) PCB design standards [25]. IPC is an organization that collects and votes on standards for PCB development and manufacturing. IPC standards apply at each step of the development process, from determining trace widths to laying solder on the board. For instance, IPC has standards for file formatting, use cases of various electronic components, PCb assembly, etc. Specific to this project, adhering to PCB manufacturing standards was critical to ensuring a physical realizable board. We used FreeDFM [26] to ensure our prototypes passed a design for manufacturing check, which checks that designs follow the IPC

standards and are feasible. To meet our manufacturer's design constraints, the PCB had limitations on the maximum via density, number of layers, and total area of the board.

For communication between our MSP432 and Raspberry Pi, we rely on a UART serial connection. Formally speaking, a UART is a hardware device for asynchronous serial communication between two chips based on a unidirectional two wire bus. Implementation specifics for UART devices vary on a per-chip basis, but the protocol itself is based on Electronic Industries Association-232 (EIA-232), formerly known as Recommended Standard-232 (RS-232) [27]. UART devices perform first-in, first-out (FIFO) transmission, with speeds of up to 1 Mbit/s on the MSP432 family of microcontrollers.

Mechanical

The Underwriter's Laboratory (UL) has released specification UL 4600 on the evaluation of autonomous products [28]. This specification applies to any autonomous system capable of movement, as is the case for our chess robot. Specifically, it is aimed at safety concerns for autonomous systems related to the reliability of their dependent hardware and software, and the integrity of data contained within the system. UL 4600 makes a point of not prescribing specific requirements for autonomous systems; instead, it asks the systems' designers to answer a series of questions such as whether maintenance is required and how the system will interact with people. To this end, we have installed limit switches to prevent the robotic arm from extending beyond the sides of the gantry, and an emergency stop button that can completely disable the system. This helps protect users against potentially hazardous components. All major circuitry (e.g., the PCB, MSP432, Raspberry Pi, etc.) is also secured to the frame in locations that limit potential collisions and interference.

Section 106 of the Consumer Product Safety Improvement Act (CPSIA) [29] codifies the ASTM International Standard Consumer Safety Specification for Toy Safety (ASTM F963-07) [30]. This specification requires that any device sold as a children's toy must sufficiently label potential hazards that may not be commonly recognized by an average user of the system. While our intent is not to produce a children's product, the nature of chess as a household game necessitated our attention to this standard. As such, we have placed warning stickers at all major pinch points that we identified along the gantry's frame

Software

Embedded code written for the MSP432 microcontroller was written using the "BARR-C" coding standard written by Michael Barr [31]. Its purpose is to enforce good coding practices that reduce programming defects in embedded software. It describes every aspect of development from the naming of files and variables to use cases for various data types.

Intellectual Property Issues

Following a search of prior art, we believe the current iteration of our project likely would not be patentable. Patent number 4,398,720, *Robot Computer Chess Game*, describes a chess robot with a rotational arm that uses magnetic pieces [32]. However, this patent claims the robot goes a step further in that it possesses "emotional characteristics." It can engage in a physical game of chess against a human player, sense where pieces are on the board, return pieces to the board after

a game has ended, and respond with special actions/emotions when certain moves are played. The two main differences between this design and ours are the arm mechanism and the way pieces are detected. This patent describes a rotational arm, similar to our preliminary design, in place of our gantry system. The *Robot Computer Chess Game* also uses magnets on the bases of the chess pieces much like we do, but the sensing mechanism is distinct. The patent describes their sensing mechanism having two sets of magnets, one for the bottom of the piece and one that raises from below the board to close a circuit and enable detecting. In comparison, our design uses one set of magnets that close reed switches embedded in the board. While this patent resembled our project in terms of purpose, it is distinct in terms of mechanics. Patent number 6,446,966, *Chess Game and Method* [33], describes a methodology for chess gameplay, rather than specific physical system. This patent requires a chessboard with additional tiles, which are used by an orchestrator device that oversees and referees the game. Patent number 56,340, *Robot Arm With A Shearing Drive, And A Gantry Robot*, describes a more general robotic gantry system used for industrial shearing [34]. The design in this patent is realized as a gantry capable of moving along three axes. However, the grabbing mechanism employed is not an electromagnet as we use. Furthermore, while our gantry uses a stepper-driven rack and pinion for motion along the vertical axis, this patent uses a retractable grabber mechanism.

Detailed Technical Description of Project

The goal of this project was to produce an autonomous robot that is capable of competently playing chess against a human opponent. Due to the complex nature of our design, the detailed technical description of this project has been organized into the following sections:

- Software
    a. Communication
    b. Raspberry Pi Software
    c. Microcontroller Software Architecture
    d. UART and Raspberry Pi Modules
    e. Stepper Motor Module
    f. Gantry Module
- Mechanical
    a. Gantry System
    b. Sensor Board
- Electrical
    a. Overall Design
    b. Power Subsystem
    c. Motor Subsystem
    d. Sensor Network
    e. Switches and Buttons
    f. LEDs

A high-level block diagram of the system is shown in Figure 1.

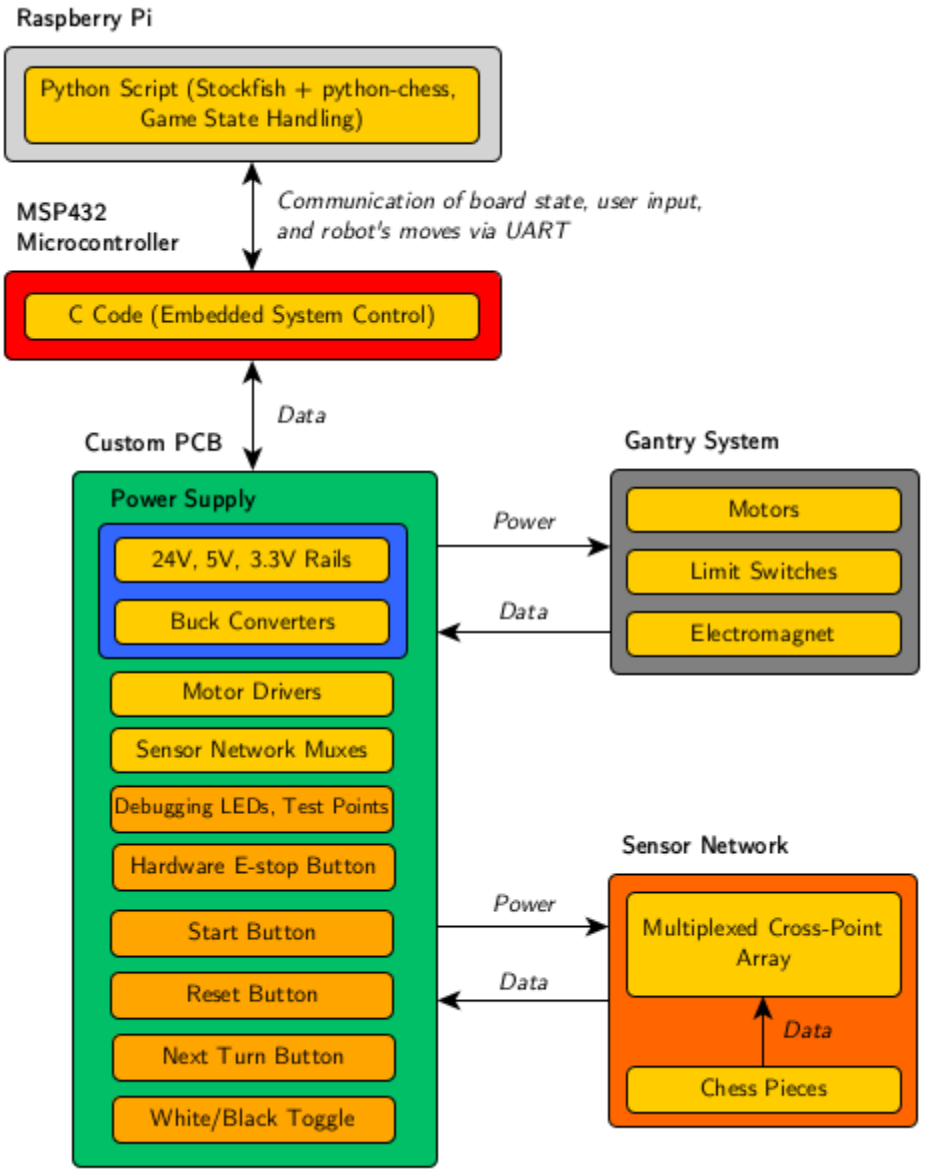

Figure 1: High-level system block diagram

Software

The software for the device is divided between two subsystems: the MSP432 microcontroller, where all embedded code lives, and the Raspberry Pi, where the chess engine lives. The MSP432 and Raspberry Pi communicate via UART to relay moves and advance the game. The MSP432 is responsible for reading moves from the sensor network, reading user input from the PCB, and performing motor functions. The Raspberry Pi is responsible for determining move legality, game state (ongoing, checkmate, stalemate), and generating the robot's moves via Stockfish. A high-level finite state machine of the software from the MSP432's perspective is shown in Figure 2.

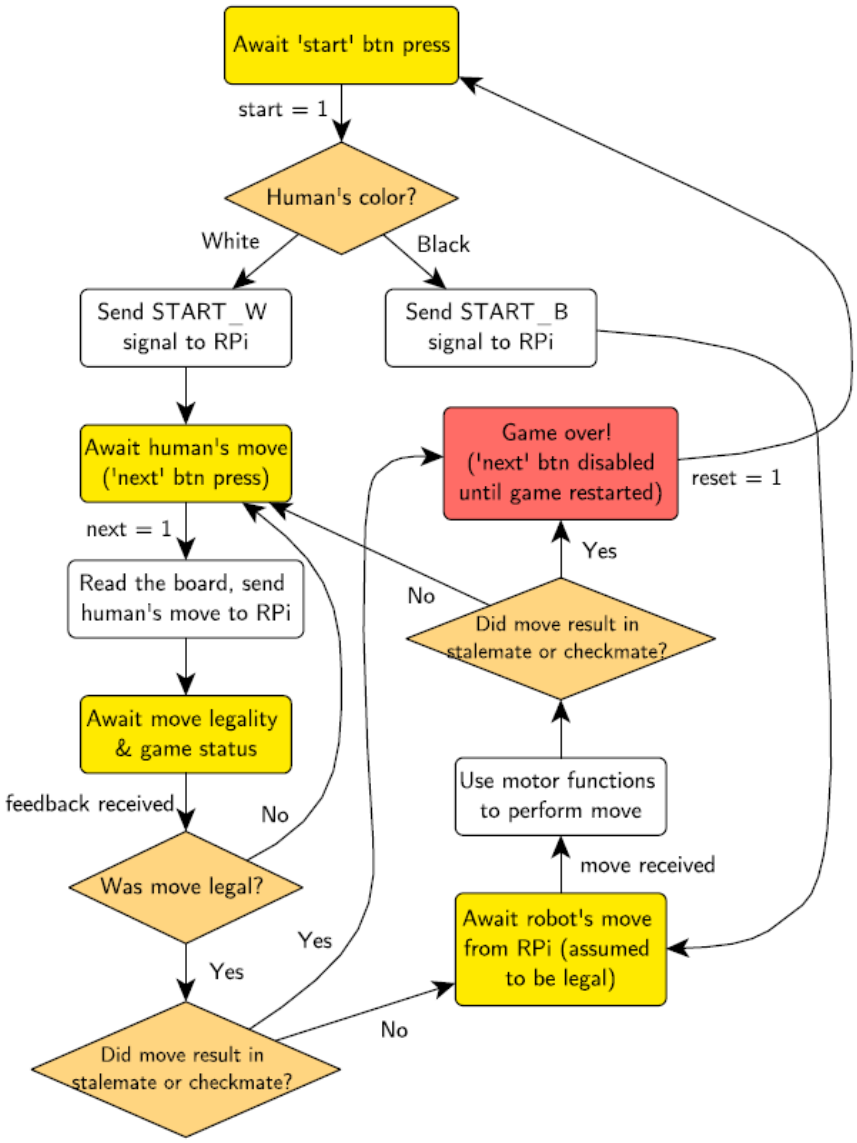

Figure 2: A high-level FSM of the software from the MSP432's perspective

Many of the 80 GPIO pins available on our MSP432 were mapped to specific components on our PCB. The specific mappings for pins we used are described in Table 1.

Table 1: All MSP432 pins in use for this project

Function	Module	MSP Port.Pin	In/Out
LED Red	GPIO	PC.4	In
LED Green	GPIO	PE.4	In
LED Blue	GPIO	PC.5	In
LED Out	GPIO	PC.6	Out
Next Turn	GPIO	PE.5	In
Stepper X Fault	GPIO	PD.3	In
Stepper X Step	GPIO	PC.7	Out
Stepper X Enable	GPIO	PB.2	Out
Stepper X Direction	GPIO	PB.3	Out
Stepper X Home	GPIO	PD.2	In
Stepper Y Fault	GPIO	PG.0	In
Stepper Y Step	GPIO	PD.7	Out
Stepper Y Enable	GPIO	PE.3	Out
Stepper Y Direction	GPIO	PE.2	Out
Stepper Y Home	GPIO	PF.3	In
Stepper Z Fault	GPIO	PB.4	In
Stepper Z Step	GPIO	PN.4	Out
Stepper Z Enable	GPIO	PN.5	Out
Stepper Z Direction	GPIO	PP.4	Out
Stepper Z Home	GPIO	PB.5	In
Stepper XY Microstep_0	GPIO	PA.6	Out
Stepper XY Microstep_1	GPIO	PM.4	Out
Stepper XY Microstep_2	GPIO	PM.5	Out
Stepper Z Microstep_0	GPIO	PK.7	Out
Stepper Z Microstep_1	GPIO	PK.6	Out
Stepper Z Microstep_2	GPIO	PH.1	Out
Limit Switch X	GPIO	PK.2	In
Limit Switch Y	GPIO	PK.1	In
Limit Switch Z	GPIO	PK.0	In
Capture Tile	GPIO	PP.1	In
Start Button	GPIO	PF.2	In
Reset Button	GPIO	PF.1	In
Home Button	GPIO	PM.3	In
Signal Light	Unused		Unused
Demux Select_0	GPIO	PD.1	Out
Demux Select_1	GPIO	PD.0	Out
Demux Select_2	GPIO	PN.2	Out
Mux Select_0	GPIO	PA.7	Out
Mux Select_1	GPIO	PP.5	Out
Mux Select_2	GPIO	PM.7	Out

Row Select_1	GPIO	PH.2	Out
Row Select_2	GPIO	PL.3	Out
Row Select_3	GPIO	PL.2	Out
Row Select_4	GPIO	PH.3	Out
Row Select_5	GPIO	PL.4	Out
Row Select_6	GPIO	PL.1	Out
Row Select_7	GPIO	PL.0	Out
Row Select_8	GPIO	PL.5	Out
Electromagnet_1	PWM0	PK.4	Out
Electromagnet_2	PWM0	PK.5	Out
UART3 TX (Transmit)	UART3	PA.5	Out
UART3 RX (Receive)	UART3	PA.4	In

Communication

Since the Raspberry Pi and MSP432 share responsibility in moderating and advancing the game, they had to have a mechanism for communication. As mentioned before, our system handles this communication via the UART serial communication protocol, which was chosen primarily for its simplicity. In a relatively simple application like this one, any serial communication protocol could have worked, and UART was the one with which the developers were most familiar.

