C.H.E.S.S.B.O.A.R.D.

An Interactive Chess Learning Aid

John Berberian Jr.
B.S. Electrical Engineering
University of Virginia
Charlottesville, Virginia, United States
ccg3sr@virginia.edu

Kevin Dang
B.S. Electrical Engineering
University of Virginia
Charlottesville, Virginia, United States
ejj4wt@virginia.edu

Paul Diaz Karhnak
B.S. Computer Engineering
University of Virginia
Charlottesville, Virginia, United States
pkarhnak@gmail.com

Lourdes Leung
B.S. Electrical Engineering
University of Virginia
Charlottesville, Virginia, United States
mqw6nf@virginia.edu

Liam Timmins
B.S. Electrical Engineering
University of Virginia
Charlottesville, Virginia, United States
uhj6qw@virginia.edu

***Abstract*—The Chess Helper, Evaluator, and Study Supporter to Boost Observation, Acumen, Reasoning, and Deduction (C.H.E.S.S.B.O.A.R.D., referred to herein as CHESSBOARD) is a smart chessboard focused on streamlining the chess learning process for beginner to intermediate players. The CHESSBOARD provides an interactive learning experience that displays possible moves and hints with light-emitting diode (LED) feedback, validates moves according to standard rules, and transcribes games automatically. The bulk of the system is built inside a custom wood and acrylic enclosure and a separate 3D-printed container ("clock box") houses the computers. The CHESSBOARD uses linear Hall effect sensors to measure the magnetic field strength from custom magnetic chess pieces in order to detect their type and color. An MSPM0G3507 microcontroller reads from these sensors, detects game updates like a player making a move, lights up the LEDs in response to new game information, and forwards this new game information to a secondary Raspberry Pi computer. The Raspberry Pi interprets these updates, records new game conditions, and plays the changes against the Stockfish chess engine to get new lists of legal moves, new hints, and new special information as applicable. Every aspect of the system design was designed and built with users and their experience in mind.**

***Index Terms*—Chess, teaching, magnetic sensors, human computer interaction, consumer products.**

CONTENTS

This was the capstone design project (ECE 4440/4991) for the authors, performed in partial fulfillment of the graduation requirements for the Charles L. Brown Department of Electrical and Computer Engineering at the University of Virginia. We would like to acknowledge our excellent advisor, Prof. Adam Barnes.

List of Figures

List of Tables

I. Statement of Work

A. John Berberian Jr.

John was primarily responsible for the embedded firmware that ran on the MSPM0G3507 microcontroller ("MSPM0"), including the drivers, real-time operating system (RTOS), and communication logic. He was also the primary team member responsible for the power subsystem and its associated design features like communication and level shifting. He designed the filter component values for the sensor subsystem and carried out noise and switching time simulations in PSPICE. In addition, he assisted with soldering and hardware testing, and advised design decisions in other areas of the project. He helped with the design of the electronics contained in the clock

box, including the buttons and liquid crystal display (LCD) interface.

John was also involved to a lesser extent in the implementation of the Raspberry Pi system. His main contribution to the Raspberry Pi was debugging the issues with powering over the header pins, leading to the team's decision to switch to a Raspberry Pi 4 from a Raspberry Pi 5. Additionally, he wrote extra firmware to verify the functionality of each hardware subsystem, and helped debug hardware issues.

B. Kevin Dang

Kevin was the primary person working on the design of the enclosures. All enclosures were modeled in SolidWorks. For the main chessboard, the enclosure used both acrylic and wood, materials which required Kevin to perform machining on the laser cutter and woodworking machines, respectively. Additionally, Kevin 3D printed an I/O shield to cover the cutout for port access. Another significant area of Kevin's work encompassed PCB layout. Kevin was the primary person designing the I/O board. First, Kevin created a power budget using the maximum load of each device in the system to size the connectors and traces on the board. Then, Kevin helped with testing of the first revision of the I/O board which revealed many issues with mismatched pins caused by erroneous footprints. This meant that stray pins had to be bent and fly-wired. Kevin either replaced or manually modified footprints in a second revision of the board. Along with his work on the I/O board, Kevin assisted the other team-members with reviewing the layouts of the sensor board and clock box board.

C. Paul Diaz Karhnak

Paul was the primary team member responsible for operating and administering the Raspberry Pi, which included installing necessary packages like `stockfish`, installing necessary Python modules like `chess` and `pyserial`, and dealing with file-mapped UART through the Linux file system. Paul wrote the Stockfish wrapper code which interacted with Stockfish in a subprocess and engaged with the `chess` module to perform chess game state tracking and move registration (including move undo requests and game restarts). Paul worked closely with John on aspects of the MSPM0 code base by writing the first draft of the MSPM0 UART driver; reviewing John's application programming interface (API) and business logic on the MSPM0; adding UART task management into the main task; writing the first draft of pushbutton interrupt handlers; and debugging UART on the two devices. Lastly, Paul wrote the first draft of the data copying code on the Raspberry Pi to enable games to be written to removable persistent storage when connected to the Pi.

D. Lourdes Leung

Lourdes was primarily responsible for the design and testing of the piece detection system. In the early stages of the project, she conducted preliminary testing of the sensing system and decided on which type of sensors and magnets to purchase. She performed several rounds of testing to determine the heights for the magnets inside the chess pieces so that the sensing system could accurately differentiate between chess piece types. Using these results, Lourdes designed the chess pieces in AutoCAD and 3D printed them. Along with the piece detection system, she designed and produced schematics for the clock box boards. Lourdes also aided in the physical enclosure design and manufacturing process. During the manufacturing process, she helped solder several of the LED strips and sensor printed circuit boards (PCBs). While performing exhaustive testing and debugging after the hardware assembly, she identified the factors contributing to the range variations of the sensors and made several revisions to the chess piece designs and sensors PCBs.

E. Liam Timmins

Liam was primarily responsible for several aspects of the hardware design and assembly. For instance, Liam designed the printed circuit board (PCB) layouts for the sensor boards and clock box boards in KiCAD. These consisted of efficiently laying out components in order to mitigate the effects of noise and minimize dimensions to effectively fit within the enclosures. When filtering became a higher priority partway through the semester, he revised these designs to better accommodate the additions while maintaining the small form factor. Liam also contributed to the sensor testing process, determining the feasibility of the use of Hall effect sensors and the sensitivity to choose.

Beyond design and testing, Liam contributed to a significant portion of the hardware assembly, including soldering the LED arrays and all of the PCBs made for the project. This included soldering all of the clock box boards, most of the sensor boards, and parts of the I/O board. After the assembly was completed, he assisted in the hardware testing process, including sensor functionality in the context of the rest of the system. This process consists of tuning the individual sensors to best read the anticipated piece type from the sensor's output to minimize the chance of misreading said piece type. Liam also resoldered components when he localized issues during the hardware testing process, including bridges forming between V_{CC} and ground and faulty ICs.

II. Background

A. Initial Idea

As we worked to come up with ideas in Summer 2024, Liam suggested making an automatic chess transcriber. The original idea was that automatic transcription would be an aid observers of speed chess tournaments. With an automatic transcriber, they would not have to wait for manual transcription of the game or watch a video recording. Under a strict 15-week timeline (ECE 4440/4991 formally met from August 2024 to December 2024), we needed a project that would demonstrate our technical proficiency while being achievable in a relatively short period of time. We were confident in our capabilities to implement an automatic chess transcriber, felt that a successful design did not need to incorporate potential hazards like

heating components or high voltage,[1] and thought that there was ample flexibility to expand the scope of the project if needed. Based on this, the team selected the transcriber for further development.

B. Rationale

After re-evaluating the target audience, we reasoned that gearing the design toward a less competitive user, like a beginner, opened up more design options that would result in an more compelling overall product. A beginner-focused chess aid gave the project more utility, since on top of transcription for post-game analysis such a chess aid could interactively provide insights for players during games. For complete beginners, the chess aid could teach them standard rules; for players looking to learn more, they could study the strategy of an advanced chess engine like Stockfish. Since discussion with the team favored this new direction, we chose to modify the project to become a chess learning aid for beginner and intermediate players instead of simply a chess transcriber.

C. Comparison with Prior Art

Many similar "smart chessboard" projects have been designed in the past. Three examples were designed here at the University of Virginia: a self-rearranging chessboard [1], an autonomous chessboard [2], and an assistive chessboard [3]. The self-rearranging chessboard was a project that aimed to speed up the process of resetting a board to begin a new game. The project was focused on moving chess pieces and detecting pieces via visual means. The autonomous chessboard had the goal of creating a mechanism that could play against a human user. Like the self-rearranging chessboard, the autonomous chessboard's main feature was its ability to manipulate piece positioning. The system utilized binary reed sensors to detect changes in board state. The last of the prior capstone examples is the assistive chessboard, which sought to improve chess learning and practice. It did this by retaining the physical interaction of a chess set and adding visual elements and cues to assist a player in remembering strategy.

Of the three examined projects, the assistive chessboard was the most similar to our project: we also planned to make a learning aid, use LEDs to provide live feedback, and use Hall effect sensors to record changes in board state. Our project focuses less on the integration of the chess engine and more on the user experience. This was evident in our design decisions from the beginning. For example, by choosing to use analog Hall effect sensors, the team gained the ability to tell exactly what piece is present on a tile. Eliminating ambiguity in detecting piece types was important to correctly allow for certain "special moves" (*e.g.*, promotion, castling, *en passant* captures) and verify board conditions (*e.g.*, piece placement at the beginning of each game). The undo feature in our project provided a better experience for the players who are trying to learn through experimentation and want to see how different options can affect a game. Building the majority of our features into the product itself allowed for a more approachable experience that was meaningfully similar to playing a normal game of chess. The self-rearranging chessboard and autonomous playing chessboard had less in common with our goal but did have features that related to our design like identifying invalid or illegal moves, and recording the board state. The value of understanding these projects was in understanding the reasoning behind the design choices and hurdles they faced.

The most similar capstone design that we could find was from students at Carnegie Mellon University. In their project, "Tactile Chess" [4], the students created an accessible interface to online chess tools for blind people. They used a sensing system almost identical to ours (analog hall effect sensors, two layers of 8:1 multiplexers) and they ran Stockfish on a Raspberry Pi, but the remainder of their system differed. They communicated with the user through a web-based GUI and speakers rather than an LED array embedded in the board, and they performed move validation through online APIs rather than locally. Because the "Tactile Chess" team was writing for a web-based platform, their board requires internet access to work, and their move validation was much slower than ours. Also, the team chose to house their complete system in a single module, unlike our choice to house the computer systems in the separate clock box.

The team also looked into miscellaneous designs from the Internet. One of the designs was a demo application of giant magnetoresistance sensors (GMR) by NVE Corporation [5] which seemed to be built for transcription. As pieces moved, the move was displayed in chess notation on a small screen with time stamps and read aloud with a speaker. NVE's sensor implementation seemed to be digital like the assistive chessboard capstone project. A hobbyist implementation of a smart chessboard was also considered [6]. The hobbyist design had a hint feature and established wireless communication with another board to allow for remote play. LEDs illuminated tiles to communicate hints and to indicate where the remote opponent had moved. All moves had to be manually input with a keypad system. These two prior projects overlap with many features that we planned to integrate in our board. We wanted to use Hall effect sensors in a similar capacity to the NVE GMR sensors, although we used analog sensors to distinguish piece color and type. We wanted to have an LED array to show all available moves as well as recommended moves. Our board combined these features to make a product more suited for our target audience: a chessboard that can transcribe and inform without any input other than the movement of pieces like in a traditional chess game.

One challenge we anticipated was detecting when a turn was over. We initially planned to automatically detect turn completion without requiring players to hit a clock button. When we changed our focus to beginner players, we quickly realized that automatic detection would be inappropriate for our customer

[1]Other ideas proposed included a robot that wound transformers and an electric kettle. In addition to being perceived as infeasible, these designs were rejected due to involving potentially dangerous high voltage and heating, respectively.

base. Beginner players cannot always be expected to play their final choice of a move simply by placing a piece down once per turn: they might want to look at the resulting board state before they finalize their move, and might change their mind halfway through a turn. A player changing their mind might, in turn, mean that the player has to revert a new board state and make a different move that has a different result. After some consideration, we decided on an interface inspired by a chess clock: we designed and implemented the clock box with turn switching buttons on top to definitively end turns and with displays which can optionally act as chess clocks. Even when the chess clock is disabled, the players still need to use the turn-switching buttons to indicate to our system that the current turn is complete.