A custom UART instruction set has been developed for this system. All instructions except the acknowledgement (ACK) instruction include the following, in order:

- 1 byte, the start byte (0x0A)
- 1 byte, the instruction ID and operand length byte
    - The upper 4 bits represent the instruction ID (one of: {0,...,5})
    - The lower 4 bits represent the operand length (one of: {0, 5, 6})
- 0, 5, or 6 bytes for the operand
- 2 bytes, a pair of check bytes calculated with the Fletcher-16 algorithm [35]

Hence, instructions can be 4, 9, or 10 bytes long. An example of an instruction and its structure are shown in Figure 3. The full instruction set is shown in Table 2.

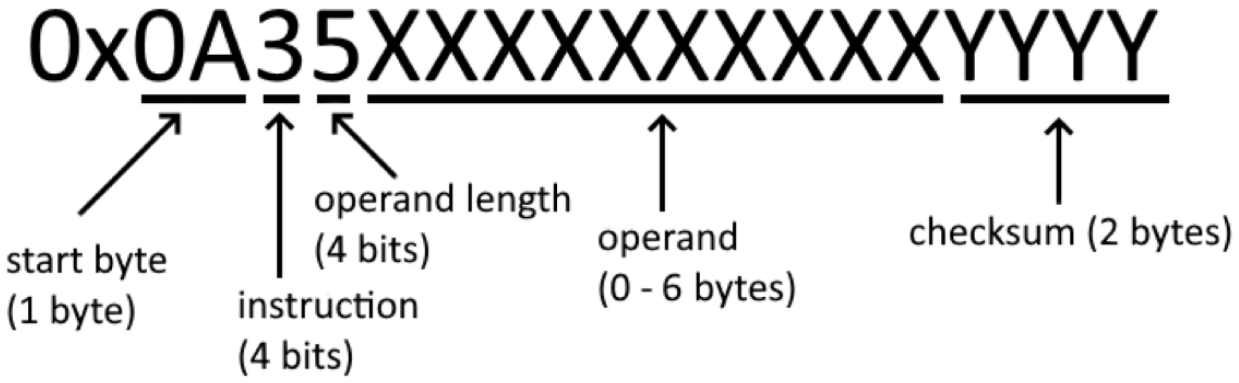

Figure 3: A visual example of one of the custom UART instructions, "Human Move"

Table 2: Custom UART instruction set; does not include 2 check bytes

Instruction	Description	Operand Description
0x0A00	Reset Game	N/A
0x0A10	Start Game (Human White)	N/A
0x0A20	Start Game (Human Black)	N/A
0x0A35XXXXXXXXXX	Human Move	X = Human's move in UCI notation (5 bytes)
0x0A46YYYYYYYYYYZZ	Robot Move + Game Status	Y = Robot's move in UCI notation (5 bytes) Z = Game status (1 byte)
0x0A50	Illegal Move	N/A
0x0F	ACK	N/A

To make the protocol more robust, every instruction except ACK has two check bytes appended to the end. The purpose of these check bytes is to detect transmission errors, and they are calculated by applying the Fletcher-16 algorithm [35] on the rest of the instruction. When a device receives a message, it will manually calculate the expected check bytes using the same Fletcher-16 algorithm the sender used. Then, two outcomes are possible:

1. If the manually calculated check bytes are **equal** to the check bytes sent (the last two bytes in the message), the receiver sends back an ACK.
2. If the manually calculated check bytes are **not equal** to the check bytes sent, the receiver does not send an ACK.

In the second case where an ACK is not sent, the sender will, after 5 seconds of not receiving an ACK, resend the previous message. The sender will continue doing this until an ACK is received.

The "Human Move" instruction is one of the core instructions for advancing the game. Of particular importance are the 5 operand bytes used for the human's move in UCI notation (see Appendix: Appendix A - UCI Notation). Moves in UCI notation can be 4 – 5 characters, so the protocol accounts for the maximum of 5 bytes. If the UCI move is only 4 characters, a '_' is substituted to maintain consistent operand lengths.

The "Robot Move" instruction is the other core instruction for advancing the game. It is similar to the "Human Move" instruction, but its fifth and sixth operand bytes contain additional functionality. The fifth byte contains a different character depending on the nature of the robot's move. This is to help the robot perform the move properly since different types of moves require different motor commands from the robot. Additionally, the sixth byte includes information about the game state (ongoing, checkmate, stalemate) after the human's move and the robot's move. The upper 4 bits are reserved for the game status after the human's move, and the lower 4 bits are reserved for the game status after the robot's move. The MSP432 uses this information to determine who won when the game has ended. The possible values for the fifth and sixth bytes in the "Robot Move" instruction and their meanings are summarized in Table 3 and Table 4.

Table 3: Possible fifth bytes for "Robot Move" instruction

Character	Meaning
_	Non-special move
C	Capture
c	Castle
Q	Promotion
q	Promotion capture
E	En passant

Table 4: Possible sixth bytes for "Robot Move" instruction

Number	Meaning
1	Ongoing
2	Checkmate
3	Stalemate

Raspberry Pi Software

As previously stated, the Raspberry Pi is responsible for determining move legality, game state (ongoing, checkmate, stalemate), and generating the robot's moves via Stockfish. It does this via a single Python script which runs automatically via a Linux cron job on every boot. The finite state machine for the software is shown in Figure 4.

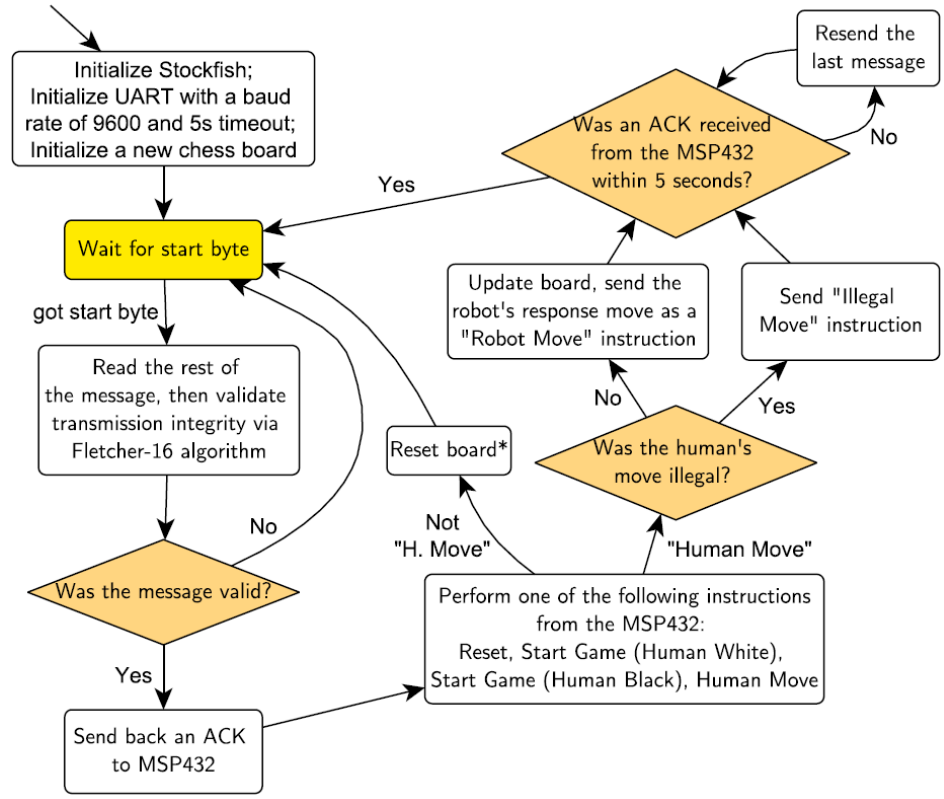

Figure 4: FSM of the Raspberry Pi's Python script

To simplify the FSM, only the flow following a “Human Move” instruction is shown in detail. The other 3 instructions (“Reset” and “Start Game” for either color) *do* reset the board, but “Start Game (Human Black)” behaves differently. For this instruction, since the robot is playing white, it must make the first move. So, the script will reset the board *and* send an opening move for the robot to make. Then it will go through the same process of waiting for an ACK from the MSP432 and resending its message every 5 seconds if it does not receive one. “Reset” and “Start Game (Human White)” *only* reset the board. They are functionally the same but exist separately to keep code more meaningful.

Microcontroller Software Architecture

The software on the MSP432 operates via a command queue. The queue holds pointers to command structs, each of which contains four function pointers (see Figure 5). Each module extends the base command struct by adding any data necessary for the command to execute. This architecture allows for loose coupling between modules and flexible development where the programmer can add new modules by creating a new command type with any data it needs. All other logic is handled by the queue.

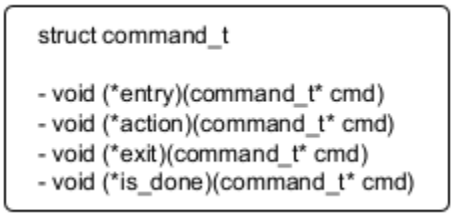

Figure 5: The command_t struct and its fields

The main function contains a *while()* loop that pops commands off the queue. When a command is popped from the queue, it will run the entry function exactly once. It will then check to see if the command is done via the *is_done()* function. If the command has not completed, it will call the action function in a loop. Once the done condition is satisfied, it will run that command exit function exactly once, then pop the next command from the queue. In the event some sort of system fault occurs while a command is executing, for example the emergency stop button is pressed, there is a check for a “system fault” or “system reset” flag before an action is run. If the system encountered a fault, it intentionally triggers a hard fault on the MSP432 that can only be cleared by power cycling the device. If a reset occurs, it breaks the action loop and runs exit, to stop the current command. The queue is then cleared, and a reset command gets added that clears the reset flag and reinitializes the system so functionality can continue as before. The flow of this command queue architecture is shown in Figure 6.

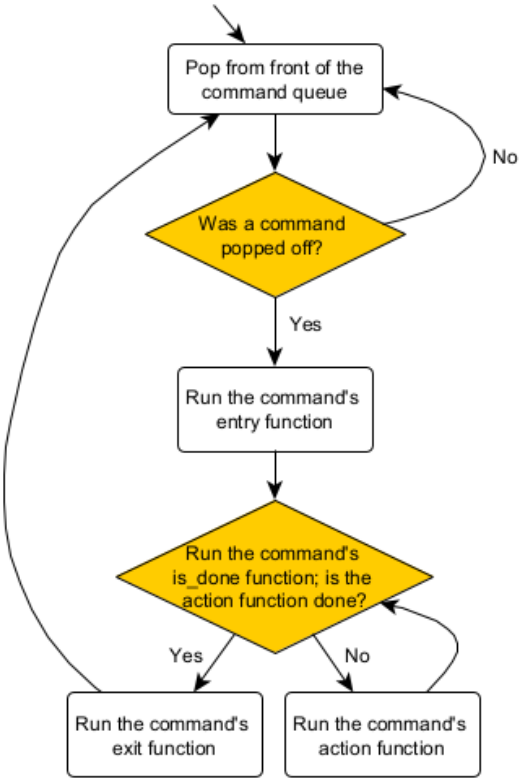

Figure 6: The flow of the command queue as it runs in an infinite loop in the main method

UART and Raspberry Pi Modules

The MSP432E401Y has 8 UART channels (UART0 – UART7). For this project, up to two channels are used at a time. In the "standard" mode, only the UART3 channel is used to establish communication between the MSP432 and the Raspberry Pi. In another mode called "three-party" mode, the human may manually supply their moves, in UCI notation (see Appendix: Appendix A - UCI Notation), through a Python emulator running on a laptop with a COM port. The UART0 channel is then used to send the move from the laptop to the MSP432, which sends the move to the Raspberry Pi. This was mainly used for debugging purposes, and not intended as the final functionality.

The UART module works by maintaining a queue in software for received bytes and bytes to be transferred (RX and TX queues) for each channel. When bytes are transferred out, they are put into a channel's TX queue and then copied into the MSP432's data register to be transferred out. Bytes are received via a periodic interrupt which copies them from the data register on the MSP432 to the software RX queue. The bytes can then be read by popping them off the RX queue. The "Raspberry Pi" module wraps many of the functions defined in the UART module to easily interact with the Raspberry Pi in particular. The most essential functions present in both modules are shown in Figure 7.

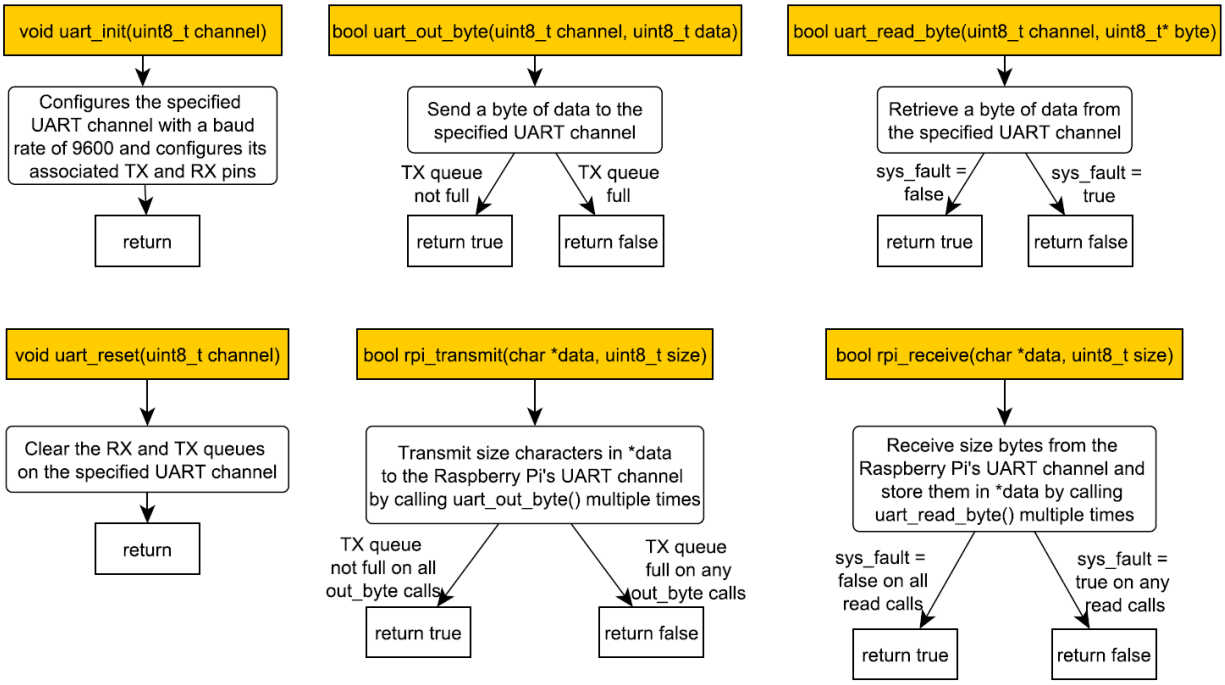

Figure 7: Essential UART and Raspberry Pi module functions

Stepper Motor Module

The X, Y, and Z stepper motors are all controlled via the same logic. Once a stepper command is run by the command queue, it takes the distance it needs to move each motor in millimeters and converts that to the number of step transitions it will take to move that distance, with two step transitions performing one step on the motor. It will then calculate the points in the motion where it needs to accelerate and decelerate to create a trapezoidal motion profile. The exact type of profile is determined by a preselected maximum velocity and maximum acceleration. Using these, the point where the motor needs to stop accelerating and then start decelerating are calculated ahead of time using Equations 1 and 2.

$$x_{\text{accel}} = \frac{1}{2} \cdot \frac{v_{\text{max}}^2}{a_{\text{max}}} \tag{1}$$

$$x_{\text{decel}} = \text{Distance Traveled} - x_{\text{accel}} \tag{2}$$

Once these calculations are complete, a clock configured to step the motors at a predefined initial velocity is started for each motor. This clock runs as a periodic count-down timer and triggers an interrupt upon reaching zero. When that interrupt fires, the interrupt handler toggles the STEP pin on the appropriate motor driver and decrements the number of step transitions remaining for that motor. The period of the clock is adjusted each time the interrupt fires to modify the velocity of the motors. Once a given motor reaches zero step transitions remaining, it is disabled to prevent unnecessary current flow. When all three motors reach zero step transitions remaining, the system will have arrived at its destination, and the stepper command exits.

As mentioned before, two step transitions result in one step being performed on the motor. More generally, the motors support microstepping whereby two step transitions cause one microstep. We can then perform simple arithmetic to translate the desired distance in millimeters to a number of step transitions. Suppose the motors perform M microsteps per step. Each motor performs 200 full steps per revolution, so $200M$ microsteps per revolution. For the X and Y motors, the belt pitch is 2 millimeters, and the rotor on each stepper has 20 teeth, so the belt moves 40 millimeters per revolution. There are two transitions per microstep, so the number of transitions per millimeter traveled is $10M$ transitions per millimeter, as shown in Equation 3.

$$\left(2\frac{trans}{microstep}\right) * \left(200M\frac{microstep}{rev}\right) / \left(40\frac{mm}{rev}\right) = 10M\frac{trans}{mm} \quad (3)$$

For the Z motor, the belt pitch is 2.5 millimeters, and the rotor on each stepper has 20 teeth, so the belt moves 50 millimeters per revolution. There are still two transitions per microstep, so the number of transitions per millimeter traveled is $8M$ transitions per millimeter, as shown in Equation 4.

$$\left(2\frac{trans}{microstep}\right) * \left(200M\frac{microstep}{rev}\right) / \left(50\frac{mm}{rev}\right) = 8M\frac{trans}{mm} \quad (4)$$

For our use case, we set the microstepping level to $\boldsymbol{M = 8}$, as this allowed us to perform an integer number of microsteps to reach the center of each tile. Hence, the X and Y motors perform 80 transitions per millimeter, and the Z motor performs 64 transitions per millimeter.

Gantry Module

The gantry module performs the main chess logic and is the interface between all of the other modules, with the exception of some utility functions. It is responsible for choreographing the robot's movement after receiving a move from either Stockfish or the sensor network. The gantry module also handles fault detection. This module has two primary command types: one where the robot is making a move, dubbed "robot move command," and one where a human is making a move, dubbed "human move command." Whenever the system is reset (e.g., to start a new game), gantry will clear the command queue and reset all its submodules. It then checks whether the human is playing black or white via a toggle switch on the PCB, and lets Stockfish know that a new game has begun and who is playing each side. If the human is playing as white, it will put a human move command on the queue; if the human is playing as black, it puts a robot move command on the queue. Human move commands place robot move commands on the queue as part of their *exit()* function, and vice versa. This is how the game continues between robot move and human move. If either command finds the game has ended, it blinks an indicator LED on the PCB to designate who won and empties the queue until the reset button is pressed.

Mechanical Design

The physical design of the robot consists of two principal components: a cantilevered gantry system, and a chessboard with embedded reed switches. The gantry system is responsible for moving the robot's pieces, while the chessboard is responsible for determining the position of each piece.

Gantry System

The frame of the gantry consists entirely of 20x20mm and 20x40mm aluminum V-Slot rail. A cantilevered arm runs along a 20x40mm rail forming the X-axis while a "cart" runs along a 20x40mm beam to form the Y-Axis. The X-axis is intentionally longer than the size of the board to allow the robot's arm to rest off the board and away from any user's hands. Both the X and Y axes are powered by a belt and pulley system driven by two NEMA 17 stepper motors, while the Z-axis consists of another NEMA 17 stepper motor driving a rack and pinon mechanism that raises and lowers an electromagnet. The electromagnet "grabs" pieces from the board by attracting screws in the top of their heads when engaged. Likewise, cutting power to the magnet allows the pieces to return to the chessboard. To prevent overruns, each axis has a limit switch at the far end of its range of movement. This allows the software to find a fixed point of reference by driving each axis in one direction until the limit switch is triggered. Since the switches are fixed, that point is consistent between power cycles. The PCB, MSP432, and Raspberry Pi attach to the backside of the robot on custom mounts. The PCB and MSP432 are positioned such that when the X-axis moves, the cables for the gantry can remain short without getting caught on any obstructions. The Raspberry Pi is positioned near the PCB and MSP432 to limit the length of the power and data lines that connect the devices. The robot's CAD model is shown in Figure 8, while the final product is shown in Figure 9.

Figure 8: Final Robot CAD

Figure 9: Final Robot Assembly

The lifting mechanism consists of a 3D printed mount attached to a cart running along the Y-axis as shown in Figure 10. The mount supports the NEMA 17 stepper motor, a limit switch, a channel for the rack to travel on, and various cable tie mounts. Attached to the mount is another platform that supports a 70.4 cm long cable carrier (not shown in Figure 8) that connects the X-axis cart and the Z-axis mount. This protects all of the wires needed to drive the Z-axis motor, read the limit switch, and utilize the electromagnet, while neatly keeping them away from the belts.

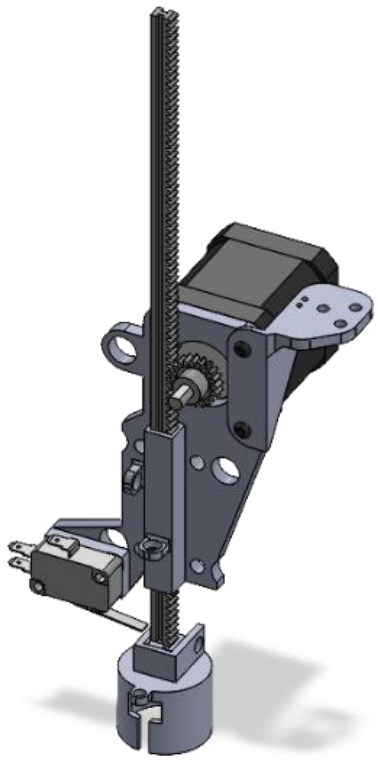

Figure 10: Piece Lifting Mechanism

The sensor board is supported by six 3D printed supports. The two closest to the back of the robot are bolted to the sensor board, the middle two are attached to the robot's frame but are not bolted to the board. The final two rest at the far end of the board and serve to prevent buckling in the event someone rests their weight against the board. The main structure consists of one 1/8" layer of polycarbonate upon which the reed switches rest, glued to another 1/4" layer of polycarbonate with slots to correctly orient the switches. Beneath the polycarbonate are four large PCBs which are bolted onto the bottom layer. These boards serve mostly to provide a strong connection to the reed switches and a neat method of handling the crosspoint array formed by the rows and columns. On top of everything is a 17" by 17" chess mat that is held in place by four corner bolts. Holes were made in the mat to support the heads of the bolts holding the PCBs so the mat lays flat against the polycarbonate. The final board is shown in Figure 11.

Figure 11: Assembled Sensor Board

Electrical Design

Overall Design

The electronic design was broken into a series of subsystems that handle different functionalities. The power supply subsystem steps the input voltage down to the operating voltages of our Raspberry Pi and MSP432. The MSP432 subsystem is a placeholder for the GPIO pinout on the actual MSP432 microcontroller. The Raspberry Pi subsystem connects the Raspberry Pi UART lines for serial communication. The button, switch, and limit switch subsystems are used for hardware peripherals, and each feature transient voltage suppression diodes on their input/output lines to prevent electrostatic discharge. The light emitting diode (LED) and signal light subsystems are used for lights on the system. The electromagnet and stepper subsystems are used to drive the electromagnet and stepper devices, respectively. The sensor network subsystem interfaces with the sensor network in our chessboard. The heatsink subsystem is used for thermal dissipation, whereby two large heatsinks on the PCB are connected by thermal vias to a large ground plan spanning the bottom of the board. Several "future proofing" subsystems were included that mirror the limit switch subsystem to leave free terminal connections on the PCB in the event that we decided to add a component (e.g., the emergency stop which was added later). Figure 12 shows the top level schematic for this design while Figure 13 shows the layout of the components.

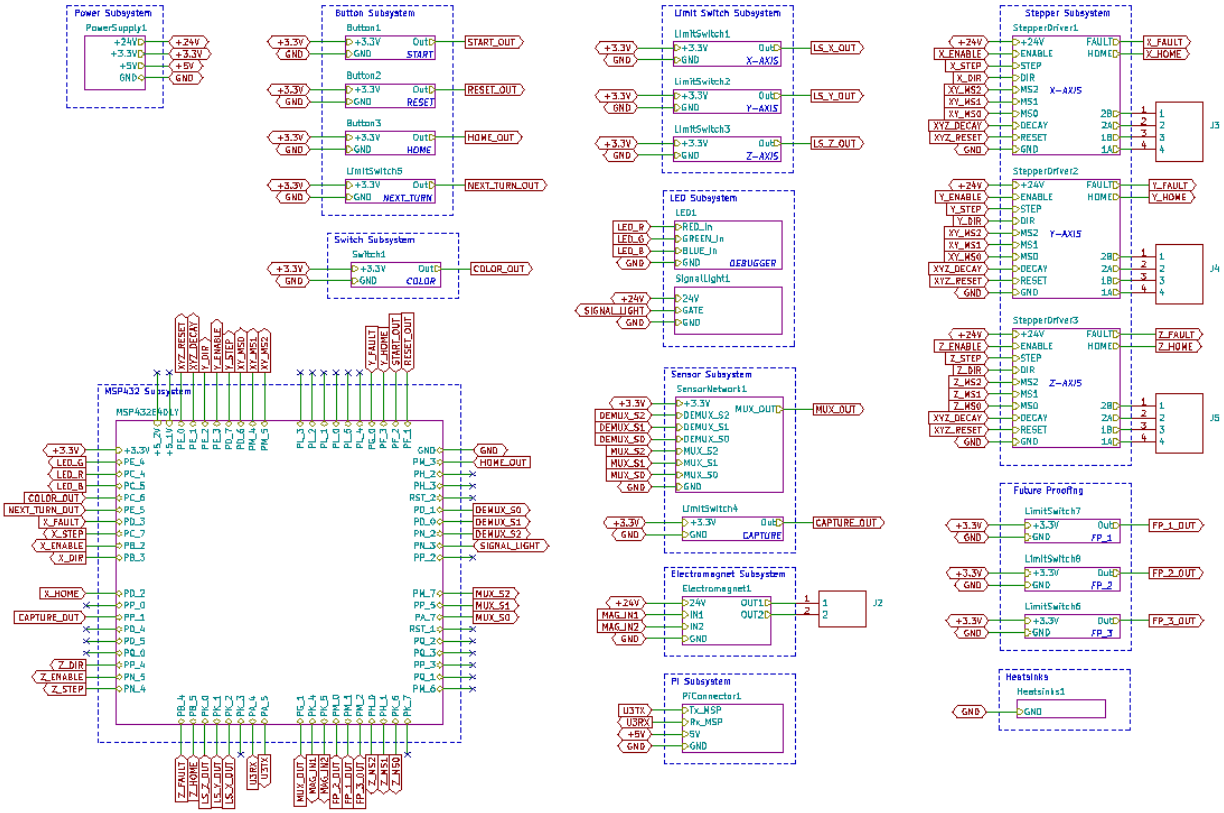

Figure 12: Top Level Schematic

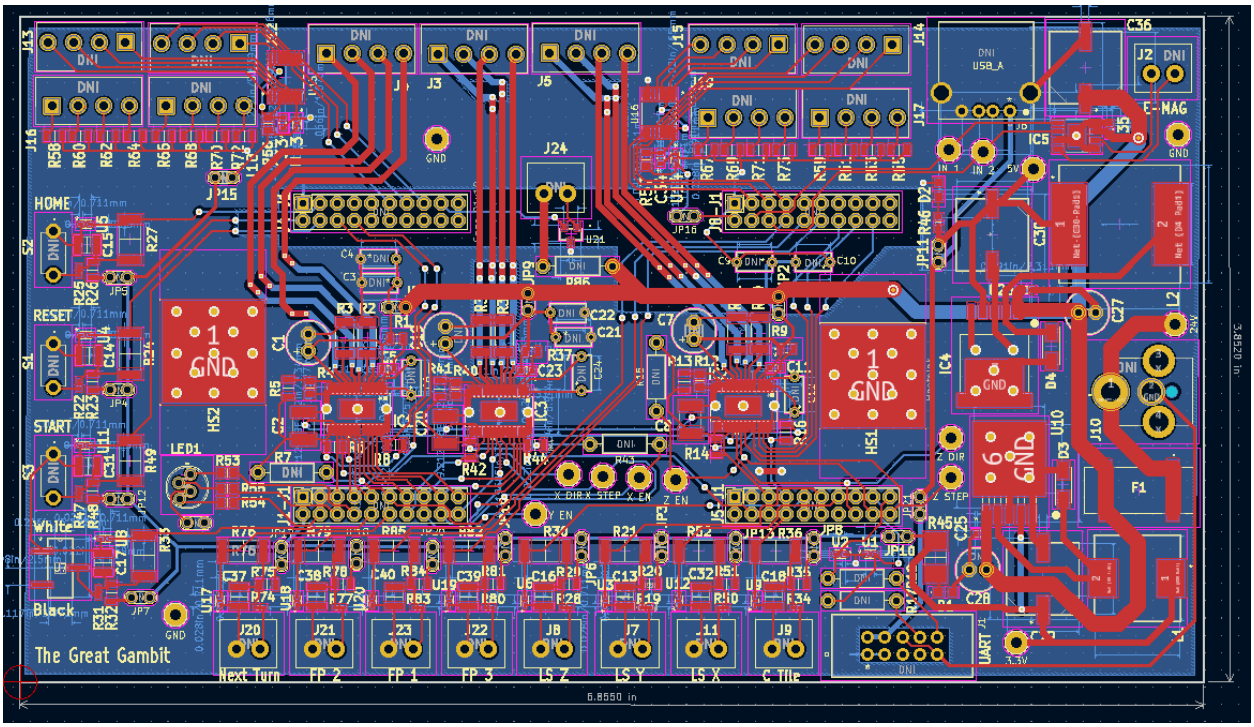

Figure 13: PCB Layout

Power Subsystem

The power for the entire system is supplied by a 24V, 160 W power supply [36] that connects to our PCB through a barrel jack. The barrel jack selected is rated for 24V at 8A to accommodate this load [37]. That supply is routed through a 10A fuse, which serves as a line of defense against short circuits and to disconnect the 24V input from the rest of the system when powering the MSP432 from a computer. The 24V line is then stepped down to 3.3V and a 5V rails using two fixed voltage LM2576 switching regulators from Texas Instruments [38]. These were chosen for their high efficiency and ease of use. Schottky diodes were selected based on the manufacturer's recommendations. Inductors for both circuits were selected according to the expected maximum current draw on both lines: ~400 mA on the 3.3V line, and 2.8A on the 5V line. Both circuits used the same 0.06 Ω low Equivalent Series Resistance (ESR) capacitors to reduce the ripple voltage as determined by Equation 5. This gives us an expected ripple voltage of 0.7 mV on the 3.3V line and 50mV on the 5V line.