D. Relevant Coursework

Throughout the project, we drew upon our experience from the Electrical and Computer Engineering Fundamentals sequence (ECE 2630, ECE 2660, and ECE 3750) to design the project's PCBs. In the Fundamentals series, we practiced circuit analysis and simulation as well as PCB design, which were skills used to design the sensor, I/O, and clock box PCBs. Most of the team had taken Electromagnetic Fields (ECE 3209) and Electromagnetic Energy Conversion (ECE 3250), so they were comfortable in their understanding of what was fundamentally happening in the interaction between magnets in the chess pieces and the Hall effect sensors in the board. Every member of the team had taken the Introduction to Embedded Computer Systems course (ECE 3430), which taught the essentials of embedded systems programming in C. Two team members, including the primary embedded programmer on the team (Paul), had taken Advanced Embedded Computing Systems (ECE 4501): a course which teaches more complex embedded programming in C, including RTOS kernel development. The two embedded systems courses (particularly ECE 4501) were immediately useful in the project: they enabled us to orchestrate the RTOS which interprets changes in the CHESSBOARD's sensor readings and manages the associated changes in chess game logic. FreeRTOS was deployed on the MSPM0 and skills from ECE 4501 were used to understand the FreeRTOS kernel and analyze aspects such as its multitasking behavior, methods of software interaction with peripherals, and thread and process management.

III. Project Description

A. Performance Objectives and Specifications

The team designed CHESSBOARD to be an intuitive learning resource for beginner and intermediate chess players. We intended for the primary means of communication to be an array of LEDs that light up specific tiles as needed. The value of CHESSBOARD is that basic information such as which pieces can move, where can they move, and when an illegal move has been made can be communicated in real time. Additionally, tools like post-game transcription and an undo move queuing system help improve the experience by giving the user the ability to easily analyze how they played and learn by trying out different options from one position.

The CHESSBOARD features a $20\text{in} \times 20\text{in} \times 1.5\text{in}$ board. The base is made of wooden materials like composite wooden panels and planks and is topped with layers of acrylic. To reduce the number of manufactured components, the tiles were made from one solid sheet of clear acrylic. The white tiles were etched into the acrylic and further sanded to whiten the surface appearance. A sheet of window tint underneath the acrylic gives the board the appearance of having black and white tiles while maintaining the translucent characteristics necessary for observing the LEDs underneath the tiles (Fig. 1).

To teach players the movement rules of each chess piece, the LEDs light up to show all possible tiles to which a player may move a piece when it is lifted. The LEDs are placed beneath the Hall effect sensors. The Hall effect sensors are used to detect which piece has been picked up; specifically, they observe the change in a square's output voltage from a bin associated with that specific chess piece to the bin associated with an empty square. Each chess piece has a slotted magnet in its center at a varying distance from the base, producing a specific voltage output from the Hall effect sensors. Our goal was to make the response time of the entire system faster than human reaction time (around 250ms). We were able to exceed this goal: based on our calculations, we believe our system is able to keep response times under 150ms.

Another main feature that helps players develop their skills is the hint button. Once pressed, tiles light up to display the best possible move a player can make during their turn. Once players learn the rules of the game, they can start to learn new strategies using the suggested moves. Because the goal of CHESSBOARD is to be a learning aid, the board also features an undo button to allow players to retry their last move. As their skills develop to a more intermediate level, users may practice faster thinking during a game with the use of the built-in chess clock. When a player's clock starts, that clock will count down when the clock is reset or when the opposing player ends their turn. The chess clocks are displayed on seven-segment LCD screens that can display six digits for the remaining time in hours, minutes, and seconds.

The last feature is the automatic chess transcriber that logs each move as it is made and thus immediately provides a full list of moves for each game. Using this transcription, players may analyze how the game played out and study or revise their strategies to become better chess players. The transcription may be retrieved by plugging a thumb drive into the USB port on the side of the clock box. The Raspberry Pi automatically recognizes the new device and copies recent transcriptions (stored as text files) onto the thumb drive. While this requires that the thumb drive be formatted with a supported file system, the copying script assumes that every thumb drive is formatted for FAT32, which is a common and widely supported format.

B. Block Diagram and Functionality

The system is physically split into two modules: the chessboard itself and the separate clock box. The two modules are

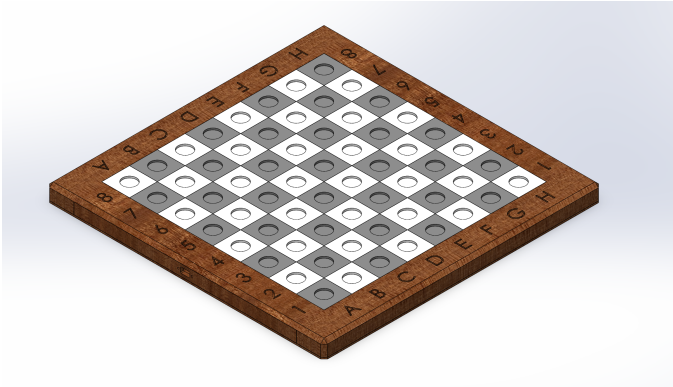


Fig. 1. Initial CAD model of CHESSBOARD physical structure made for project proposal.

connected by a high-density D-Bus 15-pin (DE15) connector. The entire system is powered by an external 5V DC power supply, which plugs into a barrel jack on the chessboard I/O PCB. The block diagrams for the chessboard and the clock box are shown in Fig. 2 and Fig. 3, respectively. In this section, we traverse the block diagram and explain how the components work together to produce the intended functionality.

The user expects to move pieces and see the LEDs in the board respond to the movement. For this to be possible, we needed a mechanism for detecting the movement of a piece. We chose to put magnets in each piece and place Hall effect sensors under the squares of the board. Well-defined spacing and offsets for the magnets allowed us to detect the type and color of any piece on the board. Since our MSPM0 microcontroller did not have enough pins to uniquely connect to each square's sensor (64 total), we decided to place analog multiplexers in two stages (8:1 to select one column from each row, then 8:1 to select between the row results) to reduce the number of analog wires to just 1. This did, however, require six digital wires for selection ($64 = 2^6$). Because sensors we chose are ratiometric—they will scale their output domain to their power rails, within some range—we were able to choose which power supply level to place them on. The microcontroller requires 3.3V power (and the ADCs will range from 0 to 3.3V) so we decided to place the Hall effect sensors on 3.3V power.

The red-green-blue (RGB) LEDs we selected run off of 5V power and use a digital protocol for control which permits them to be daisy-chained to an arbitrary length. This protocol means that we only need to use two pins for communication with the LEDs: one for data in and one for a clock. The LEDs expect 5V logic levels but the microcontroller that we selected used 3.3V logic levels, so we needed a level shifter to translate the signals. After reviewing our power requirements, we decided it was prudent to isolate the 5V components from the 3.3V components to reduce power supply noise. The components we expected to be noisiest (the LEDs and the Raspberry Pi) were on 5V power and the components that are sensitive to noise (MSPM0 and sensors) are on 3.3V power, so we decided on an isolated 3.3V switched-mode power supply and an isolated level shifter. We expected that this would protect the sensor output signal from any voltage drops due to high currents through the 5V-ground.

The whole system is controlled by the MSPM0 microcontroller [7] [8] on a Launchpad development board [9]. The MSPM0 runs the code to measure the positions of the pieces, control the LEDs under the squares, respond to button presses, and keep a chess clock running. The MSPM0 has many integrated peripherals that we used for communication with the various system components. The data protocol our selected LEDs use is similar to a simplified version of the Serial Peripheral Interface (SPI), so we used one of the MSPM0's two built-in SPI hardware modules to offload the work of communication. SPI was likewise a logical choice to communicate with the LCDs, so our driver used the MSPM0's other SPI module. The Raspberry Pi 4 [10], which runs a chess engine for producing the hints, communicates with the MSPM0 via UART; both the Raspberry Pi 4 and MSPM0 have hardware support for UART, and both of the MSPM0's hardware SPI modules were already occupied with the previously mentioned uses. Though the Pi 4 uses 3.3V logic levels, it uses 5V power, so we needed an isolator within the UART link to keep the two power domains isolated. The Pi 4 also logs a text representation of the game (chess notation) to a USB flash drive if one is plugged into the system.

Because the digital signals to communicate with the LEDs were relatively high-frequency (1.6MHz), we expected to see some coupling between those signal lines and the sensor subsystem's analog signal lines. Fortunately, the frequencies present in the coupled noise were substantially higher than in the sensor signal. To attenuate the noise, we added low-pass Bessel filters, placing second-order filters after each row multiplexer and a fourth-order filter directly next to the analog-to-digital converter (ADC) pin on the microcontroller board. To avoid using inductors, we built the filters with operational amplifiers using the Sallen-Key topology. To attenuate any 60Hz environmental noise that we have picked up from power supplies, we also added first-order RC filters to each sensor line. This was in accordance with the stability recommendations in our Hall effect sensors' datasheet [11].

We wished to support users in their competitive development, so we decided to integrate a chess clock to optionally simulate the conditions of a competitive game. The time remaining for each player is displayed on two six-digit seven-segment LCDs. LCD segments must be driven with an AC signal, so we connected them to an LCD driver chip that could be chained and which used an SPI-like protocol.

Aside from the sensors, the main means of user input is through the button system. To prevent mechanical bouncing from generating erroneous signal edges on the general-purpose input/output (GPIO) pins (which could then be incorrectly interpreted as multiple button presses), we added an RC network and Schmitt trigger to each button as described in *Debounce a Switch* [12].

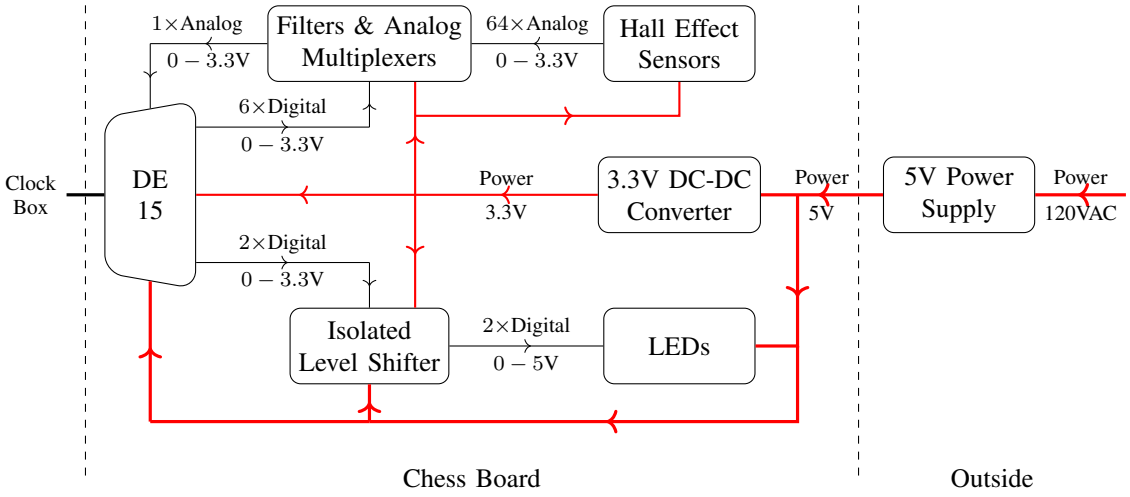


Fig. 2. Chess Board Block Diagram.

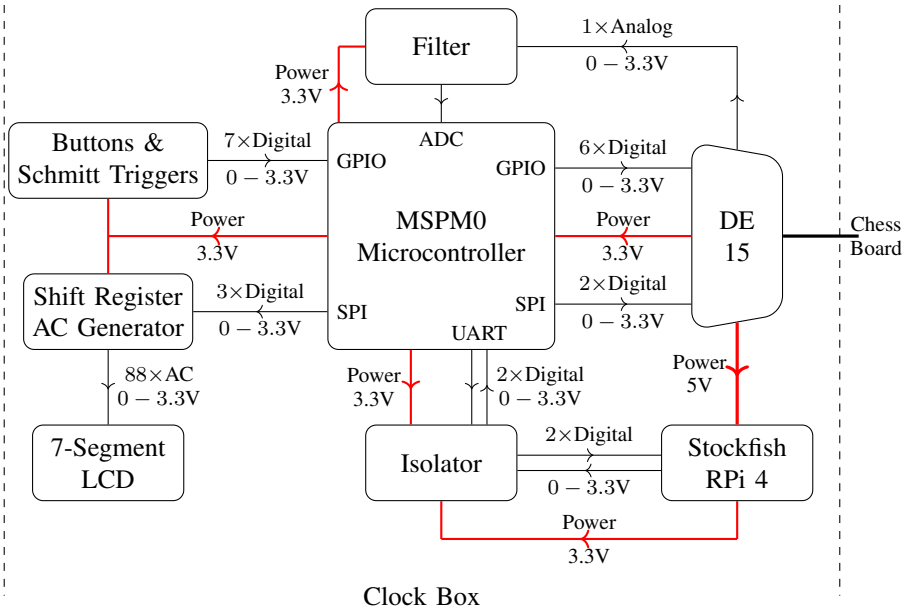


Fig. 3. Clock Box Block Diagram.