$$V_{\text{ripple}} = 0.3 \cdot I_{\text{max}} \cdot \text{ESR}_{\text{Cap}} \tag{5}$$

A 0.1uF and a 100uF capacitor were placed in parallel at the input of the regulators to stabilize the incoming voltage. The 100uF capacitor was from the datasheet's recommendations while the 0.1uF was merely used as a bypass capacitor placed near the chip. Finally, to give some indication that the supplies are live, two LEDs with accompanying resistors were put at the output of the 3.3V and 5V lines. The power subsystem schematic can be found in Figure 14.

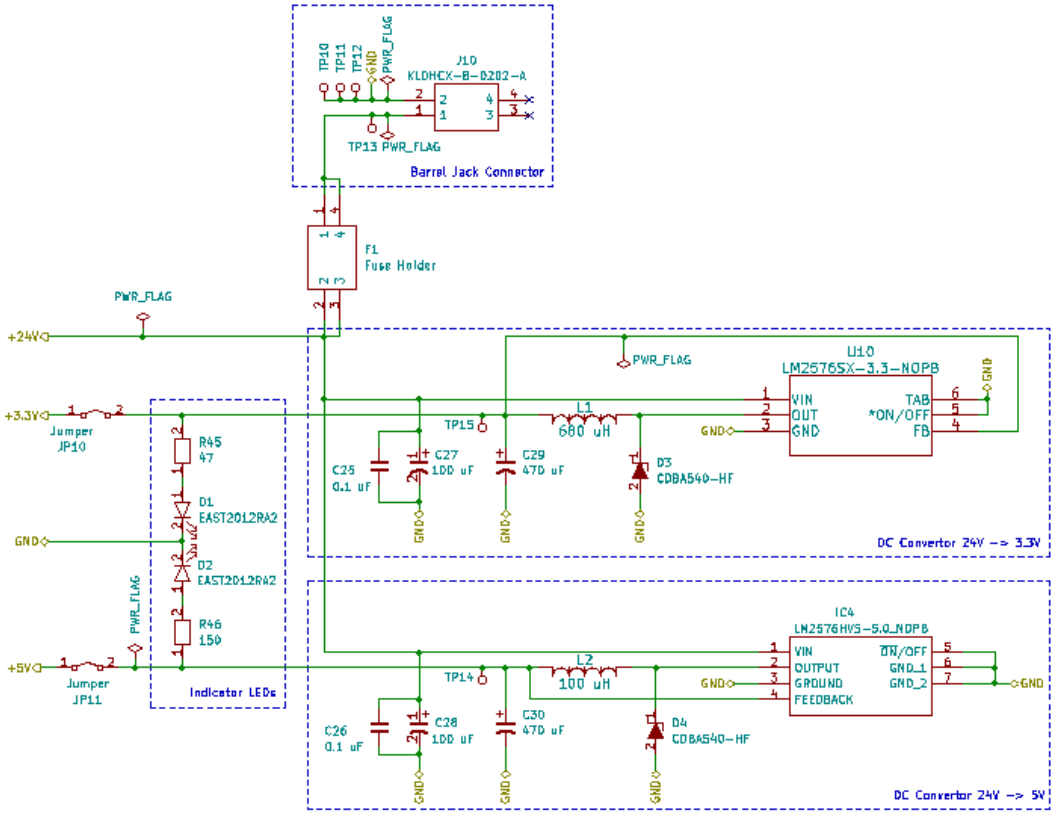

Figure 14: Power Subsystem Schematic

Motor Subsystem

Each of the X, Y, and Z stepper motors share a subsystem architecture. The driver used is a DRV8824 from Texas Instruments [X]. Due to the limited stock of driver chips available at this time of this project, use of this chip was required of us. Each driver contains two H-bridges, which switch the polarity applied to the four output lines that run to corresponding motor. Three microstep select lines set an indexer inside the driver chip that allows the motors to perform full steps, 1/2 steps, 1/4 steps, 1/8 steps, 1/16 steps, or 1/32 steps. A fault output indicates if the motor encounters an error due to thermal regulation, undervoltage, or overcurrent. A home output indicates when the motor thinks it has reached the position it was in when enabled, but our application did not end up using this functionality. A charge pump capacitor allows for quick start times after the chip has been disabled for an extended period. The driver contains an internal regulator that steps its 24V input down to a 3.3V line. This 3.3V line is used for the stepper subsystem alone, as it the rest of the system draws too much current to route all devices through these drivers. The enable, reset, and sleep lines are all active-low. We use the reset line to initialize the motor, and the enable line to turn the motor "on" or "off." A pullup resistor on the active-low enable ensures that the motor starts up in an "off" state until the system is ready to move. A thermal pad on the back of the chip connects to the ground on the bottom of the board through a series of thermal vias. The motor being driven by a given chip is turned by toggling the step input the driver, with the direction input determining if the motor will turn clockwise or counterclockwise. The stepper motor subsystem schematic can be found in Figure 15.

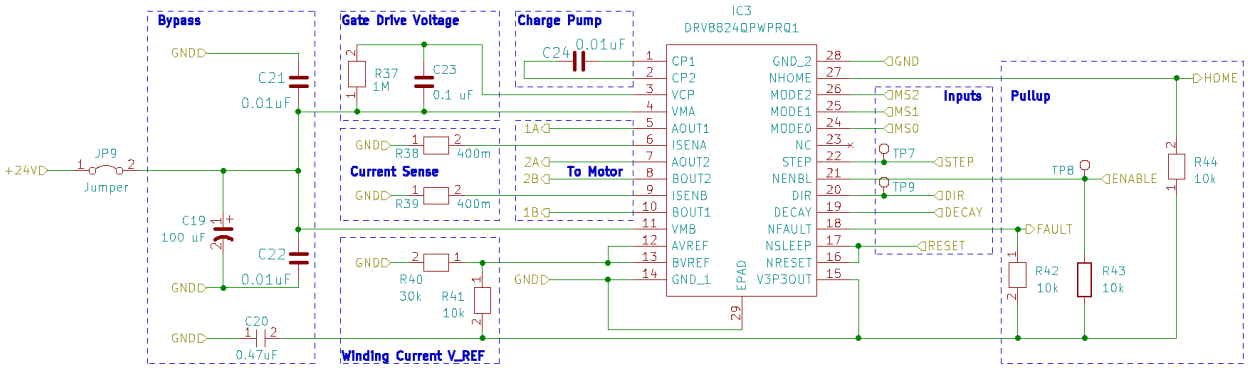

Figure 15: Stepper Motor Driver Subsystem Schematic

Sensor Network

The sensor network schematic in Figure 16 is the connection to the sensor that appears on the main PCB. The sensor board schematic in Figure 17 is the implementation of the network that is used on a series of secondary PCBs below the physical chessboard. The sensor board is arranged as a cross point array of the ranks and files on the chessboard, resulting in 16 output lines for the sensor network subsystem. Each of these lines are the output of an analog multiplexer/demultiplexer that reduces the number of GPIO pins needed down to seven. There are also 16 input lines for the rows and columns that connect to ground via pulldown resistors. Unfortunately, this subsystem caused some issues in practice. First, the power (VCC) and data (COM) lines on the row selection multiplexer were shorted. This was fine on the column selection

demultiplexer, where 3.3V was always applied to the data line, but not for the demultiplexer that was meant to output its data line. This was resolved by removing the multiplexer from our final PCB and introducing eight additional GPIO lines directly to the row inputs. The second issue was that pulldown resistors were fine on the row lines, but not the column lines. In fact, they caused all rows to constantly read as ground. This was an easy fix, as we simply unplugged the column pulldown terminal connectors.

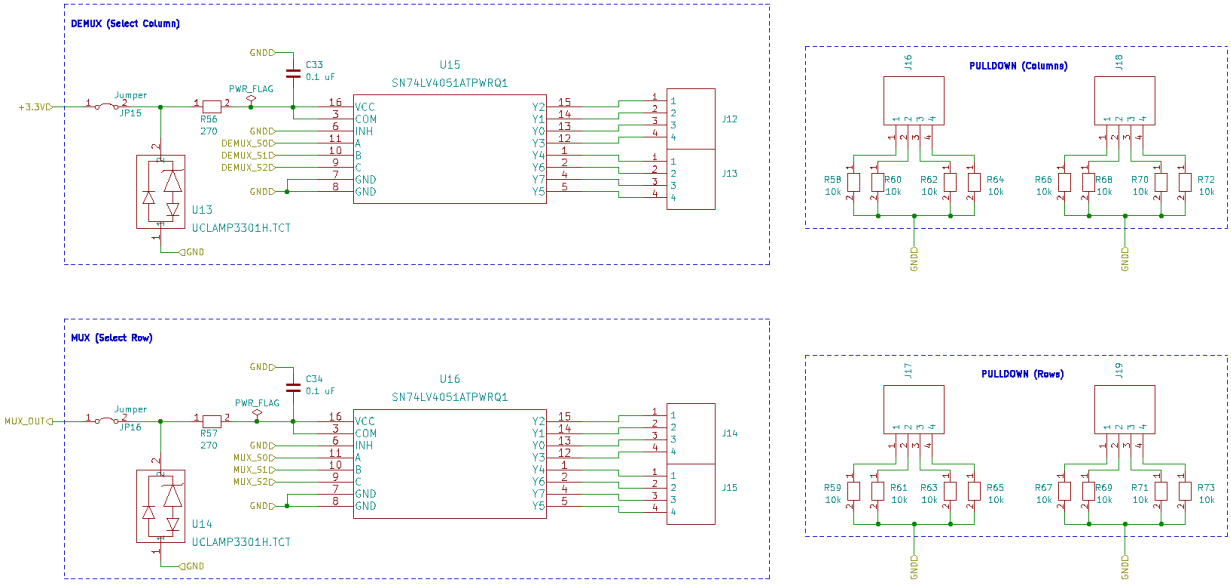

Figure 16: Sensor Network Subsystem Schematic

The sensor board layout is shown in Figure 17. This schematic was implemented across four secondary PCB's (each containing two columns of sensors) that were jumped together below the chessboard. There are 64 reed switches, each connected to one row and one column, and lined up below a tile on the chessboard. This portion of the sensor network also featured an issue, in that the diodes were initially overlooked. This was an easy correction after-the-fact, as the reed switches selected were through-hole components. Realistically, testing should have been conducted sooner to ensure that reed switches were the proper sensors for our device. We discovered that the switches can be incredibly inconsistent with their detection consistency; in particular, certain orientations of magnet did not trigger some switches. We initially chose not to use Hall effect sensors to avoid additional cost for the components and corresponding PCBs. Were we to do this project again, we would use Hall effect sensors in place of the reed switches, due to their increased consistency.

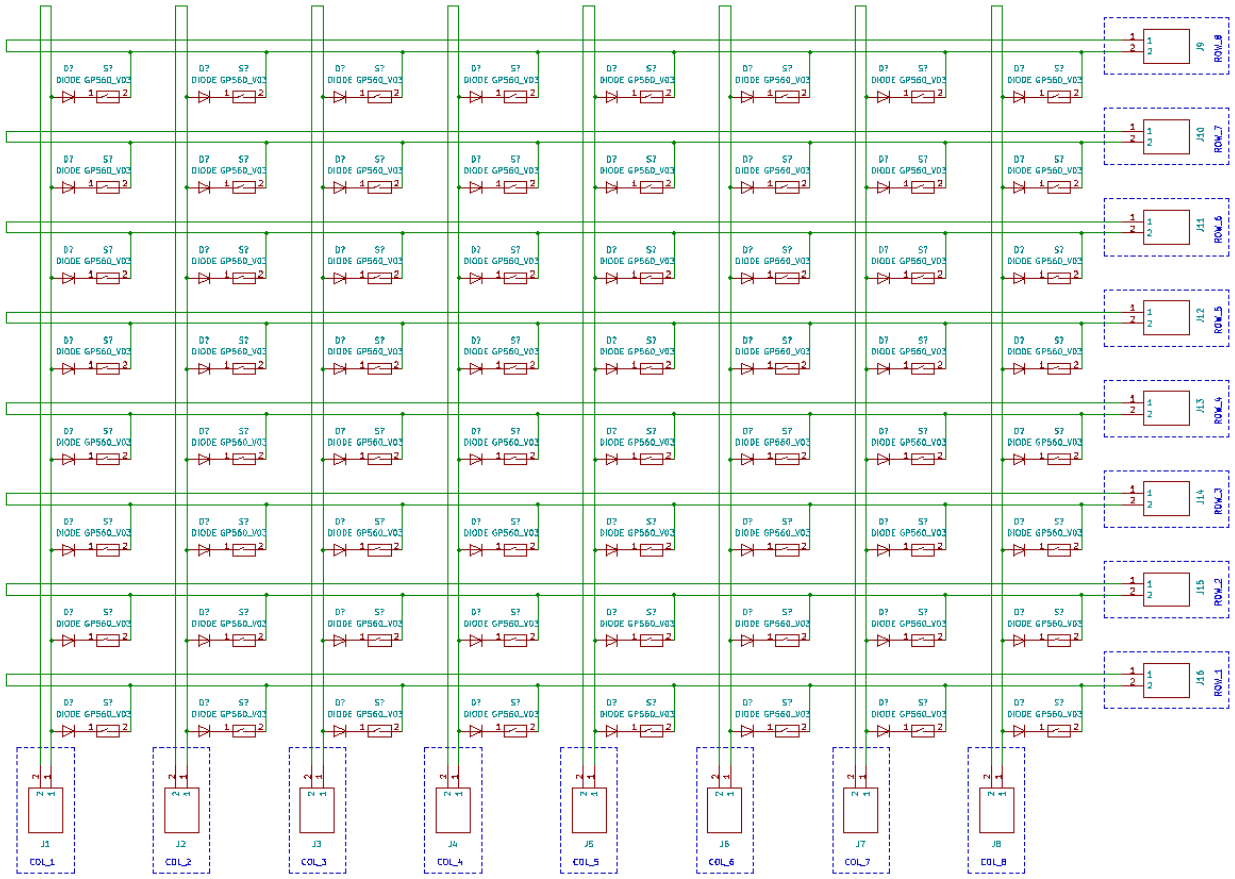

Figure 17: Sensor Chess Board Array Schematic

Switches and Buttons

All switches, button, limit switches, toggles, future proofing, etc. followed the same base pattern. A transient voltage suppression diode prevents electrostatic discharge from leaking into the system's main lines. This is important on inputs that connect directly to GPIO and may be accidentally shorted by human hands. A simple lowpass circuit acts as a preliminary debounce for the switches, which generally had bounce times of 50 ms. A pullup resistor drives the line high until the switch is activated, when it then becomes low. The only difference in the subsystem for switches, buttons, etc. was the device connected as the "switch." The switch subsystem schematic can be found in Figure 18.