C. Hardware Design Details

1) Sensor PCBs: The initial criteria and components were for 8 boards with 8 sensors each. The sensors collect into an 8:1 mux with three select pins on each sensor board. The TMUX1208 [13] satisfied our requirements for a multiplexer: it had easily accessible documentation, a relatively low price, low on-resistance, and high switching speed. The decision to make the system as a unified PCB rather than as a network of wired components was made to introduce a greater level of regularity in spacing, which is critical for piece sensing accuracy. The schematic for this board can be seen in Fig. 4. The team tested a sample group of three sensors to establish sensitivity bins: the DRV5055A1QLPG ("A1"), DRV5055A3QLPG ("A3"), and DRV5055A4QLPG ("A4") [11]. These sensors were chosen due to their availability and price at the scale that we required in addition to satisfying our technical requirements for sensing. The A1 sensor was selected since the greater sensitivity allowed for the highest resolution within the expected height range (3mm to 13mm) of the six discrete magnet height bins. When we tested with the magnet we selected [14], this range gave us reasonable space between the different pieces' voltage bins while avoiding significant overlap. Each sensor required a bypass capacitor of at least 0.01μF according to the datasheet [11].

Initially, we planned to use a single set of voltage bins for all the squares. As the project moved into the manufacturing phase, it became clear that this was infeasible. Our final design for the height choices and their respective voltage outputs is visible in Tab. I. When we tested pieces with these magnet

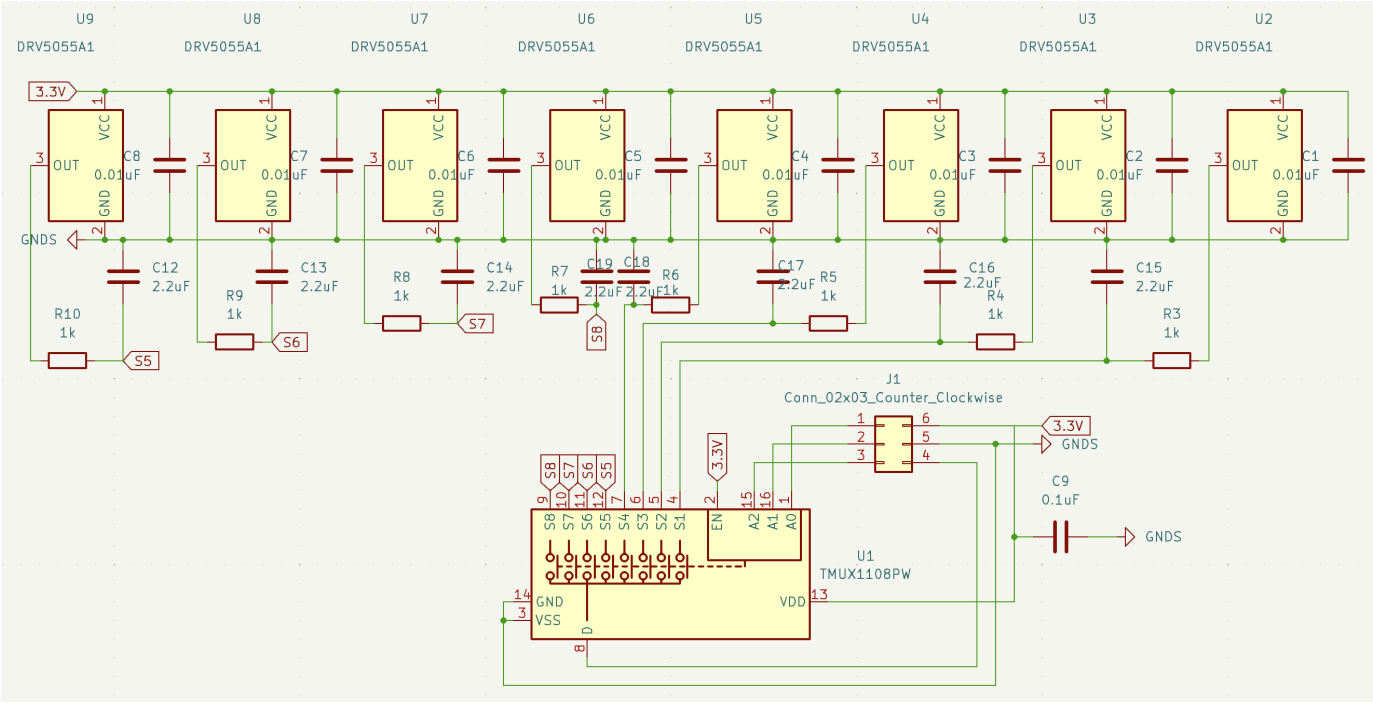


Fig. 4. The sensor board schematic used in the final design. In the PCB layout, the sensors are spaced two inches apart from each other.

heights on different squares, we saw substantial discrepancies. We believe this was caused by slight variations in the angle and position of the sensor soldered onto the board. We were unable to precisely fix the sensor positions while soldering them by hand, so manufacturing differences occurred despite our efforts. After a few failed solutions, we decided to fix this in software. We measured the minimum and maximum voltage values for each piece type on each square, and created a separate set of bins for each individual square. The firmware required for this is discussed in Section III-D.

The first major modification to the sensor boards was the inclusion of filtering. It occurred to the team that the long traces in the board would be prone to picking up 60Hz environmental noise. First-order low-pass filters with a corner frequency of 10Hz were deemed sufficient to attenuate this noise. The sensor output was assumed to be nearly DC, though AC components were expected to be introduced into the signal by the high-speed switching of the mux selection as the system scanned the board. To filter the output of an individual sensor, we selected 10kΩ resistors and 2.2μF capacitors. This RC filter results in a corner frequency of 7.2Hz, which met our specification for a satisfactory filter. Higher order filtering was also introduced in other sections to better attenuate the high frequency noise coupled from digital signals within the system.

A problem was found in the footprints of the the Hall effect sensors: the pins were in reverse order relative to the schematic symbol. The problem was resolved by flipping the orientation of the sensors and bending the sensor in such a way that the flat face of the sensor head was close to flush with the ceiling of the enclosure. The voltage bins corresponding to black and white had to be swapped to compensate for the sensors' orientation change. Brief testing concluded that aside from this, the change would result in negligible differences from the anticipated orientation. A few sensor boards are shown installed in the acrylic board top in Fig. 5.

2) LED Arrays: The original design for the LED array was to wire together a matrix of 64 addressable LEDs arranged in rows. Power was distributed from a central I/O board to each row in parallel. This wiring arrangement allows the power bearing wires to remain relatively short, so the current load of each LED only compounds within the row instead of through the whole array. Because the LEDs selected used a serial protocol for programming, clock and data had to be routed through each of the rows, alternating direction each time. We decided to use RGB LEDs to give us flexibility to use color as a means of conveying different meanings without text or audio.

In choosing which RGB LED chip to use, we faced a complex decision. Serial addressable RGB LEDs can be generally considered within two categories: synchronous 4-pin (APA102-like) LEDs [15] and asynchronous 3-pin (WS2812-like) LEDs [16]. The synchronous type requires four pins: clock, data, power, and ground. The asynchronous type is able to function without a clock, but this introduces some downsides: the asynchronous protocol relies on very precise timing to signal the data. We found that the 3-pin protocol was unsuitable for our application: the timing sensitivity would necessitate that we disable interrupts while signaling the LEDs, but our product needed to be able to tolerate interruptions in LED programming to achieve the low latency we were

TABLE I
FINAL SENSOR HEIGHT AND VOLTAGE BINS USED FOR THE PIECE BASES. THE HEIGHT REPRESENTS WHERE THE MAGNET IS POSITIONED FROM THE BOTTOM OF THE PIECE.

Piece Type	Height (mm)	White Voltage Range (V)	Black Voltage Range (V)
Pawn	1	0.040 - 0.200	3.28 - 3.40
Rook	3.2	0.560 - 0.960	2.40 - 2.60
Knight	4.5	1.00 - 1.12	2.20 - 2.40
Bishop	6	1.08 - 1.24	2.04 - 2.14
Queen	8	1.20 - 1.32	2.00 - 2.08
King	11	1.44 - 1.56	1.76 - 1.80

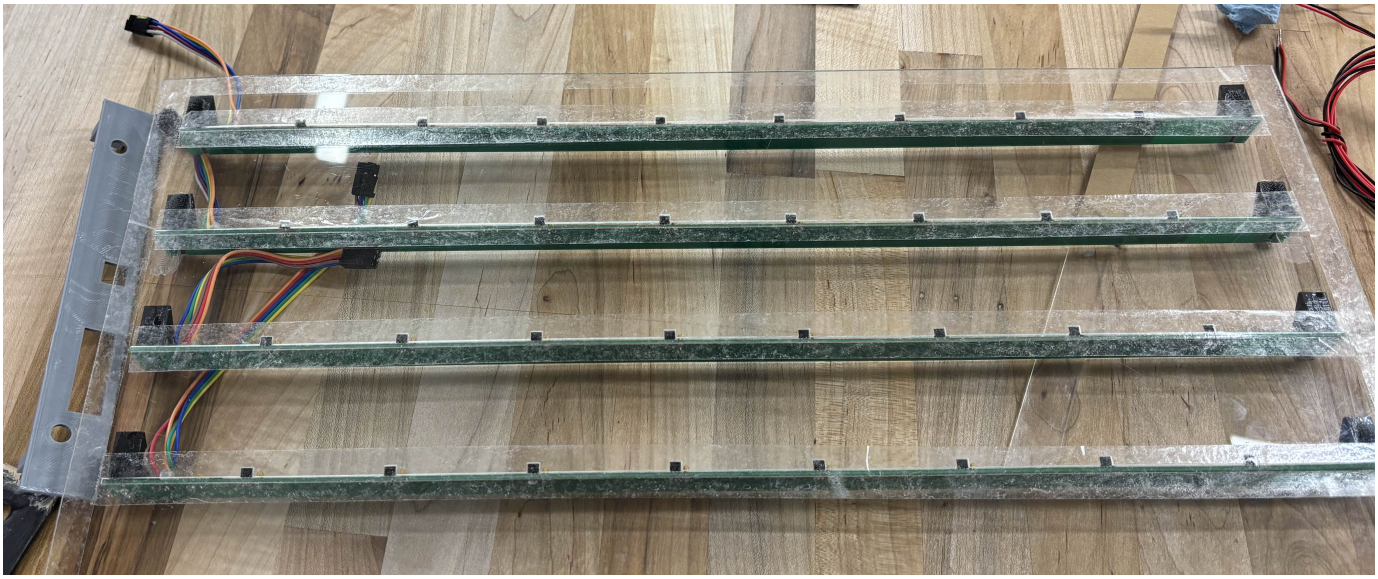


Fig. 5. Sensor boards installed into the acrylic surface sheet.

targeting. After reviewing the synchronous LED products available on the market, we selected the AdaFruit DotStar LED strips [17], which use the SK9822 (APA102-compatible) chip [18]. The synchronous protocol they use is compatible with SPI, so we were able to take advantage of the MSPM0's built-in SPI hardware modules when using them. Because these LEDs use 5V logic and our MSPM0 uses 3.3V, we had to add a level shifter (ISO6731) [19] in the middle.

One modification to this design was to add 18 additional LEDs. These LEDs were added underneath the piece silhouettes at the edge of the board, to allow software to highlight a specific piece type during gameplay. These silhouettes help in setting up the initial board state and in reverting to prior board states. For instance, if a piece has been captured, reverting that move will light the indicator to display which piece type was captured and should be placed in the empty square.