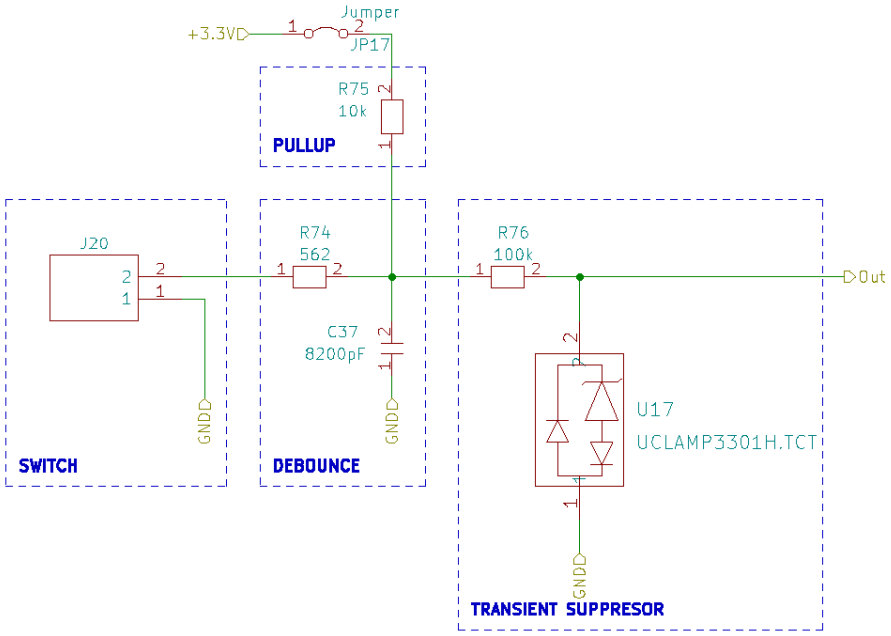

Figure 18: Switch Subsystem Schematic

LEDs

There are two LED types included in our design. One is a red/green/blue LED used for debugging and game state information. The other is a single, large, red status LED to indicate the robot is in operation. The debugging LED had 3 pins, one for red, blue, and green, each driven by a GPIO pin on the MSP through a 270Ω resistor. Its cathode is grounded through a jumper as demonstrated in Figure 19. The status light is controlled through an N-Channel MOSFET. The gate is connected to a GPIO pin on the microcontroller, while the LED is connected to the source pin and the drain is grounded. A 1.2kΩ resistor makes sure the current from the 24V supply stayed within the 20mA limits of the LED. The status LED is not soldered directly onto the PCB like the debugging LED. Instead, it connected through a 2-pin terminal block as shown in Figure 20. This status LED configuration did not end up working as expected and is detailed more in **Test Plan**.

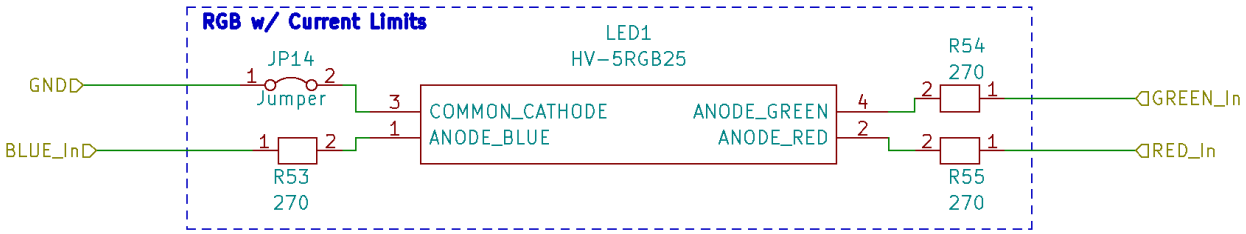

Figure 19: Debugging LED Schematic

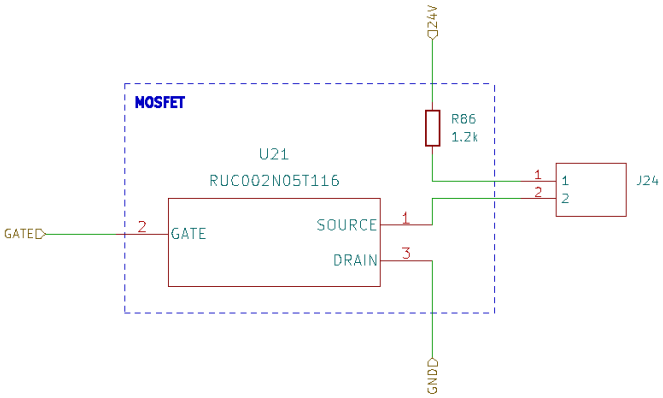

Figure 20: Status LED schematic

Project Time Line

Our initial schedule sought to have a working version of the project by the start of Thanksgiving break, which turned out to be a very optimistic assessment (see Figure 21). The PCB design took far longer, and was far more involved, than we had initially expected. As a result, we missed the first board send out deadline and thus had to use send outs two and three. This significantly compressed our timeline, as we were unable to fully assemble and test our first board before the final version was sent out for manufacture. This meant we could not test our circuitry until early November as opposed to early October that we had hoped. Additionally, the timing of the last board send out meant that, despite sending it off to be assembled as fast as possible, we did not get the assembled board until after Thanksgiving break. The combination of these two delays resulted in us only being able to do a full system test the week before the final deadline. Thus, our fixes were with limited resources, and we did not have time to remove all the bugs and corner cases.

The mechanical design of the robot, the design of the PCB, and much of the software was done largely in parallel. How many motors, sensors, and the grabbing mechanism were determined very early, so the circuity and code to control them could be created at the same time as their exact placement and configuration were determined. What could not be done in parallel was the sensor board and the final system integration. The sensor board was done after the robot assembly and PCB design as that was what determined both its size and what type of sensors it would contain. The final system integration required all the physical and electronic components to be completed, as without them we could not create movement without motors or detect pieces without sensors. The software to control and interpret all of our hardware was written well beforehand but could not be thoroughly debugged until everything was complete.

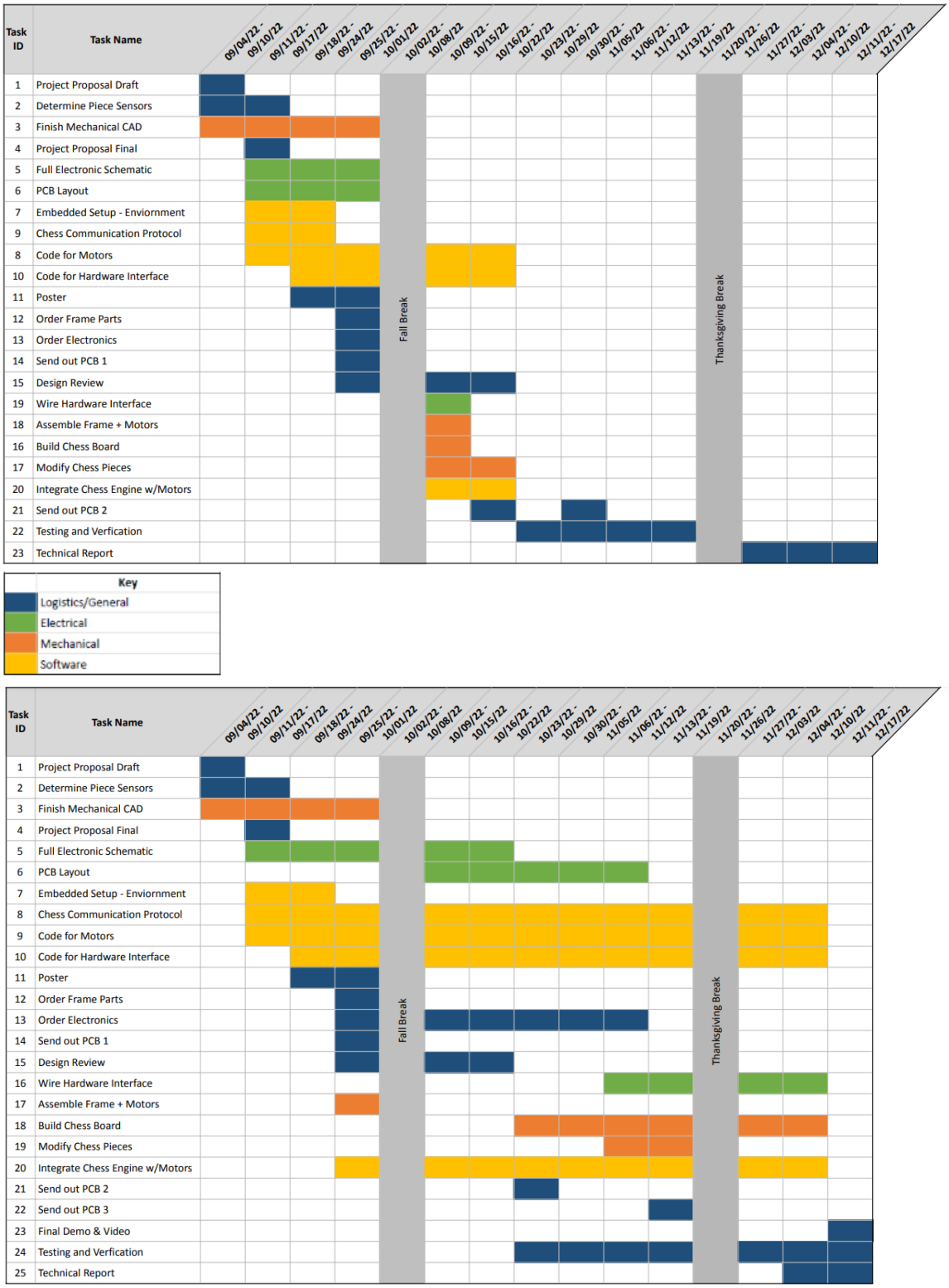

Figure 21: Original (Top) and Final (Bottom) Gantt Charts

Test Plan

Mechanical

Our mechanical testing mainly served to confirm the physical frame was both the correct size and as ridged as possible. Size was easy to determine, as if the chessboard fitted within the confines of the gantry and the electromagnet could both reach a pawn at its maximum extension and be higher than two kings when fully retracted, we were of an acceptable size. Structural rigidity was tested by applying force along each major axis. If the axis moved, then extra brackets were added to help hold it in place. The was repeated until twisting any of the beams resulted in no movement with one exception. The Y-axis beam drooped slightly under its own weight as its connection to the X-axis consists of 4 roller bearings. These were tightened as much as possible, and the Y-axis's beam moved back to reduce the deflection, but we were not able to completely remove it. To compensate, the software adjusted the vertical height it moved to grab each piece according to its position on the board. We also found that when the motors started moving the robot would slide around on the table. To stop this, we added four rubber feet which provided sufficient friction to hold the frame in place.

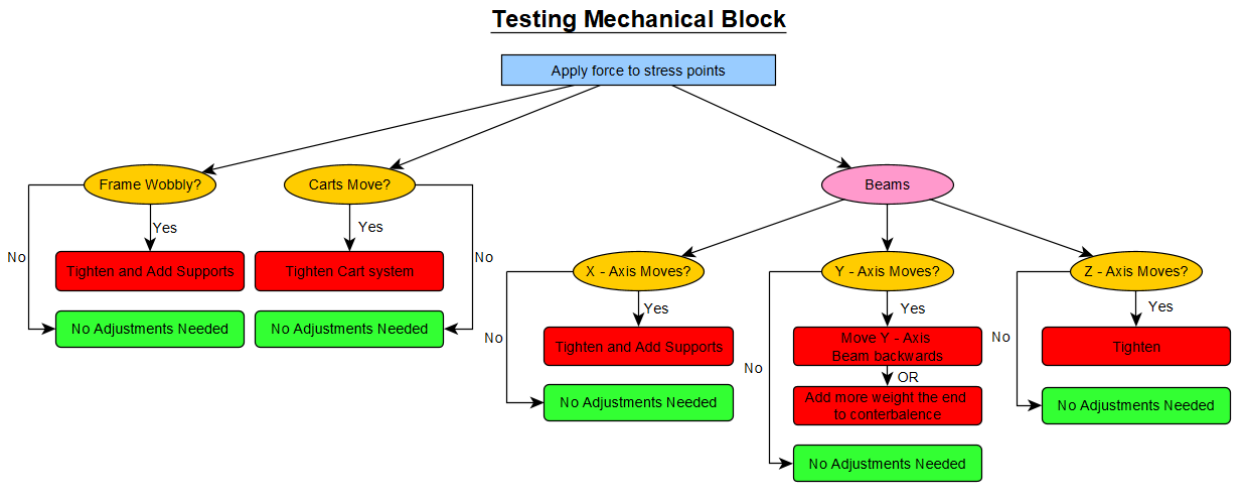

Figure 22: Mechanical System Test Plan

PCB Testing

Once the PCB was assembled, the first step in our testing regimen (see Figure 25) was to check that all the components were connected to the correct power net, and not to any others. This was done by using a digital multimeter in continuity mode to check between each pin and the power test points (GND, 24V, 3.3V, and 5V). Once it was determined that there were no obvious shorts, we proceeded to test the power supply. This proceeded as follows: first the external power supply was connected to the barrel jack while the 10A fuse was removed and the voltage from ground to the fuse holder was measured. This confirmed that the voltage level was correct and with the correct polarity. Next, the shunts connecting the 3.3V and 5V rails to the rest of the board were removed and the 10A fuse replaced. The 3.3V and 5V rails were then measured to determine if they were correct, first with a multimeter and then with an oscilloscope, the results of which are

in Figure 23 and Figure 24. Both showed themselves to be within acceptable limits, although the ripple voltage was a little higher than expected. However, the 2 LEDs that should indicate the supplies are live were not lit up when the voltages were present. Examining them under the microscope showed that they had been installed backwards, and so were reattached in the opposite direction. In doing so, the 3.3V light was damaged and thus showed very dimly, but the 5V LED worked as expected.

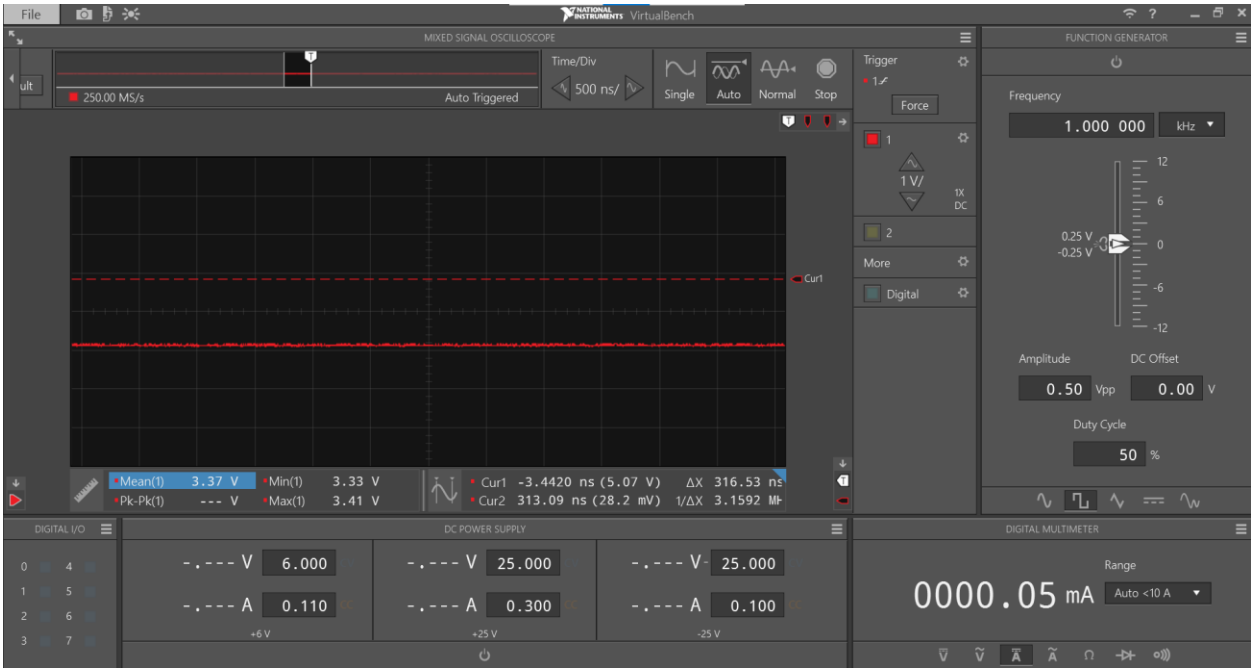

Figure 23: 3.3V Rail Measurement

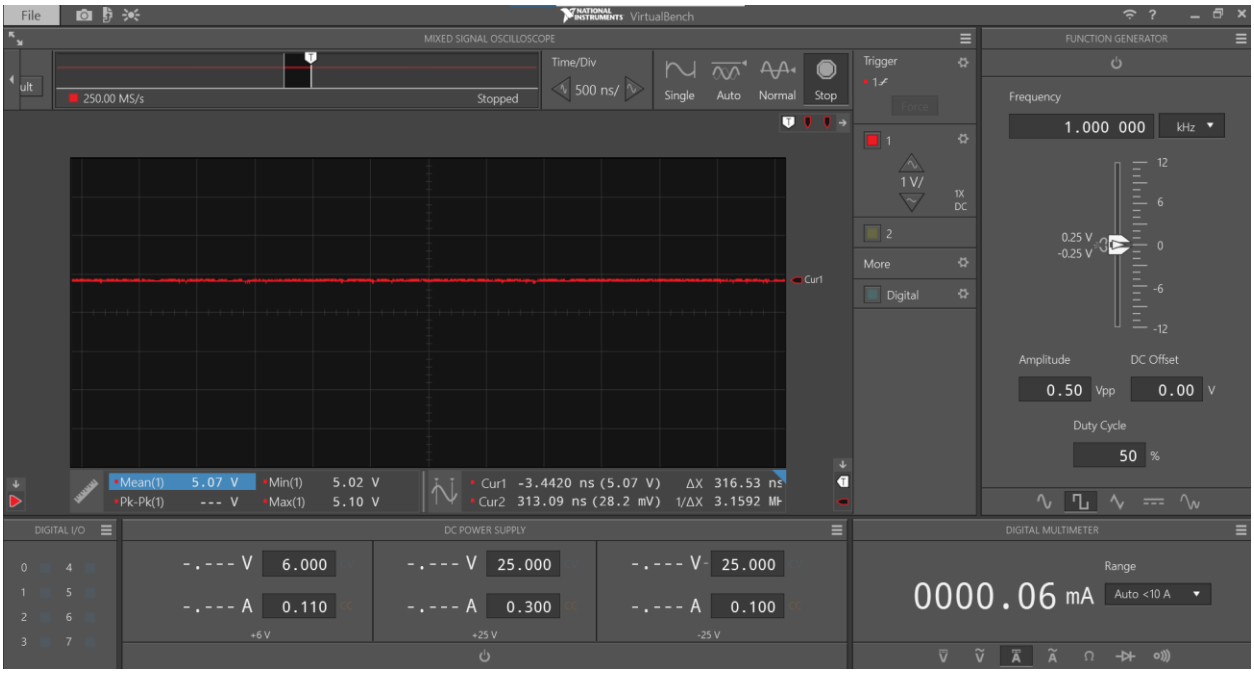

Figure 24: 5V Rail Measurement

After the supplies had been proven acceptable, the button subsystems were tested. This involved connecting the MSP432 to the board, placing each button's shunt into place, and reading the value. This showed that every button was reading as closed, i.e., the voltage at the pin was low. By measuring the voltage over the resistor going to the microcontroller we determined that current was flowing through the resistor regardless of the button's state. This meant that the transient voltage suppression diodes had been installed backwards and were being grounded. Fortunately, this was discovered early enough that we could have the diodes installed backwards of that in the schematic for the final board. Checking the button system again with the diodes reversed showed no voltage across the resistor and a correct reading from the MSP432. However, this was not the case for all buttons. The X-axis limit switch circuit would read an open button regardless of the state of the switch, so one of the future proof circuits was used instead.

During the testing of the buttons, we also determined that some of the GPIO pins on the MSP432 board were faulty. This was determined by shorting the intended pins to different GPIO pins and reading both. The other pin showed a correct reading while the original was always low. To fix this, a replacement board was used.

Next, the stepper motor drivers were tested to see if they were correctly driving the motors. This was done by connecting each driver's shunt and using the MSP to set the appropriate select, enable, direction, and step lines. Initially, the drivers were dormant. We soon realized that we had been testing the stepper motors with a different driver chip that did not have a reset pin. Once this was accounted for, the drivers performed as expected and we could successfully move the motors to a desired position.

The final subsystem tested was the electromagnet. This was done by applying a 10kHz PWM signal with a 30% duty cycle to the electromagnet's controller chip. This had been calculated to drive about at 300mA current through the magnet which was the manufacturers rated max current. We then placed the top of the pieces, to which a zinc screw was attached, to the electromagnet and felt for any attraction. At 30% duty cycle we observed no field. We then attached an ammeter in series with the electromagnet to determine the current flowing through the wire. This read at about 1mA. The duty cycle was then increased in increments of 10% until at 70% we found that the magnet would now strongly grab the pieces. The measured current however, never seemed to grow beyond 2mA, so we abandoned the ammeter and just assumed that the current was sufficient. However, when the magnet grabbed a piece, we noticed a high-pitched whine. It is our belief that the screw is changing the resistance of the magnets core and thus allowing more current to flow after it has grabbed a piece. Since we could not take an accurate current measurement, we could not prove this, but the magnet is never driven for long periods, and we never observed it to grow hot during use. Therefore, we determined this state of affairs to be acceptable.

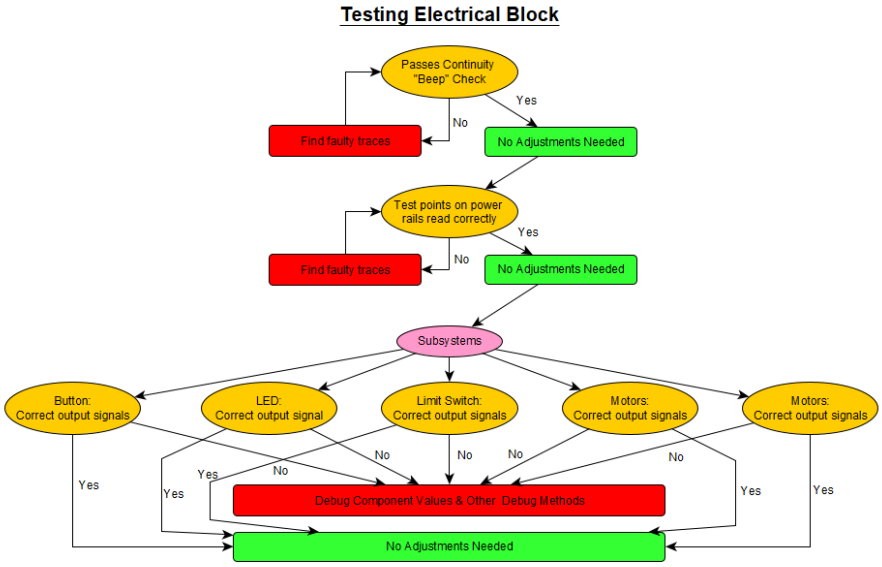

Figure 25: Electrical Test Plan

Sensor Board

Before the entire board could be tested, we first tested several magnets of different sizes and strengths to determine which would consistently trigger a switch within its square, but not trigger its neighbors and in what orientations. This was done by putting two neighboring switches into their slots and connecting them in series with a resistor and LED on a breadboard. The magnets were then placed over and around both switches and we observed where one or both LEDs lit up which indicated that the switch had closed. Eventually we decided upon 1/8" x 1/8" x ½" N42 bar magnets as the most consistent [39].

Once the magnet was determined and the board assembled, we plugged the board into the PCB and tried to read the different positions by placing a magnet over the switch and examining what the MSP432 was reading for each position. This revealed several problems. Firstly, we discovered that we had erroneously connected the power pin and the data pin on the row multiplexor as shown in Figure 26: Problem in Row Multiplexer. This meant the chip was unpowered, and thus was giving very unreliable data. To solve this problem, we removed the chip from the board and connected jumper wires to eight unused GPIO pins. We also discovered that we should not have connected the return path on the columns (where the power is delivered) to ground, as we were effectively shorting our signal to ground. Simply disconnecting the return path on the columns solved this issue.

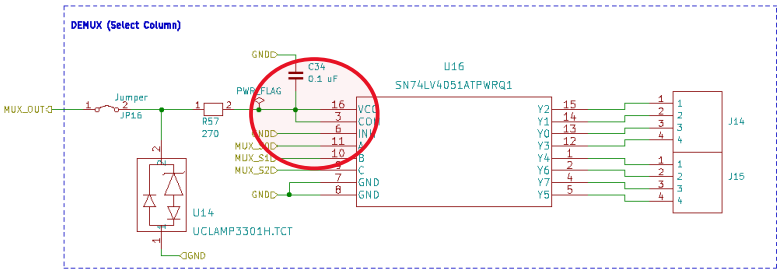

Figure 26: Problem in Row Multiplexer

Once we were reliably reading the board, we discovered that if a row had multiple magnets over them, and a neighboring row had but one switch closed, the entirety of the second row was erroneously reading high. To determine if this was a hardware problem or a software problem, we used a multimeter to check for continuity on opposite ends of open switches and discovered that there was a connection. Eventually, we determined that this was due to voltage "leaking" from the a given row through a closed switch on another row, so that no matter which column was selected, so long as there was a closed switch somewhere in both, both would always read high. An example of this is shown in Figure 27. To correct this, we added a 1N4001 diode in series with each reed switch (Figure 28). This prevents anything from one row going into another row through an unselected switch. We likely should have used a 1N4148 switching diode instead, but we could only acquire the 1N4001 in sufficient supply on short notice. This meant that we had to scan the board at a slower rate, but it ended up still being fast enough that there was no noticeable delay. This solved the problem, and after slowing down the speed at which we scanned through the columns, the board could accurately and consistently detect magnets at each switch.

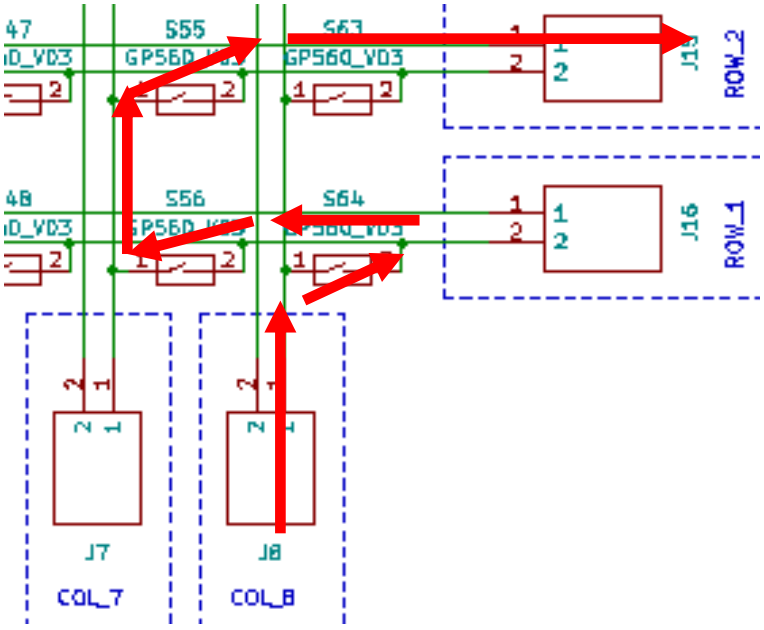

Figure 27: Erroneous Current Flow

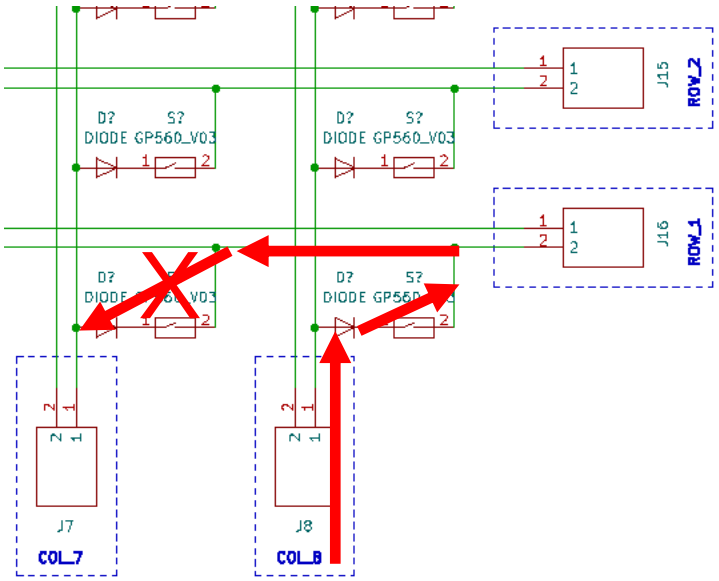

Figure 28: Corrected Sensor Board Layout

Software

Software was tested continuously to reach the final product. The earlier stages of software testing mostly involved testing the UART module on the MSP432 to ensure it could communicate reliably with the Raspberry Pi. The Python script running on the Raspberry Pi included printing and logging of the messages it received to support this process. When unexpected errors occurred that involved the Raspberry Pi, the logs could be accessed via Secure Shell (SSH) to see what went wrong. On the MSP432, Code Composer's debugger was used to access register and variable values or set breakpoints when unexpected behavior occurred.