Another modification to the LED array was the inclusion of an RC circuit at the beginning of both the clock and data lines. Initial testing showed substantial flickering when the LEDs were connected to the level shifter. This issue was made more challenging to debug by the fact that connecting an oscilloscope probe made the oscillation disappear. From this, we decided to add an equivalent circuit model of the oscilloscope probe (an RC circuit in parallel) to the clock and data lines. We only had a limited range of values in our laboratory, so we scaled the resistance and capacitance to maintain the same time constant, ending up with 220kΩ and 1nF. This successfully resolved the flickering issue.

The completed LED array can be seen installed into the board in Fig. 10.

3) I/O PCB: The I/O board is a hub for power, sensor, and control distribution between the chessboard and clock box. It has 6-pin connectors for the sensor boards, 4-pin connectors for the LEDs, the second-stage 8:1 mux, a barrel socket, a 5V-3.3V DC-DC converter, a level shifter, and a DE15 connector. The 6-pin connector was chosen to provide simple header pins for debugging when the harness was not attached. The 4-pin connector was only necessary for the one LED strip that started the clock and data chain. However, using 4-pin connectors for all of the strips simplified parts logistics, and a bag of JST connector pairs was already available in the lab. The barrel socket was sized to match the 2.10 mm standard of the power supply and to handle the maximum wattage of 50W (10A at 5V) [20]. Our system has both 5V and 3.3V loads. Given that

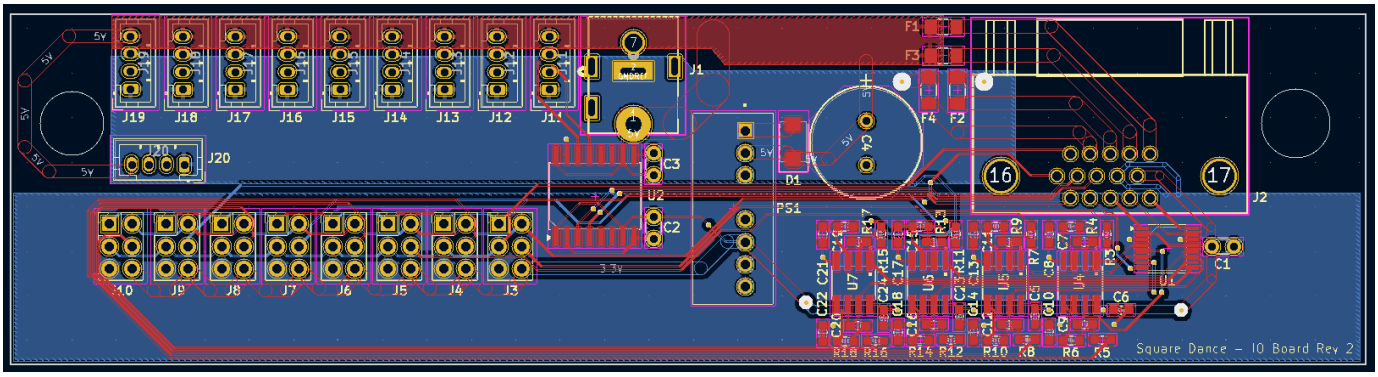


Fig. 6. Revised I/O board layout.

power flow is more efficient at higher voltages, we decided it was best to start with a 5V supply and step down to 3.3V with a DC-DC converter as needed. The DC-DC also isolated the 5V and 3.3V power domains and mitigated some of our noise concerns. The exact model used was the RECOM R3K-053.3S/H3 [21] since it could supply above the required 3W, had isolation, and was the cheapest option from one major distributor. The level shifter was necessary to translate the MSPM0's 3.3V control signals for the 5V LEDs. Since the two power domains needed to be isolated, we used an isolated level shifter [19]. The DE15 was an acceptable choice since it had sufficient pins for the number of connections we needed between the chessboard and clock box: 13 unique nodes. It is a relatively simple connector with easily accessed, direct pinouts. Moreover, the shielding built into the cable helped to address some environmental noise concerns. The additional two pins were necessary, though, because each pin is only rated for 3A [22]. Our initial design used a Raspberry Pi 5 in place of the eventual Pi 4; the Pi 5 which draws up to 5A from its 5V power supply [23], so we used the extra two pins as extra rails (in parallel) for the 5V power system to support the needed current.

The I/O board had one major modification to its design prior to the manufacturing of the first PCB set. Like the sensor board, filters were added to the design to address noise concerns. Unlike the sensor boards, though, we decided to use active filters: eight second-order Sallen-Key filters were used, one for each row. Having second-order active filters allowed us to effectively remove high-frequency noise while keeping step response times tight. The expected input to these filters was a rapidly switching DC value; to keep latency as low as possible and accommodate these inputs, we optimized the step response of the filter to converge very quickly without oscillating. These filters were designed as Bessel filters to produce an ideal step response for the effectively DC values being read from the sensors. Using the Texas Instruments analog filter designer, resistor values of 1.5kΩ and 2.7kΩ and capacitor values of 15nF and 10nF were found to produce the desired step response: a convergence to 0.5% within 150us. These choices resulted in a passband gain of 0dB and a corner frequency of 5kHz, which were within our design parameters. The operational amplifier chosen for the filters was the TLV9052 [24], which was relatively cheap and had sufficient bandwidth for our application.

We ended up making a second revision of the I/O board to address footprint issues. Numerous devices whose symbols and footprints were imported from UltraLibrarian had mismatched pin numbering. In the barrel jack, the polarity was reversed and pads needed to be widened. The DE15 connector's middle row of pins needed to be shifted one position to the left so that the "arrow" formed by the pins pointed the other direction. The DC-DC enable pin needed to be disconnected from V_{CC} instead of shorted. Lastly, the pin 1 silkscreen dots for the operational amplifiers needed to be shifted down to make them visible. Although footprint issues also existed for the CHESSBOARD's other PCBs, the I/O board's issues were significant enough to justify the manufacture of a formally revised board. Without the revision, the chances of introducing noise or bridging due to faulty soldering would have increased greatly. The revised version of the I/O board can be seen in Fig. 6.

4) Clock Box PCBs: As a cost saving measure, two different PCBs were arranged on the same PCB and cut apart later. One board was a header board to interface with the microcontroller; the other was the display board, housing the LCD timing display [25] and the buttons through which the users can interact with the device. These two boards can be seen conjoined in the KiCad layout (Fig. 7).

The MSPM0 header board takes the power and data from the main DB-15 connector and routes it to the appropriate pins on the MSPM0 development board, which plugs in from jacks on its underside. The header board also contains the necessary filtering and isolation blocks for the sensor and UART signals. To filter out the interference from the nearby high-frequency digital traces, we added a 4th-order Bessel filter composed of two cascaded Sallen-Key blocks. The Bessel filter response was selected for similar reasons as it was for the I/O board: the Bessel step response was preferred because the expected input signal was entirely composed of discontinuous transitions between stationary values. To achieve the desired response,

resistor values of $4.7\text{k}\Omega$ and $6.2\text{k}\Omega$ and capacitor values of 1nF and 1.1nF were used for the first filter; for the second, resistor values of $2.4\text{k}\Omega$ and $3.9\text{k}\Omega$ and capacitor values of 1nF and 2.7nF were used. These values produced a corner frequency of 20kHz and a passband gain of 0dB. Besides this filtering, the board featured various header pins for interfacing with the MSPM0 and routing signals and power to the display board, which is also stored within the clock box. For noise shielding, we grounded the casing of the DE15 connector on the MSPM0 header board. To avoid a ground loop, we left the other side disconnected.

The display board is relatively simple in terms of functionality. The desired display state is sent from the MSPM0 to the device's LCD driver chip [26] over an SPI-like communication protocol. This signal is routed to the display board via the header pins installed onto the board, which also supply power and ground. When the load pin on the LCD drivers is pulled high, the LCD driver outputs the correct AC waveforms to produce the display state it received from the MSP. Alongside the displays, this board houses 5 of the buttons on the clock box. These buttons are connected to the MSPM0 through a Schmitt trigger buffer [27] which debounces the signal. The time constant of the debouncing circuit was tuned by adjusting the values of the capacitors soldered in parallel with the buttons and the values of the pulldown resistors. Given the complexity of the code required for button debouncing, we decided that a hardware solution was more practical and simpler. The debouncing circuit is nearly identical to that described in *Debounce a Switch* [12], but with the polarity of the output flipped: the resistor pulls the output down, rather than up.

Compared to the I/O board, there were fewer issues in the design and assembly of these boards. The display board needed to be expanded from what was initially designed to account for the footprint of the LCD drivers. The header board also expanded unexpectedly: an improper board cutting job resulted in excess PCB dielectric on the header board compared to what was designed. By the time we noticed this, the board had already been completely soldered. After some consideration, we decided that a redesign of the clock box enclosure to account for the slightly larger dimensions was the most efficient option.

Unfortunately, like with the I/O board, the clock box PCBs encountered footprint problems. Though we were able to overcome these issues without a second revision, the solutions were not straightforward. The most significant footprint issue to deal with for this board was that of the LCD itself. This board fit the pins horizontally, but not vertically: the vertical spacing was 7mm too small. We were able to work around this by bending the pins of the LCD inward. This increased the vertical distance of the clock box's face, necessitating another alteration to the enclosure's design.

These footprint issues notwithstanding, we encountered relatively fewer problems when designing the clock box PCBs. There was an unforeseen issue with the Raspberry Pi, though. Initially, we had set up the system so that the Raspberry Pi would be powered from the same 5V power supply as the rest of the system; we justified this after reading online that a Raspberry Pi could be powered either from the USB port or the header pins. Since we had an easy way to bring 5V jumpers across, we planned to power the Pi over the headers. Unfortunately, testing showed that this somehow was not the case for the Pi 5. We briefly considered cutting another hole in the clock box enclosure for a separate power cable (supporting USB-PD negotiation) or trying to emulate the USB-PD protocol in a customized USB-C cable, but decided that downgrading to a Pi 4 was the most straightforward option. This design modification was not of consequence, though: the Pi 4 was still sufficiently powerful to run Stockfish with enough resources to produce powerful insights.

5) Power Budget Design: In order to help with sizing traces and finding a proper power supply, the team assembled a power budget sheet using the maximum power draw of each component. This was performed separately for the two power domains. The 3.3V power budget (Tab. II) was used in the selection of the DC-DC converter to ensure that it would be able to supply sufficient power. For the 5V system (Tab. III), we designed conservatively by assuming that the DC-DC converter was as inefficient as a three-terminal regulator. The team made sure to add margins either for discrepancies from datasheets or modifications to the power budget. For the entire system, we selected the Adafruit 658 power supply [28], which is rated for 50W (a 25% margin above the expected maximum).

D. Software Design Details

Software was designed in two fundamental modules: the firmware deployed on the MSPM0 and the software deployed on the Raspberry Pi. The MSPM0 firmware was based on FreeRTOS and was responsible for managing the project hardware, including the sensors, buttons, clock display, and LEDs. The Raspberry Pi ran the latest (as of November 2024) version of Raspberry Pi OS, which is a port of Debian Linux. Many common and useful utilities were thus provided "out of the box" either as default software or accessible packages, including Python, a character device mapping for UART, and Stockfish. The Python modules `chess` and `serial` were ultimately incorporated into the overall Raspberry Pi script; each of the modules was a stable, functional option for offloading software tasks and reducing the scope of new software which needed to be written.