The software test plan in Figure 29 tests the software for, among other things, proper LED indicators, UART communication, input handling, and motor control. It does this by simulating a power-on of the system followed by a turn made by the human. Further testing extended this to a power-on followed by many turns from both the player and robot, testing for behavior like recognizing captures and illegal moves throughout. This scenario was repeated many times, and when unexpected behavior like system faults, incorrect LEDs, or failed communication occurred, the software was debugged using either of the tools previously mentioned: Code Composer's debugging tools or the Raspberry Pi's script logs.

In addition to testing the software for functionality, some values in the MSP432's code that controlled motor movement needed to be tested for accuracy. This meant ensuring the motors could accurately go to any tile on the X and Y axes and pick up any piece without missing nor crushing them on the Z-axis. This involved placing a particular piece on a tile and ensuring the gantry system electromagnet would move directly over the piece without crushing it. If the electromagnet was not centered above the piece, then the X and Y offset for the rank or file the piece was on was adjusted. If the electromagnet did not reach the piece or pressed down

too hard on it, the height offset for that piece was adjusted. This process was repeated until the software had offsets that would accurately pick up any piece on any tile without crushing them.

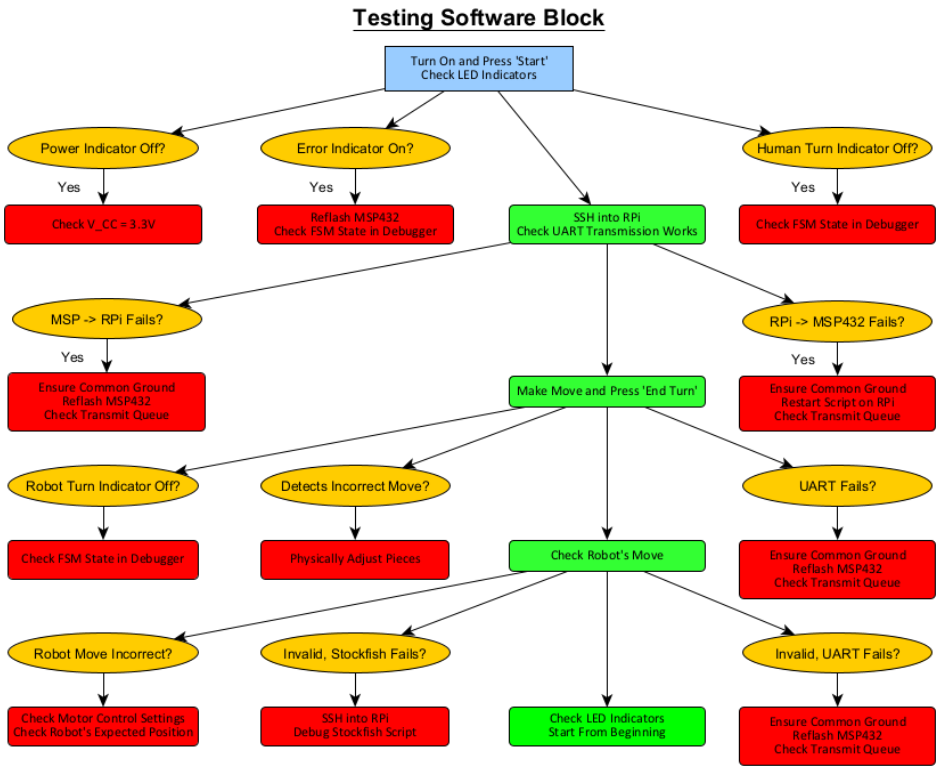

Figure 29: Software Test Plan

Final Results

Overall, the robot functioned well and met our initial goal: to create an autonomous robot that could play chess competently against a human. In the original proposal for this project, the team created a rubric by which to assess the success of the project. It is shown in Figure 30. Assessing these criteria, the motors always functioned well and moved reasonably swiftly, and the system was well-calibrated to account for different piece heights and moving to any square accurately. As a result, the robot consistently took less than 20 seconds per movement (defined as time taken to pick up, move, and place a piece from one tile on another) and very rarely missed picking up pieces. In a sample taken on demo day, the robot only missed pieces 2 times across many games (61 moves, 3.3% miss). When it did miss pieces, this was more due to inconsistency in the height of the protruding screw on certain pieces and not poor Z-axis calibration. Over time, the screws would work themselves deeper into the piece thus lowering the total height. In these situations, the missed piece's screws would need to be adjusted slightly with a screwdriver. The chess robot never moved the wrong piece; the X and Y axes were calibrated very well to avoid this. The robot could handle all special moves appropriately, including capturing, castling, en passant, and queen promotion. The sensor network usually detects all pieces, but occasionally they would need to be adjusted for the robot to recognize their presence. By this assessment, this project earned 3 points for speed, 3 points for accuracy, 3 points for correctness, 3 points for level of play,

and 2 points for sensor accuracy. So, this project scored a 14/15, which is an A by the evaluation rubric in Figure 30.

Points	Speed	Accuracy (Physical Movement)	Correctness (Chess Move)	Level of Play	Sensor Accuracy
3	< 20s per movement	Only misses picking up one or fewer pieces per game	Moves the wrong piece once or fewer times per game	Robot handles special moves (capturing, castling, en passant, and queening (once))	Robot detects all pieces and identifies them correctly every time
2	< 40s per movement	<=2 pieces per game	<=2 pieces per game	Handles >= 2 special moves	Detects some pieces and identifies >= 50% correctly
1	< 1min per movement	<= 4 pieces per game	<= 4 pieces per game	Plays chess but cannot handle special moves	Detects some pieces and identifies < 50% correctly
0	> 1min per movement	> 4 pieces per game	> 4 pieces per game	Does not play chess	Cannot detect pieces

Points	Grade
13-15	A
9-12	B
6-8	C
3-5	D
0-2	F

Figure 30: Evaluation Rubric

Although this project met the criteria for an A, it was not without flaws. While the Z-axis motor was well-calibrated to avoid overdriving into pieces with the electromagnet, it would sometimes, albeit rarely, miss a piece by not moving the electromagnet sufficiently close to the protruding screw. This occurred more frequently the longer we played, as the metal screws were driven into the heads of the plastic pieces when the electromagnet moved them. If we were to do this project again, we would 3D print custom pieces that would remove the need for magnetic screws or use metallic pieces.

Additionally, the sensor network sometimes failed to read pieces which were present, often ones recently moved by the robot. This is because the reed switches depend on the orientation of the magnet above them. While pieces could be set up with the optimal orientation relative to the reed switch, the electromagnet could sometimes move the piece while rotating it from its initial position. If the piece was rotated into a poor orientation and placed back down, then a legal move would manifest itself as an illegal move since the robot could not detect the piece. Then, the suspected piece would need to be reoriented to be recognized by the robot and proceed with the game.

Costs

Original Budget and final spending can be seen in Budget. Our original budget underestimated the amount needed for electrical assembly. Since our PCB was mainly surface mounted parts, it was unfeasible for the team to solder all the components on by hand. The company that the class uses for PCB assembly, WWW Electronics [40], charges 50 cents per component to assemble, so with more than 200 components on the board, it would cost a bit more than $100 for full assembly. This means that only one board was able to be fully assembled. The chess mat board and the chess pieces never came in, so we had to purchase another set for $20. OpenBuilds was the main distributor for mechanical parts such as aluminum, screws, nuts, gantry kits, mounting plates and more. The original budget also accounts for supplies that we already had, like stepper motors, belts, pullies, cable drivers and stepper drivers. When mass manufacturing, it would be simpler to have one large 17x17 PCB for the sensor network rather than 4 individual PCBs that we ended up going with. The 3D printed parts did not factor into the budget, but for mass manufacturing, these parts would need to be made of something more sturdy and easier to make. Molds could be made for the parts, and then those parts could be made of aluminum. Our original budget also did not include having to buy a new MSP432. This purchase was made of a need for more working GPIO pins. Table 5 shows the total costs for this project if we had to buy everything from scratch, the total expected for a mass manufacturing setting, and the actual total amount spent. Ideally, the project would have stayed under the $500 original budget, but to obtain full functionality in a timely manner, extra expenses had to be made.

Table 5: Final Budget Calculations

Total Theoretical	Total Bulk	Total Actual
1035.08400	952.56471	765.198

Future Work

In terms of revisions to the project, we would want to replace the reed switches with hall effect sensors. The field for the switches is parallel to the switch, whereas with hall effect sensors the field is normal to the sensor. This means that a piece could be placed in any orientation without issue. We also had three buttons on the PCB that ended up doing the same thing. We would want to move one of them for reset to in front of the board where the player is and reduce their number. Additionally, the PCB is exposed, which presents a possible electrical safety risk. We would need to move the indicator light next to the signal light, then put the board within a proper enclosure. This removes any safety concerns. Lastly, having two separate computers, the MSP432 and the Raspberry Pi is a bit redundant. We could try to modify the Stockfish chess engine to run on the MSP432 and eliminate the Raspberry Pi to reduce complexity.

In terms of adding new features, an extension to this project could support “chess variants.” Chess variants are games derived from standard chess, but with slightly different rules. For example, “Fischer random chess” randomizes the back ranks of both players to the same arrangement, and “3-check” changes the goal from placing the king in checkmate to placing the king in check 3 times. With some additional UART instructions and motor instructions, one could set up the software to support both variants, among many others.

References

[1] "Raspberry Pi 3 Model A+," *Raspberry Pi*. https://www.raspberrypi.com/products/raspberry-pi-3-model-a-plus/ (accessed Sep. 26, 2022).

[2] "MSP432E401Y," *Texas Instruments*. https://www.ti.com/product/MSP432E401Y (accessed Sep. 26, 2022).

[3] "Stockfish." official-stockfish, Sep. 25, 2022. Accessed: Sep. 24, 2022. [Online]. Available: https://github.com/official-stockfish/Stockfish

[4] H. J. R. Murray, *A History of Chess: The Original 1913 Edition*. Oxford, England: Oxford University Press, 2015.

[5] "No. 2765: The Mechanical Turk." https://www.uh.edu/engines/epi2765.htm (accessed Sep. 26, 2022).

[6] D. A. Christie, T. M. Kusuma, and P. Musa, "Chess piece movement detection and tracking, a vision system framework for autonomous chess playing robot," in *2017 Second International Conference on Informatics and Computing (ICIC)*, Nov. 2017, pp. 1–6. doi: 10.1109/IAC.2017.8280621.

[7] P. K. Rath, N. Mahapatro, P. Nath, and R. Dash, "Autonomous Chess Playing Robot," in *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, Oct. 2019, pp. 1–6. doi: 10.1109/RO-MAN46459.2019.8956389.

[8] "Can't Help Myself | The Guggenheim Museums and Foundation." https://www.guggenheim.org/artwork/34812 (accessed Sep. 27, 2022).

[9] *How We Made an NFC Chess Board*, (Dec. 18, 2018). Accessed: Sep. 27, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=ZltkZSNX_d4

[10] "Student Creates Robot That's One Move Ahead on Chess Board," *YouTube*. https://www.rose-hulman.edu/news/2021/student-creates-robot-that-is-one-move-ahead-on-chess-board.html (accessed Sep. 27, 2022).

[11] "GitHub: Where the world builds software," *GitHub*. https://github.com/ (accessed Sep. 26, 2022).

[12] "KiCad EDA." https://www.kicad.org/ (accessed Sep. 26, 2022).

[13] "CADLAB.io | Visual collaboration and version control platform for your PCB." https://cadlab.io/ (accessed Sep. 27, 2022).

[14] "3D CAD Design Software | SOLIDWORKS." https://www.solidworks.com/home-page-2021 (accessed Sep. 26, 2022).

[15] "Original Prusa i3 MK3S+ | Original Prusa 3D printers directly from Josef Prusa," *Prusa3D by Josef Prusa*. https://www.prusa3d.com/category/original-prusa-i3-mk3s/ (accessed Dec. 13, 2022).

[16] "Ultimaker S3: Easy-to-use 3D printing starts here," *https://ultimaker.com*. https://ultimaker.com/3d-printers/ultimaker-s3 (accessed Dec. 13, 2022).

[17] "PrusaSlicer | Original Prusa 3D printers directly from Josef Prusa." https://www.prusa3d.com/page/prusaslicer_424/ (accessed Dec. 13, 2022).

[18] "Ultimaker Cura: Powerful, easy-to-use 3D printing software," *https://ultimaker.com*. https://ultimaker.com/software/ultimaker-cura (accessed Dec. 13, 2022).

[19] "CCSTUDIO IDE, configuration, compiler or debugger | TI.com." https://www.ti.com/tool/CCSTUDIO (accessed Sep. 26, 2022).