1) MSPM0 Firmware: The entirety of the firmware was written in C. When deciding between real-time operating systems, we had two options: a lightweight RTOS written by John and Paul during ECE 4501 or the community-supported FreeRTOS [29]. Our team selected FreeRTOS as the base for the firmware because of its extensive testing and official support from Texas Instruments on the MSPM0. In a similar vein, we chose not to write our own peripheral drivers to the extent possible: instead of interacting directly with the memory-mapped peripheral registers on the MSPM0, we used TI's DriverLib library [8], which provides a convenient

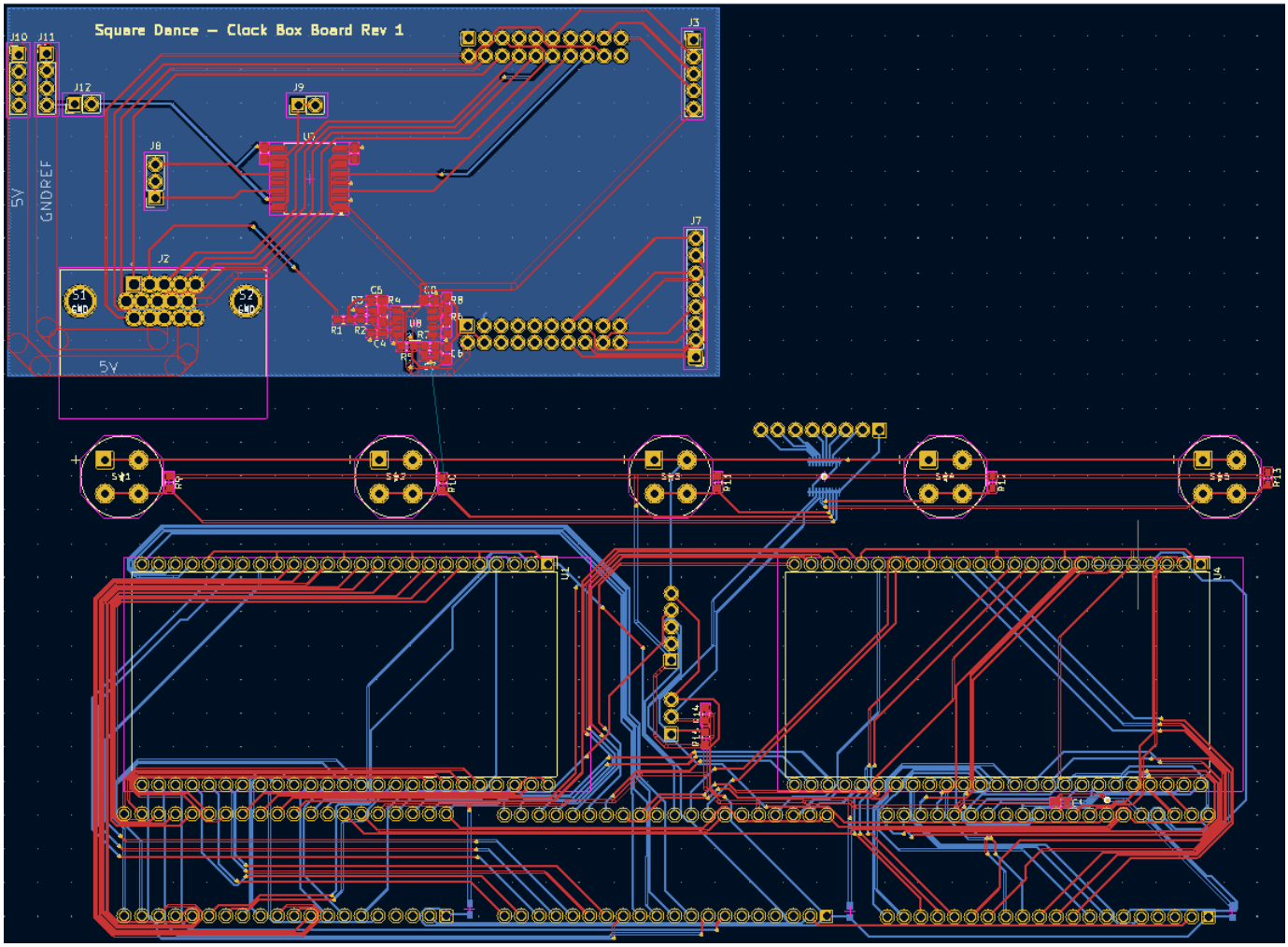


Fig. 7. Revised clock box board layout.

TABLE II
POWER BUDGET FOR 3.3V SYSTEM.

Part	Max Current (mA)	Quantity	Net Current (mA)	Power (mW)
Hall effect	10	64	640	2112
Isolated Level Shifter	5.8	2	11.6	38.28
Muxes	$1 \cdot 10^{-5}$	9	$9 \cdot 10^{-5}$	$2.97 \cdot 10^{-4}$
7-Segment AC Generator	0.06	3	0.18	0.594
Op-Amps	50	5	250	825
MSPM0	7.68	1	7.68	25.3
Total			909.46	3001.22

TABLE III
POWER BUDGET FOR 5V SYSTEM.

Part	Max Current (mA)	Quantity	Net Current (mA)	Power (W)
LEDs	60	82	4920	24.6
Isolated Level Shifter	5.8	1	5.8	0.029
3.3V DC-DC	909.46	1	909.46	4.5473
RPi4	1800	1	1800	9
Total			7635.26	38.1763

interface for configuring and operating the peripherals. We still needed to write drivers to generate data in the format expected by the hardware connected to the peripherals, but being able to use well-tested and well-documented peripheral drivers undoubtedly saved us substantial development and debugging time.

Deadlock, race conditions, and state corruption were primary concerns when designing the firmware. To prevent these issues, the firmware was designed in a modular fashion, and each peripheral was only accessed from one thread. The SPI hardware for the LEDs, for example, is only accessed from a dedicated LED thread. Communication between the threads was accomplished through synchronized FIFO queues that FreeRTOS provided. This project's philosophy toward inter-thread communication followed the Go language proverb: "Do not communicate by sharing memory; instead, share memory by communicating" [30]. Instead of using global state, we chose to use a system of message passing to ensure consistent state with no race conditions or deadlocks. Following this philosophy required using more memory but resulted in much greater flexibility and maintainability: for example, all the hardware driver logic could be logically from the main state machine logic. Splitting the firmware into these modules was also helpful from a testing perspective, allowing us to independently test each hardware driver with minimal changes to the rest of the code base. The block diagram for the firmware is shown in Fig. 8 and depicts the interactions between the various threads and peripherals within the MSPM0. The arrows between separate threads indicate communication through a synchronized FIFO.

2) Internal 32-Bit Protocol: Communication between the microcontroller and the Raspberry Pi occurs over UART. To facilitate this, we designed a custom 32-bit protocol for encoding chess moves in both directions. The protocol has four modes: RPi→MSP normal, RPi→MSP undo, MSP→RPi normal, and MSP→RPi calibration. We designed the protocol to be able to encode any valid chess move and any valid chessboard action, including hints, undo moves, multi-piece moves (*e.g.*, castling, *en passant*), and piece promotion. The calibration also allowed us to quickly and efficiently collect calibration data for creating per-square bins to correct for differences in sensor height under the board. The calibration data, once received on the Raspberry Pi, was saved to a cloud-synchronized file for analysis. The protocol is described in further detail in Appendix A.

3) Stockfish Wrapper: The Stockfish wrapper code deployed on the Raspberry Pi (`sf_wrapper.py`, plus a helper module `wrapper_util.py`) ran in an infinite loop in a similar fashion as the embedded code deployed on the MSPM0. Once setup is complete, the wrapper script effectively runs as a finite state machine that generates predictable responses to the limited changes of turn switches, undo button presses, restart button presses, and hint button presses.

Fig. 9 demonstrates the finite state machine behavior in `sf_wrapper.py` and its helper module. At the beginning of every loop iteration, the wrapper blocks until it receives a custom UART packet from the MSPM0 (represented by the `ReadPacket` state). When a packet is received, the wrapper decodes this and detects special conditions based on specific packet encodings. A zero packet indicates a sentinel move with a special meaning that the player has finished requesting undos. If the packet encodes a hint request from the MSPM0, meaning a player has pressed the hint button, the wrapper sends the MSPM0 the most recently computed best (*i.e.*, most skilled) move from Stockfish, then continues a new iteration. If the packet encodes an undo request, meaning a player has pressed the undo button, the wrapper rewinds the internal chessboard data structure by one move, sends the MSPM0 back the undone move, and continues a new iteration. If the packet encodes a restart, meaning a player has pressed the start/restart button, the wrapper clears its internal chessboard data structure and confirms this board state to the MSPM0 (including by sending the initial legal moves), then continues a new iteration.[2] Otherwise, the wrapper decodes the packet as a regular move and plays the move using Stockfish. When the new turn is played, Stockfish gives a new best move and list of legal moves.

Based on the results from the new turn play, the wrapper indicates endgame conditions or otherwise sends the new list of legal moves to the MSPM0 and transitions back to `ReadPacket`. If the wrapper detects through `chess` module routines a checkmate has occurred, the wrapper sends a special checkmate packet to the MSPM0; similarly, if the wrapper detects a stalemate, it sends a special stalemate packet. In either case, the wrapper transitions back to listening for a packet from the MSPM0. This is, however, a no-op until a start/restart packet is received and the wrapper begins recording a new game.

The wrapper runs its main event loop as an infinite loop. The only disruptions that can occur are due to loss of system power or from interruption due to catastrophic errors on the system.

4) Raspberry Pi Administration: As the Raspberry Pi ran a fully functional Linux instance, it required careful system administration. The Pi ran a Secure Shell (SSH) server to allow the team to remotely develop on the system; the server allowed only public key authentication, however, rather than the simpler password authentication. The configuration was sufficient for prototyping with the caveat that, if the CHESSBOARD was mass-produced, such networking capabilities would likely be disabled and the wrapper script would effectively run as firmware. Configuration tasks for the Raspberry Pi included creating a Python virtual environment for the wrapper script to run within; installing the `chess` and `pyserial` (aliased in code as `serial`) packages within the Python virtual environment; installing Stockfish; ensuring Git was installed if not already present; and cloning the Git repository containing the code to be run on the Raspberry Pi. Like the SSH server,

[2]The diagram hides these extra steps within the `Reset` state for readability reasons. Sending the MSPM0 the legal moves effectively means that the `Reset` state transitions to the `SendLegal` state, then to the `ReadPacket` state.

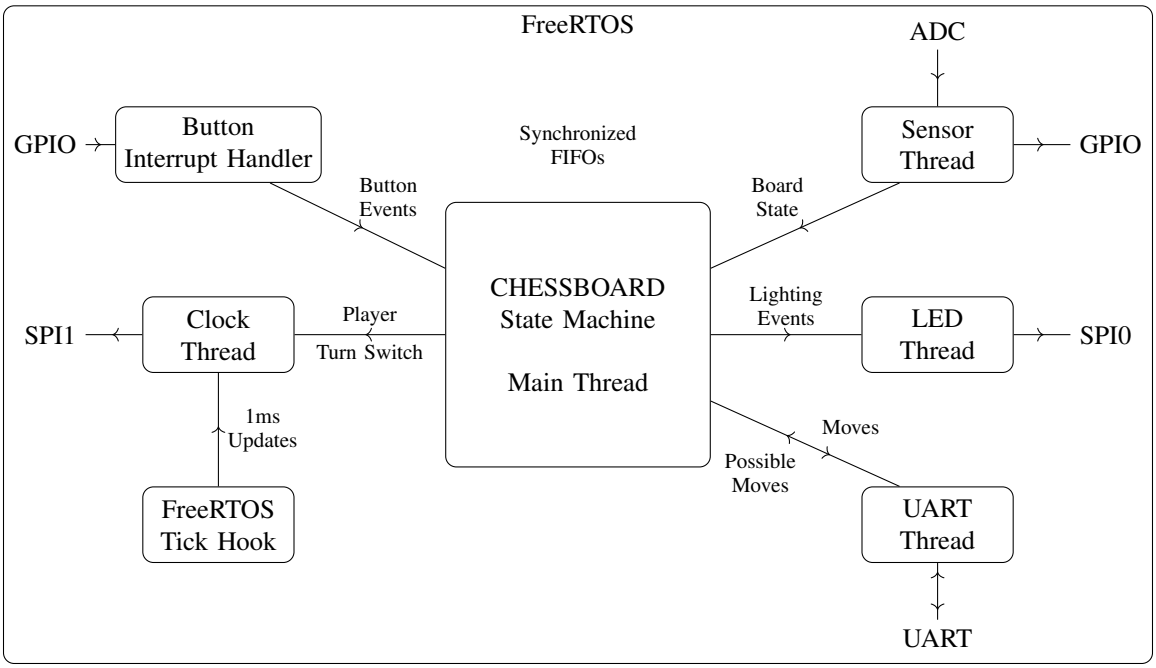


Fig. 8. MSP Firmware Block Diagram.