[20] "Visual Studio Code." Microsoft. Accessed: Sep. 26, 2022. [Online]. Available: https://code.visualstudio.com/

[21] D. Paraskevas, K. Kellens, A. Voorde, W. Dewulf, and J. Duflou, “Environmental Impact Analysis of Primary Aluminium Production at Country Level,” *Procedia CIRP*, vol. 40, pp. 209–213, Dec. 2016, doi: 10.1016/j.procir.2016.01.104.
[22] “Guides to Pollution Prevention: The Printed Circuit Board Manufacturing Industry,” p. 83.
[23] D. Xiang, P. MOU, J. Wang, G. Duan, and H.-C. Zhang, “Printed circuit board recycling process and its environmental impact assessment,” *Int. J. Adv. Manuf. Technol.*, vol. 34, Oct. 2006, doi: 10.1007/s00170-006-0656-6.
[24] “UCI protocol,” *WBEC Ridderkerk*, 2004. http://wbec-ridderkerk.nl/html/UCIProtocol.html (accessed Sep. 26, 2022).
[25] Millennium Circuits Limited, “GUIDE TO IPC STANDARDS FOR PCBS.” [Online]. Available: https://www.mclpcb.com/blog/ipc-standards-for-pcbs/#:~:text=What%20are%20the%20IPC%20standards,%2C%20assembly%2C%20packaging%20and%20more.
[26] “FreeDFM.” [Online]. Available: https://www.4pcb.com/free-pcb-file-check/index.html
[27] Telecommunications Industry Association, “Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange,” TIA-232-F, 1997. [Online]. Available: https://www.tc.faa.gov/its/worldpac/Standards/eia-tia/tia%20232f.pdf
[28] “UL Standard | UL 4600.” https://www.shopulstandards.com/ProductDetail.aspx?productid=UL4600 (accessed Sep. 27, 2022).
[29] “Consumer Product Safety Improvement Act,” Public Law 110–314, 2008. [Online]. Available: https://www.cpsc.gov/s3fs-public/cpsia.pdf
[30] “ASTM International - ASTM F963-07 - Standard Consumer Safety Specification for Toy Safety | Engineering360.” https://standards.globalspec.com/std/3828113/ASTM%20F963-07 (accessed Sep. 27, 2022).
[31] M. Barr, “Embedded C Coding Standard.” Barr Group, 2018. Accessed: Dec. 12, 2022. [Online]. Available: https://barrgroup.com/sites/default/files/barr_c_coding_standard_2018.pdf
[32] L. T. Jones, A. Howden, M. S. Knighton, A. Sims, D. L. Kittinger, and R. E. Hollander, “Robot Computer Chess Game,” 4,398,720 Accessed: Dec. 10, 2022. [Online]. Available: https://patentimages.storage.googleapis.com/32/ae/42/c43a9d7aac8229/US4398720.pdf
[33] H. Crozier, “Chess Game and Method,” 6,446,966 [Online]. Available: https://patentimages.storage.googleapis.com/3d/fd/b7/60ace1484a1996/US6446966.pdf
[34] M. KAUFMANN and H. THUT, “Robot Arm with a Shearing Drive, and a Gantry Robot” [Online]. Available: https://www.quickcompany.in/patents/robot-arm-with-a-shearing-drive-and-a-gantry-robot#documents
[35] J. Fletcher, “An Arithmetic Checksum for Serial Transmissions,” *IEEE Trans. Commun.*, vol. 30, no. 1, 1982, Accessed: Dec. 01, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/1095369
[36] “24V Power Supply.” [Online]. Available: https://en.globtek.com/model/ict-ite-power-supply/desktop-external/gt-46240-24vv-x-x-t2/w71120#output_connectors
[37] Kycon, “Thru-hole DC Power Jack, High Current: 8A for 2.0 and 2.5 mm plugs, RoHS Compliant.” [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/Kycon%20PDFs/KLDHCX-8-0202-x.pdf

[38] Texas Instruments, “LM2576xx Series SIMPLE SWITCHER 3-A Step-Down Voltage Regulator.” Texas Instruments. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm2576hv.pdf?HQS=dis-mous-null-mousermode-dsf-pf-null-wwe&ts=1670925206237&ref_url=https%253A%252F%252Fwww.mouser.com%252F
[39] K&J Magnetics, Inc., “B228.” [Online]. Available: https://www.kjmagnetics.com/proddetail.asp?prod=B228
[40] “Three W - WWW Electronics Incorporated.” [Online]. Available: https://3welec.com/

Appendix

Appendix A - UCI Notation

"Universal Chess Interface" (UCI) is a communication protocol that allows chess engines to communicate with user interfaces. In this project, the most important aspect of UCI is the way it represents moves. Moves in UCI notation are represented by 4-5 characters, explained in Table 6.

Table 6: Explanation of UCI Notation

Character #	Character	Description
0	a, b, c, d, e, f, g	Initial File
1	1, 2, 3, 4, 5, 6, 7, 8	Initial Rank
2	a, b, c, d, e, f, g	Final File
3	1, 2, 3, 4, 5, 6, 7, 8	Final Rank
4	q	Promotion (if applicable)

So, a move like "e2e4" would translate to "move the piece on tile e2 to e4; if there is a piece present at the final tile, capture it." For promotions (the only type of move which can be 5 characters long), a move like "c7c8q" would translate to "move the pawn on tile c7 to c8, then promote it to a queen."

Appendix B – Schematics

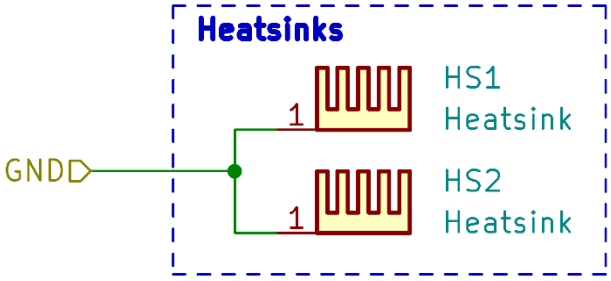

Figure 31: Heatsink schematic

Appendix C – Budget

	Part Name	Estimated Cost	Need to Buy (y/n)	Quantity
CHESS	Chess Board	$4.99	y	1
	Chess Pieces Set	$12.95	y	1
FRAME	Stepper Motors	$17.99	n	2
	Servo Motor	$43.99	n	1
	Aluminum 80/20 20x20	$5.49	n	6
	Aluminum 80/20 20x40	$6.99	n	1
	Screws	$1.49	y	5
	Nuts	$0.99	y	5
	V-Slot Gantry Cart Kit (Small)	$31.99	n	1
	V-Slot universal gantry plate	$11.99	y	1
	Stepper motor mounting plate	$6.99	y	2
	Idler mounting plate	$6.99	y	2
	Idler pulley kit	$5.99	y	2
	Belts	$2.49	n	5
	Pulley	$6.99	n	2
	Electromagnet (for arm)	$25.00	y	1
	Magnets (for pieces)	$0.45	y	32
ELECTRICAL	PCB	$33.00	y	2
	Various Electrical Components	$50.00	y	1
	Cable Carrier	$13.50	n	1
	Stepper Driver	$15.49	n	2
Sensor Network	Sensors	$150.00	y	1
Compute	MSP 43X	$17.28	n	1
	Rasperry Pi	$34.99	n	1
Misc	Emergency Fund	$50.00	y	1
Total	Worst Case Scenario			$712.74
	Expected			$437.67

Figure 32: Original Budget

Table 7: Final Budget

Item	Quantity	Per Unit Price	Unit Price Per 10k	Total Cost Per Item	Total Cost Per Item - bulk	Bought (y/n)
chess board	1	18.99000	18.99000	18.99000	18.99000	y
SKU: 621	1	11.99000	11.99000	11.99000	11.99000	y
SKU: 575	2	6.99000	6.99000	13.98000	13.98000	y
SKU: 570	2	6.99000	6.99000	13.98000	13.98000	y
SKU: 550	2	5.99000	5.99000	11.98000	11.98000	y
SKU: 946-pack	1	0.99000	0.99000	0.99000	0.99000	y
SKU: 20-pack	1	1.39000	1.39000	1.39000	1.39000	y
SKU: 130-pack	1	1.59000	1.59000	1.59000	1.59000	y
SKU: 10-Pack	1	0.99000	0.99000	0.99000	0.99000	y
20 Degree Pressure Angle Plastic Gear	1	3.00000	3.00000	3.00000	3.00000	y
20 Degree Pressure Angle Gear Rack, 0.8 Module	2	4.40000	4.40000	8.80000	8.80000	y
Rubber bumper	4	4.81000	4.81000	19.24000	19.24000	y
M5 8mm screws	1	0.99000	0.99000	0.99000	0.99000	y
M5 10mm screws	1	1.09000	1.09000	1.09000	1.09000	y
M5 Tee nuts	2	2.99000	2.99000	5.98000	5.98000	y
PCB Board	1	66.00000	66.00000	66.00000	66.00000	y
PCB Assembly	1	163.00000	163.00000	163.00000	163.00000	y
C1812X474K2RACAUTO	3	1.15000	0.39259	3.45000	1.17777	y
C1210C822K2RACAUTO	12	0.65000	0.18585	7.80000	2.23020	y
GRM033R61A103KA01D	9	0.10000	0.00279	0.90000	0.02511	y
860020573008	1	0.15000	0.08100	0.15000	0.08100	y
SEK331M050ST	1	1.75000	0.62100	1.75000	0.62100	y
31DQ04	1	0.38000	0.10084	0.38000	0.10084	y
PPTC102LFBN-RC	4	1.30000	0.64870	5.20000	2.59480	y
OQ0432500000G	11	0.76000	0.14589	8.36000	1.60479	y
OQ0212500000G	10	0.46000	0.07469	4.60000	0.74690	y
KLDHCX-8-0202-A	1	2.14000	1.12130	2.14000	1.12130	y
SRR1210-681M	1	1.41000	0.60354	1.41000	0.60354	y
HV-5RGB25	1	0.41000	0.32703	0.41000	0.32703	y
TNPV08051M00BEEA	3	0.83000	0.24708	2.49000	0.74124	y
WSL2010R4000FEA	6	0.93000	0.31416	5.58000	1.88496	y
ERJ-PB6B3002V	3	0.40000	0.04305	1.20000	0.12915	y
RMCF1206FT10K0	37	0.10000	0.00365	3.70000	0.13505	y
SDR03EZPF2201	2	0.16000	0.02225	0.32000	0.04450	y
CR0402-FX-5620GLF	12	0.10000	0.00170	1.20000	0.02040	y

RCV2512100KFKEGAT	6	1.39000	0.41922	8.34000	2.51532	y
SDR10EZPJ271	5	0.23000	0.02451	1.15000	0.12255	y
PTS636 SL43 LFS	3	0.10000	0.05650	0.30000	0.16950	y
5010	2	0.42000	0.17044	0.84000	0.34088	y
5126	21	0.42000	0.17044	8.82000	3.57924	y
5011	3	0.42000	0.17044	1.26000	0.51132	y
5012	2	0.42000	0.17044	0.84000	0.34088	y
LM2576SX-3.3/NOPB	3	4.06000	2.16720	12.18000	6.50160	y
3873	1	9.95000	9.95000	9.95000	9.95000	y
LM2576HVS-5.0/NOPB	1	9.24000	5.49150	9.24000	5.49150	y
CD74HC4051PWT	2	1.24000	0.52500	2.48000	1.05000	y
RCV2512100KFKEGAT	6	1.39000	0.41922	8.34000	2.51532	y
CL10B104KB8NNNC	8	0.10000	0.00513	0.80000	0.04104	y
EEE-FP1E471AP	5	1.16000	0.37228	5.80000	1.86140	y
EAST2012RA2	2	0.27000	0.04202	0.54000	0.08404	y
CDBA540-HF	4	0.44000	0.13500	1.76000	0.54000	y
3588	1	1.33000	0.64309	1.33000	0.64309	y
ATS-52150P-C1-R0	2	10.96000	7.61288	21.92000	15.22576	y
DRV8210DRLR	1	0.89000	0.33750	0.89000	0.33750	y
USB1130-15-A	1	0.54000	0.38037	0.54000	0.38037	y
PH1-02-UA	20	0.10000	0.01400	2.00000	0.28000	y
PA4344.104NLT	1	4.44000	2.17933	4.44000	2.17933	y
CHP2512AFX-47R0ELF	1	1.00000	0.36407	1.00000	0.36407	y
SDR03EZPF1500	1	0.16000	0.01730	0.16000	0.01730	y
UCLAMP3301H.TCT	16	0.64000	0.22950	10.24000	3.67200	y
JS102011SCQN	1	0.85000	0.42069	0.85000	0.42069	y
UCLAMP3301HCT-ND	20	0.56100	0.22950	11.22000	4.59000	y
350-4316-ND	1	5.14000	2.20495	5.14000	2.20495	y
WSLE-.40CT-ND	10	0.80500	0.31416	8.05000	3.14160	y
2057-PH1-02-UA-ND	20	0.03700	0.01400	0.74000	0.28000	y
2073-USB1130-15-ACT-ND	1	0.79000	0.38037	0.79000	0.38037	y
2092-KLDHCX-8-0202-B-ND	1	2.16000	1.12130	2.16000	1.12130	y
732-8942-1-ND	6	0.15000	0.08100	0.90000	0.48600	y
609-3799-ND	20	1.09500	0.53308	21.90000	10.66160	y
399-C1210C822K2RACAUTOCT-ND	1	0.65000	0.18585	0.65000	0.18585	y
1276-1000-1-ND	8	0.10000	0.00513	0.80000	0.04104	y
399-C1812X474K2RACAUTOCT-ND	1	1.15000	0.39259	1.15000	0.39259	y
17x17 board	1	0.00000	15.74210	0.00000	15.74210	n
2x8 board	5	7.46000	0.00000	37.30000	0.00000	y

Reed Switches	65	0.47720	0.31603	31.01800	20.54195	y
Diodes	64	0.18900	0.02877	12.09600	1.84128	n
polycarbonate - 18x24	2	0.00000	47.98000	0.00000	95.96000	n
pinch point stickers - pack of 10	1	11.55000	11.55000	11.55000	11.55000	y
E-Stop	1	38.24000	25.78980	38.24000	25.78980	y
Magnets - B228	34	0.52000	0.52000	17.68000	17.68000	y
Next Turn Button	1	11.69000	11.69000	11.69000	11.69000	y
MSP432	1	53.19000	53.19000	53.19000	53.19000	y
Raspberry Pi	1	34.99000	34.99000	34.99000	34.99000	n
Stepper Motors	2	17.99000	17.99000	35.98000	35.98000	n
Servo Motor	1	43.99000	43.99000	43.99000	43.99000	n
Aluminum 80/20 20x20	6	5.49000	5.49000	32.94000	32.94000	n
Aluminum 80/20 20x40	1	6.99000	6.99000	6.99000	6.99000	n
Cable Carrier	1	13.50000	13.50000	13.50000	13.50000	n
Stepper Driver	2	15.49000	15.49000	30.98000	30.98000	n
Belts	5	2.49000	2.49000	12.45000	12.45000	n
Pulley	2	6.99000	6.99000	13.98000	13.98000	n
V-Slot Gantry Cart Kit (Small)	1	31.99000	31.99000	31.99000	31.99000	n
				Total Theoretical	**Total Bulk**	**Total Actual**
Totals				**1035.08400**	**952.56471**	**765.198**