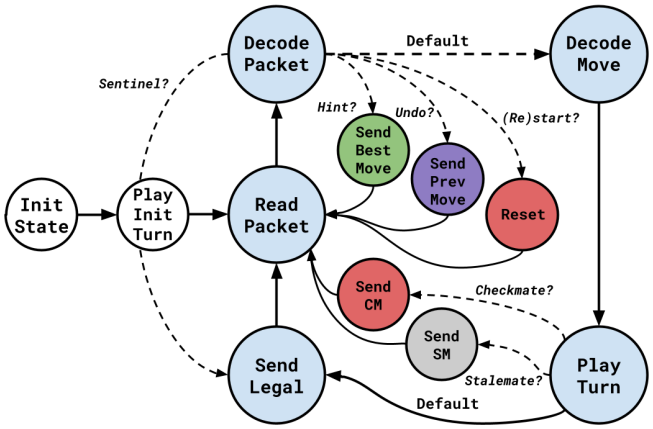


Fig. 9. Finite state machine (FSM) diagram representation of the Stockfish wrapper script, which was the main code run on the Raspberry Pi.

Git was used solely for development and prototyping and would be omitted from the Raspberry Pi's software if the CHESSBOARD was mass-produced. Lastly, for file retrieval purposes, a special data directory was created, and a `systemd` rule was written to automatically mount any detected USB drives and copy game data over accordingly.

While using Git version control and relying on commonly available packages increased system reliability, it became apparent during development that the Raspberry Pi was only as reliable as the microSD card it booted Linux from. Unlike systems like laptop and desktop computers that use larger hard drives, the smaller Raspberry Pi boots from a microSD card. If the microSD card broke (most commonly due to accidental bending), a new microSD card had to be flashed with a Raspberry Pi OS image and set up according to the procedure described above. Recovery was doable in a development setting but could pose significant risks in an environment where the CHESSBOARD is a mass-produced consumer product. In a real product, protections like better casing and padding around the microSD card, which was otherwise exposed when using the available Raspberry Pi case, would significantly improve the microSD card's resistance to bending and breaking. This would reduce the likelihood that consumers would experience a sudden malfunction in their product due to an unexpected failure in a delicate microSD card.

E. Manufacturing Design Details

1) Chessboard: The chessboard houses the sensor boards, LED array, cell dividers, and I/O board. The computational devices (Pi and MSPM0) were kept external to the chessboard to keep the profile as slim as possible and reminiscent of a traditional chessboard (see Fig. 1). Although we were willing to compromise where necessary, the team wanted the CHESSBOARD to resemble a normal chessboard in order to make the process of switching between the CHESSBOARD and a normal chessboard easy. The base and walls were intended to be made of a solid block of wood milled out with a cavity, but time constraints prevented this. On the floor of the enclosure the LED array was hot glued in position. The cell dividers were made from sheets of composite wood, then cut with a laser cutter to have cutouts that interlocked with the other dividers and supported the placement of the other components in the enclosure. The use of cell dividers helped isolate light produced by LEDs to a distinct cell—we wanted to avoid

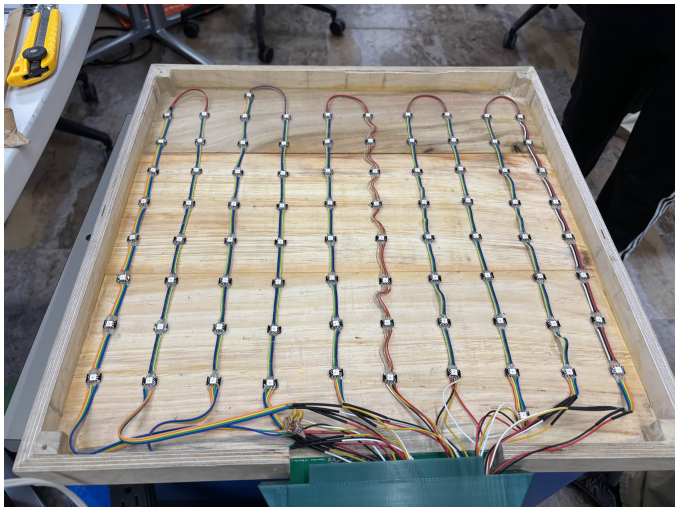


Fig. 10. LED strips glued to the bottom of the chessboard enclosure.

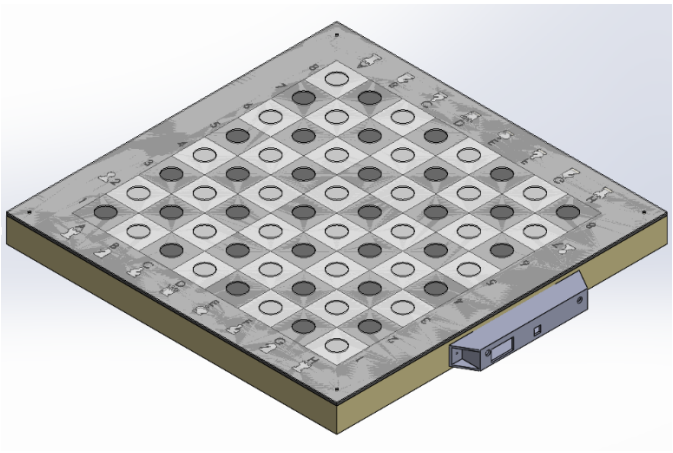


Fig. 11. Revised CAD model of CHESSBOARD physical structure with simplified geometry, expanded I/O shield, and added piece silhouettes.

confusion for the user when we tried to light up a particular tile.

The last major component of the design was the lid. The lid is a composite of three layers that were originally intended to be acrylic, polycarbonate, and acrylic, in that order. Polycarbonate was the only offering from McMaster Carr for translucent, colored plastic with dimension $24\text{in} \times 24\text{in}$ (determined by the smallest acceptable FIDE tile length of 2in multiplied out by the eight rows plus extra for a border). We needed a colored middle layer so we could laser etch in any white detailing (*e.g.*tiles, borders, coordinates). The composite would be held together with glue. Functionally, the top layer would be for piece alignment with the sensor: it consisted of hole cutouts to help center a chess piece in a square above the corresponding sensor. The middle layer was the layer on which the pieces rested and in which aesthetic details were etched. The lowest layer would help align the sensor boards using specifically shaped cutouts (see Fig. 5). The composite panel would be secured to the base with bolts.

After discussing the manufacturing of the enclosure with staff from UVA makerspaces, the base design was changed to suit traditional woodworking methods (see Fig. 11). It was the opinion of the staff that the time investment to get trained on the CNC would have been too great for the timeline of this project and that the milling process would have been relatively wasteful (about half of the material would have been lost). Instead, scrap wood planks were planed and glued together for the base. The base was surrounded by composite wood panels with miter joints. Square wooden posts were glued to the inner corner of the base to provide something for the top panel (lid) screws to bite into. Also, we found that polycarbonate could not be laser cut without producing toxic gases, so the team pivoted to applying color to another layer of clear acrylic. When a dyeing method did not work, the team instead used a film of window tint. Applying the window tint was difficult since the adhesive left many bubbles on the surface. We compromised by using no adhesive for the tint or the acrylic and using thin strips of double sided tape instead, which limited the air bubbles to less visually important areas of the board. We added silhouettes of each piece on the border of the board to help guide the player with initial game setup and to give us a method to signal what piece to place at the identified tile when reverting to a prior board state. After manufacturing, the team was dissatisfied with the hatching density, which made it a bit difficult to tell what tile was white. To make the tiles appear whiter, we sanded the tile parallel to the edges of the square.

2) Clock Box Module: The clock box contains the clock box PCBs, the Raspberry Pi, and the MSPM0. The display was angled up at the user and the Pi and the MSPM0 were housed in a cavity at the bottom of the clock box. We chose to use 3D printing to make the enclosure since it gave us the flexibility to custom-fit the cavity to the devices so mounting would be "friction fit."

We held off on making the clock box enclosure until the end to give us the flexibility to adapt its layout in response to any unforeseen issues. Some examples of changes that occurred were expanding the cavity in response to board misalignment, moving port windows after swapping from the Pi 5 to the Pi 4, and placing button cutouts on the display face of the enclosure as well as on top of the enclosure to accommodate the final PCB layout. Additionally, the LCD screens were mounted at a greater height above the display PCB than anticipated, resulting in another adjustment to the enclosure model. Vents were added above the Pi to ensure it would have sufficient air flow to prevent thermal throttling. The revised model can be seen in Fig. 12.

Due to limited printer bed size, the enclosure was segmented into four separate prints (two for the top and two for the bottom) that were glued together. The top and bottom halves can be screwed together, but we often kept them unscrewed during the development phase for ease of access.

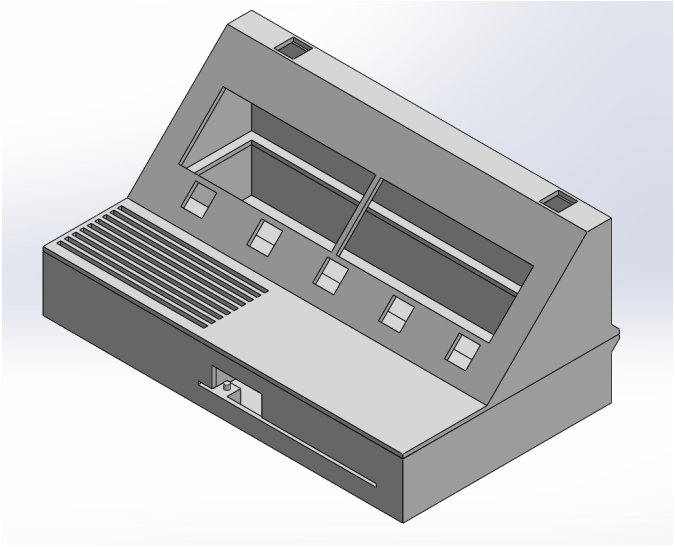


Fig. 12. Revised CAD model of clock box enclosure with button ports, ventilation grill, and horizontal split line.

Fig. 13. Initial chess piece design with a slot and plug (left) and final design with a base and body (right).

3) Chess Pieces: In our initial design, we planned to 3D print the chess pieces with a hollow cavity to fit the magnets. Chess piece models were taken from Thingiverse and modified to hold the magnets. The original project is the OpenSCAD Chess by TimEdwards [31]. Initially, a separate cylindrical plug would have been printed to secure the magnets inside the chess pieces. The height of the plug would vary depending on the type of chess piece to position the magnet at a certain distance from the Hall effect sensors. This setup allowed each chess piece to be associated with a specific magnetic field strength. However, after printing a few test pieces, we encountered issues when inserting the plugs and attempting to align the bottoms of the plugs with the chess pieces. In the redesign, the chess pieces were split into base and body segments. The bases held the magnets at a certain distance from the Hall effect sensors. The upper sections of the chess pieces were bonded to the bases using acrylic glue. The initial and final designs are visualized in Fig. 13.

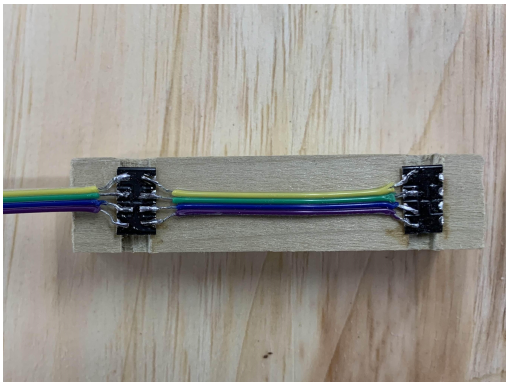


Fig. 14. Manufacturing tool to ensure consistent distances between LEDs.

4) LED Array and Soldering: A total of ten LED strips were manufactured, each consisting of eight LEDs (or nine LEDs to light up the pawn silhouettes). For each strip, the LEDs were connected by wire groups of a 2-inch length. To ensure that the LEDs would be evenly spaced and centered with each chessboard tile, a wooden jig was designed to hold the LEDs during soldering as shown in Fig. 14. While soldering the LEDs strips, we performed connectivity tests to ensure that the rails were electrically connected. Next, the LED strips were soldered together along with their connectors. When trying to install the LED network into the chessboard enclosure for testing, though, there were several instances where the wires became disconnected. To cushion the solder joints, we applied hot glue across the wire and LED contacts. Hot glue was also used to secure the LEDs to the bottom of the chessboard as shown in Fig. 10.

5) Computer Systems: The MSPM0 (formally, the Texas Instruments MSPM0G3507-LP) and the Raspberry Pi (formally, the Raspberry Pi 4B) were directly sourced from their respective vendors. Manufacturing for these components was controlled through external processes.

F. Testing and Verification

1) PCBs: As a general testing process for all PCBs, we first checked for continuity between ground and V_{CC} to detect errant shorts. We also checked for continuity between small SMD component leads with other nodes on the board sharing the net to find poor solder joints. In most cases through-hole components were reliably soldered and did not need special attention prior to supplying power and testing system behavior. If the continuity checks produced correct results, the PCBs were powered with an external bench power supply with current limits close to the theoretical maximum power draw. If the voltage sagged, indicating that the supply's current limit had been reached and a short probably existed, the PCB was sent back for visual inspection under a microscope to try to find the issue. Disconnecting devices where reasonable was a strategy used to try to isolate the source of the problem. If the boards cleared all basic inspection, they would be then examined based on their respective requirements and expectations as outlined in the following sections.

2) Sensing System: To determine if our sensing method with the Hall effect sensors worked, we performed preliminary testing with the three different sensors: A1, A3, and A4 (see Section III-C). To vary the magnetic field strength that would be detected by the Hall effect sensors, we varied the distance between the sensors and the magnets. With the Hall effect sensors connected to 3.3V and ground, we laid the sensors face-up on a tabletop. The magnets were displaced a certain distance from the sensors by placing a number of printer paper sheets between the face of the sensors and bottom of the magnets. Since printer paper has a standard thickness, we assumed it would work well as a variable spacer. Starting with 16 sheets of paper, we measured the height of the paper medium with calipers. The voltage output of the sensors was measured with an oscilloscope for both cases where the magnet poles were faced down. Next, we added another 16 sheets of paper and repeated this process until we reached 128 sheets.

Based on the preliminary testing, we decided to use the A1 sensors as they provided the best resolution of the output voltages. Using these results, bins for the output voltages and distances (between magnet and sensor) for each of the chess piece types (*e.g.*, pawn) were initially determined.

The next stage of sensor testing involved the 3D printed bases that held the magnets. Test tiles were also modeled to have a 32mm diameter divot to center the chess pieces at a standard height of 1/16 inches to replicate the physical design of the chessboard enclosure. The sensors were placed against the bottom of the 3D printed test tiles at their center and the 3D printed bases rested inside the divot. Each base and its associated bin were measured by moving the base around within the divot and recording minimum and maximum output voltages produced. The magnet was then flipped inside the base and the same procedure was performed. This process was repeated for each of the 3D printed bases and six different sensors.

First, we tested the original design for the chess pieces with the plug-and-cavity method described in Section III-C. When inserting the plugs into the chess pieces, we found it difficult to align the bottoms of the plugs with the bottoms of the chess pieces. This led us to redesign the chess pieces with the base-and-body design. In the later rounds of testing, we also realized that the sensors were not perfectly centered when taped to the test tiles. The test tiles were then redesigned to have a bottom slot for the sensor, centered in the correct location (see Fig. 15).

There were some complications in the displacement tests: the sensors were not perfectly centered and were slightly mobile during testing as tape was used to secure the sensors to the test tile. The test tiles were redesigned to perfectly align the sensors with the center of the divots. Bottom and top layers sandwiched the sensors to minimize air distance between the top of the sensors and bottom of the magnets.

After manufacturing the sensor PCB boards and the chess-board enclosure, we retested the chess piece bins with the prototype model. In this testing procedure, we interfaced to

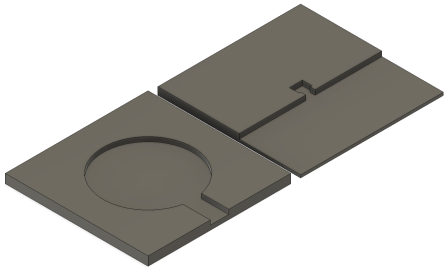


Fig. 15. Model of test tiles with the bottom layer (right) to fit the sensors and top layer (left) with 32 mm divot.

the analog multiplexer on the I/O board through the MSPM0 breakout board. First, we confirmed column and row selection by inputting 3.3 V logic levels to the appropriate pins. To determine which sensors were selected, we observed whether the data pin output varied from the idle state voltage ($\approx$1.64 V) when a magnet was placed on the above tile.

When we first tested the sensor boards, we realized that the Hall effect sensor footprints were incorrect as 3.3 V was traced to the ground pin of the sensors and ground was traced to the supply pin. To resolve this issue without purchasing new PCBs, we flipped the sensors so that they were face-down. We also had to bend the pins to align the sensors with the PCB edge so that they were flushed against the middle acrylic layer. The resulting voltage bins were different from when we tested the bases using the test tile, which was partially a result of the middle acrylic layer being slightly lifted from the bottom layer near the center of the board. To resolve this, we applied double-sided tape between the bottom acrylic layer, window tint, and middle acrylic layer. After this redesign, we performed another round of sensor testing and adjusted the heights of the chess piece bases accordingly.

When integrating the software system, a testing procedure was developed to verify the voltage bins. The LED under a specific tile would light up to indicate which sensor was selected and actively reading. Two buttons on the clock box were set up to allow us to select adjacent sensors. When a chess piece base was placed on the active tile, the silhouette of the detected chess piece (*e.g.*, white pawn) lit up. We discovered that many of the sensors detected different chess pieces when measuring the same base. This variability must have been from errors introduced by bending the Hall effect sensor pins. A calibration procedure was then created to measure the full output range produced by each of the 64 sensors when testing each type of base. Three more clock box buttons were utilized to start and end calibration and to select which type of base was being measured.

3) LED Arrays: Two tests were performed regarding the LEDs: one to verify the operation of an individual row and a second to test the entire LED array.

When the LED data and clock lines were connected directly to the MSPM0, the single-row test worked perfectly, cycling through a wide range of colors and brightnesses as expected. However, after it was connected through the level shifter, we

found that the data was being corrupted, resulting in rapid flickering. It was determined during this test that it would be necessary to add a parallel input RC network for the LED data and clock lines. After the input network was added, the row's data was observed to output the expected result at all 8 LEDs. After verifying functionality on a single row, the array was constructed.

After the LED array was assembled and installed, this code was flashed onto the MSPM0 to observe the LED output. Compared to the response of the individual row, there was far more noise introduced into the full array, with an expanded version of the row code resulting in LED values seeming to randomly change. The root of this problem was found to be poor soldering between some of the data and clock lines throughout the system. Resoldering some of the most obviously problematic joints and applying hot glue to provide flex resistance resulted in a system that acted exactly as designed.

4) UART Communication: Loopback tests were conducted for both the MSPM0 and the Raspberry Pi to verify as a "ground truth" that UART for both devices worked. At first, the MSPM0 and the Raspberry Pi were respectively isolated, had their UART TX connected to their UART RX, and were programmed with code that sent test messages to UART TX and immediately read from UART RX. This verified that, independently of the integrity of the communication medium between them, TX could correctly encode serial messages according to the expected UART protocol and RX could correctly decode serial messages by the same standard. UART data sizes were fixed at 8 bits for both devices, and a standard 1-bit start bit and 1-bit stop bit were used with no parity bit included with the word. Not only were test words verified for correctness in software, but oscilloscope captures were taken for both devices to verify electrically that their UART peripherals behaved as expected.

For the Raspberry Pi, a reference to file-mapped UART (at `/dev/serial0` on the Raspberry Pi 4B) was first opened. In the initial version of the UART interaction code which used C, this directly invoked the Linux `open`, `write`, and `read` system calls to operate on 4 bytes at a time. Loopback testing on the Raspberry Pi using this version of the UART interaction code (referred to during the design process as the "UART booster") first revealed that a kernel patch unexpectedly altered direct memory access (DMA) and inhibited UART communication. After the UART interaction code was replaced with invocations of the `pyserial` module and its routines (specifically `serial.Serial` to create the file reference and `serial.Serial.read`, `serial.Serial.write`, and `serial.Serial.read_until` for byte-level I/O), the loopback tests were repeated. It was at this point that the kernel version had been reverted in an effort to fix the DMA issues. Using `pyserial` quickly and conclusively verified that, in fact, loopback testing was successful on the Raspberry Pi. The change to `pyserial` was maintained even after the team concluded that the Pi's UART DMA issue was resolved since using the trusted `pyserial` library instead of the previous custom driver code saved further development and debugging time.

The MSPM0 firmware was all C-based, so loopback testing occurred in C. Unlike blocking Linux file operations, which introduced complexity that using `pyserial` was necessary to deal with, the MSPM0 UART device was simpler and more straightforward to test. Conducting a loopback test on the MSPM0 involved calling lower-level Texas Instruments driver functions that directly moved bytes to and from the memory-mapped UART device. First, the custom UART routines written for the project (which still used the TI drivers but wrapped repeated invocations of them to deal with 32-bit UART protocol words) were invoked for loopback testing. When these drivers busy-waited indefinitely for space to become available (for TX) and for FIFO entries to become available (for RX), loopback testing was done using simpler examples sourced directly from TI rather than from more complex derived code. Using the simpler TI code examples was unsuccessful, however. Faced with no other software-based explanations for the MSPM0's UART failures, the team repeated the test while attaching oscilloscope probes to the configured UART TX and RX, discovering that no part of the UART communication appeared to occur. Bolstering this conclusion was the fact that the logic level was never pulled high (*i.e.*, to 3.3V) during testing. Because no part of the UART initialization appeared to occur successfully, the team concluded that some underlying hardware defect caused the UART failure on the MSPM0. When the team repeated the loopback testing using the same code and oscilloscope setup but with the UART module configured to use different pins, the loopback testing succeeded. A quick fix was implemented in which the functional pins were physically shunted to the nonfunctional GPIO pins (which nonetheless remained connected to maintain compatibility with PCBs that had already been manufactured). While the team could not have anticipated the UART failures for either the Raspberry Pi or the MSPM0, the team's loopback testing approach allowed for the most accurate attribution of these issues and, therefore, for the best solutions (in the context of this project) to be implemented.

Once loopback testing was successful on both the Raspberry Pi 4B and the MSPM0, the devices were programmed with testing code that propagated a duplex communication (*i.e.*, two-way with reception and transmission on both ends) over UART using the custom 32-bit protocol. With some initial hurdles in verifying `pyserial` usage overcome, breakpoints on the MSPM0 allowed halting the microcontroller and viewing variables in watch windows to verify that words sent by the Raspberry Pi were correctly received on the MSPM0. Duplex testing was done interactively in that specific words and byte sequences were sent from the Raspberry Pi using an interactive Python shell with `pyserial`, received on the MSPM0 using code that triggered breakpoints, viewed in a debugger connected to the MSPM0 to verify proper receipt, transmitted back on the MSPM0 UART TX to the Raspberry Pi, and read back on the Raspberry Pi. Thus, data integrity was verified at all points of the communication (first transmission,

first reception, second transmission, and second reception), and the overall accuracy of the transmission was verified by successfully comparing the words transmitted from the Raspberry Pi to the MSPM0 with the words subsequently received by the Raspberry Pi from the MSPM0.

When we connected the entire system together, we ran into one last set of problems with the UART protocol. The Raspberry Pi and the MSP have different startup timings: the MSP comes up within fractions of a second, but the RPi takes 20-30 seconds. The word size supported by the UART hardware is a single byte; each protocol packet consists of four. Depending on the specific timing of the boot process, the first word received by the UART on the RPi might be halfway through a packet. Similarly, the first word received by the MSP might be erroneous noise generated during the boot process, rather than the start of a packet. To remedy this, we introduced a handshake protocol (similar to TCP's three-way handshake) that runs during the start-up, to synchronize the packet frames on both sides. Upon starting up, the MSP sends `0x00000008` ("SYN") every 100ms until it receives a response. Because the first (least significant) byte is unique in this frame, it can be used to align the framing on the RPi side. When the RPi program starts, it waits for a `0x08` byte and then begins processing packets. It sends back `0x00000001` ("SYNACK") for each SYN it receives, which is similarly used to align the MSP's framing. To acknowledge this reception, the MSP sends back `0xFFFFFFFF`. This triggers the start of the chess move list generation.

5) Stockfish Wrapper: The Stockfish wrapper was best tested with all hardware subsystems working to replicate a live chess game. Before all subsystems were assembled into a complete prototype, the wrapper's helper module was extensively tested using a set of white-box unit tests that covered various code paths and checked correctness.

First, the `parse_button_event` function which decodes the least significant two bits of every packet to check for button event indicators was tested. The intended behavior was verified in each case: for lowest two bits 01_2[3], `parse_button_event` flagged the packet as indicating a start/restart button press; for lowest two bits 10_2, `parse_button_event` flagged the packet as indicating a hint button press; for lowest two bits 11_2, `parse_button_event` flagged the packet as indicating an undo button press; and in all other cases (*i.e.*, lowest two bits 00_2), `parse_button_event` indicated that the packet represented a normal move.

Next, the `decode_packet` function was tested to ensure it properly constructed an intended move from a packet. The custom UART protocol encodes the source and destination squares within the most significant 16 bits of the word. Unit tests confirmed that, for an arbitrary setting of bits to indicate squares, these squares were transcribed as intended according to rank and file (*i.e.*, $\{000, 001, ..., 111\} \rightarrow \{a, b, ..., h\}$ file translation occurred and $\{000, 001, ..., 111\} \rightarrow \{1, 2, ..., 8\}$ rank translation occurred). Accompanying unit tests confirmed that, as intended, the encoding of any bits except for those in the specifically designated source and destination square fields did not influence the decoded source and destination squares. These moves were ultimately decoded and stored as `chess.Move` objects and, in unit tests, their representation as move strings was quickly verified for successful decoding.

The Stockfish wrapper's business logic was enforced in the functions `encode_packet`, `encode_undo`, and `encode_mtype`. Unit tests for `encode_packet` were first written to reverse the process of `decode_packet` and encode a custom packet from a `chess.Move` object. White-box tests ensured that when `chess.Move` objects were encoded as packets, these new packets matched the original packets the `chess.Move` objects were decoded from. Next, `encode_mtype` was verified through a series of unit tests to return valid encodings whenever move objects indicated that castling conditions, check conditions, or castle conditions were present, or otherwise to return an encoding indicating that the move had a "normal" type. Since `encode_packet` calls `encode_mtype` as part of its logic, verifying `encode_mtype` proved not only its own correctness, but helped prove the correctness of its parent function `encode_packet`. Lastly, `encode_undo` was tested to not only encode the source and destination squares for the most recent move, but also to indicate which type of piece had been moved. A series of unit tests proved that `encode_undo` correctly marked a packet with the source and destination squares; that the function queried the chessboard correctly to get the type of piece that was moved; and that the encoded packet reflected that a special move (castling, capture, promotion) occurred if applicable. The helper utilities were comprehensively verified using this testing framework; thus, there was an assurance of software correctness as the prototype was assembled.

The overall product (including the main event loop) is currently being tested as part of whole-system testing, and is showing promising results thus far. The team is confident that further testing will show where bugs exist as applicable and that the demonstrated final product will exhibit software behavior that meets the design requirements.

One of the more interesting features we wanted to support is automatic transcription. From the chess wrapper side, this is straightforward enough: we write a text file to a special directory for each game. It is slightly more complicated to automatically copy this to a USB drive. We initially planned to use a `udev` rule to trigger a script that would mount the drive, copy the files, and unmount the drive. This turned out to be more difficult than expected: `udev` does not permit mounting or network access inside the rule handlers. Instead, we made a `systemd` one-shot service template that runs a shell script to perform the mount-copy-unmount procedure. We were able to configure the `udev` rule to trigger our custom service with the correct device as its argument. Because this service was spawned in normal userspace (rather than in the

[3]Binary values are written as literals in base two with a subscript $_2$ appended for clarity, *e.g.*, $1 = 01_2$.

udev sandbox) it was able to perform privileged operations like mounting.

IV. Constraints

A. Parts and Resource Availability

The MSPM0 had a very small amount of memory: only 32 KiB of RAM. This was far too small to store the data structures necessary for a chess engine, so we chose to separate tasks. The MSPM0 manages the hardware and the board state machine, while the Raspberry Pi coprocessor manages the more complex state machine of the game itself. Based on the power calculations for the board performed in Section III-C5, we ordered a 50W AC power supply, which was sufficient for our immediate needs and should provide enough headroom to support a variety of future extensions.

The enclosure had minimal manufacturing constraints. One goal was that we wanted to follow FIDE's standard of tile sizes. Otherwise, the only limits were our own capability to manufacture the design, our skill, and our access to resources. Factors which limited our manufacturing capacity included restricted access to machining tools and special training requirements. This was the cause for decisions like making the enclosure with traditional woodworking instead of CNC milling (in this case, for instance, the makerspace staff believed it would take too long to get trained). Another manufacturing constraint was the 400mm $\times$ 500mm limit set by the chosen PCB supplier (JLCPCB). The team followed the design flow that a PCB was designed with all of the necessary components. The enclosures were then built around the dimensions of the PCB and other components like the MSPM0 and Raspberry Pi.

B. Software Tools

FreeRTOS provided the threading, synchronization, and multitasking primitives used in this project. The FreeRTOS implementation of queues was particularly widely used throughout the firmware.

Raspberry Pi OS was installed on the Raspberry Pi to provide a powerful general-purpose operating system that supported Stockfish, file persistence for recording game data, and I/O processing for UART, at minimum. While the lead team member in charge of the Raspberry Pi (Paul) began the project with Linux system administration experience, this project's requirements dealt with lower-level Linux operations (character devices for UART) than he had encountered before. The team, especially Paul, needed to learn how to bridge applications traditionally held to be in the domain of embedded C programming (*e.g.*, UART and other byte-level serial protocols) with comprehensive operating systems and their features (Linux with its I/O system). Furthermore, the team had to gain practice with mounting removable drives and automating software to run when such drives were connected. In addition to the mounting and unmounting, the team ended up having to learn the format of `systemd` services and their limitations to successfully automate the Raspberry Pi's intended behaviors (running the wrapper on boot and running the transcription copy script upon detecting a removable drive). While not without technical hurdles along the way, the project was an ultimately successful exercise in extending Linux skills to leverage the Linux file system in more powerful ways than the team had done before.

The team used Python and Bash as needed to administer the Raspberry Pi and run the desired code. Python was selected for its breadth of modules and for its ease of implementing features in code which would have required a substantially greater amount of lower-level C code. While Python's internal complexity does typically make the language slower than C, this was a trade-off the team willingly made since the tolerable latency on the Raspberry Pi—targeted at one-quarter of a second for move computation based on human reaction time estimates—was lenient enough to accommodate any overhead that Python introduced. In other words, using Python never seriously prompted concerns since other features of the system (chiefly the physical realities of chess) contributed far more to overall latency. Like with Linux, the lead team member for Python programming (also Paul) began the project with significant experience; in this area, the project required extending existing skills rather than learning new skills altogether. Nonetheless, Paul and the team had to learn the details of Python's `chess` and `pyserial` modules (and how to test code which relied on them) to effectively meet the project demands.

For the PCB designs, we used KiCAD, which is a free cross-platform application with much better library support than alternatives like Ultiboard. Since some team members were already familiar with KiCAD from prior projects, this helped the other team members learn the software more easily. Schematic and PCB files were uploaded to a shared drive for collaboration.

The modeling software used for the design of the chessboard and clock box enclosures was SolidWorks. Fusion 360 was used to model the chess pieces. We used two different 3D CAD software tools was because the two team members responsible for the physical design of the project were familiar with different programs. Since the team members worked on separate design parts of the project, there were no issues in using different modeling software. Ultimaker Cura was used as the slicing software for the 3D prints since we were using the Ultimaker S3 printers available in the lab. Our choices of modeling software were necessarily contingent on the prior experience of the team members and the resources the team had access to. Modifying these constraints (team expertise and available resources) would likely modify the specific design software used.

C. Prototype Cost Constraints

For this project, our team was given a $500 budget. Initial estimates suggested that this project would cost $470. However, this excluded the costs of items that the group already owned personally (most significantly the Raspberry Pi 4B 8GB, which cost $80). Additional salvaged and personally owned component including buttons, passive electrical compo-

nents, 3D printing material, and an SD card had an estimated total cost of $32. In all, our actual initial budget would be approximately $582, with $30 left over to supply changes for any unforeseen issues.

Over the course of development, several adjustments needed to be made that resulted in budget increases. Such adjustments included purchasing an official power supply for testing purposes, creating a second revision of the I/O board, and purchasing surplus multiplexers and Hall effect sensors. These increased the spent budget to $520 ($20 over the provided budget), or $635 after accounting for the provided or personally owned materials. Though we were able to cover cost overruns for our prototype using personal funds, these excess costs would need to be considered and handled differently if the CHESSBOARD was manufactured at scale.

If we were to manufacture this prototype in its current state as a production version, the costs would decrease compared to the development costs as testing, revision, and surplus components would be removed from the production budget. Ignoring the taxes and shipping associated with our purchases, the cost of a production version of this product would cost approximately $346.05. This includes estimates for products that the team did not order but used from personal or lab inventory, like the Raspberry Pi 4B and 3D printing PLA. Further discussion about market-ready production can be found in Section IX, and the tables in Appendix B shows the final budget as discussed in this section.

D. Manufacturability

The design of the CHESSBOARD prototype has many aspects which could be changed to reduce cost of components and simplify manufacturing. Many choices were made to make the system modular and allowed for separate tasks to run in parallel and be tested independently. For example, jumper pins allowed for the team to have the flexibility of probing pins for testing while providing an interface to connect subsystems together when assembled as a full product.

The first aspect we would change for manufacturing is the sensor boards. We believe the sensor boards should be oriented horizontally, rather than vertically, in future versions of the product. This would have a slight drawback of making a very visible, opaque strip down each tile, but would allow for a reduced number of components and would help with sensor consistency. Horizontal sensor boards could be combined into a larger board with I/O board components integrated into it. In the PCB layout, we would just need to specify cutout regions in a grid fashion to allow light to wrap around the PCB and show through to the user. A horizontal board would also eliminate a large number of wires and connectors. Instead of cutting and soldering wires to LEDs, we could solder the LEDs directly to the underside of the sensor board and have a reflective coating on the cell dividers reflect the light up to the user. There would be no need to include connectors from I/O board to the sensors or LED array since those connections could all be internalized to the new PCB design.

Additionally, a new horizontal board would be a much better basis for an SMD-only design. From the standpoint of a product brought to market, the team believes it would be cheaper and faster to solder a board with exclusively SMD components with a stencil and solder paste. A reflow oven could practically solder every component simultaneously. Moreover, an SMD sensor would eliminate the process of bending leads to get the sensor head as close to the ceiling of the enclosure, reducing manufacturing time and improving consistency.

Improvements to the chessboard enclosure would be minor. The advice to use traditional woodworking as opposed to CNC milling was probably better for cost and manufacturing time. We did observe variability in the floor height in our prototype, but that was largely a byproduct of a nonstandard assembly process. A machine-driven drilling process for screws rather than glue would likely reduce that variability. The more significant change would be in the top panel. The tint proved problematic to use, so avoiding it completely in a consumer product would be best. Finding a supplier that makes translucent black acrylic would solve that issue. We still believe using laser etching for white detailing is better than independently coloring or machining tiles. When made from a single piece, there is no need to join together many parts, so there would be less irregularity in surface finish. Instead of an adhesive, using rivets periodically at the intersection of tiles would apply compression to the acrylic panels in a fashion that is readily performed by a robot.

Anything that was 3D printed could be manufactured at scale with injection molding techniques. To guarantee the precision of the magnet depth the magnet cavity would be mechanically milled.

A market-ready product would have the MSPM0G3507 chip installed directly to the breakout board without the rest of the development board, reducing package size and cost. Instead of placing jumpers between standalone header pins and MSPM0 launchpad header pins, traces on a new version of the breakout board could route peripherals directly to the MSPM0G3507 chip's pins for its various hardware modules. This also means the breakout board could reasonably be connected directly to the Raspberry Pi header pins using traces, eliminating wires and again reducing package size. This could allow us to build a smaller clock box that needs less plastic to form the housing.

The overall dimension of the chessboard and associated components could be scaled down so that the maximum dimensions are reduced from $20\text{in} \times 20\text{in}$ to $16\text{in} \times 16\text{in}$. This would violate FIDE regulations for tile size, but it will allow us to reduce costs substantially. The price point would drop since less material would be needed for each board. The noise captured in long traces would also be slightly reduced since traces could be shortened.

Our last few changes from the prototype would be to fix any remaining PCB design issues (formalizing any flywired solutions), add a power switch, and change the D-Bus connector to a standard 15-pin connector instead of the high-density version. These are mostly quality-of-life changes that either

help simplify the PCB layout or provide user safety.

V. Societal Impact

In this project, the main group of stakeholders are the users: chess students ranging anywhere from a complete beginner to an intermediate level player. This group is the most at risk for any technical issues that compromise the safety of the board such as a fault in the power supply. We hold a responsibility as designers to appropriately and safely create a product that can withstand the wear and tear that a standard chessboard would be expected to tolerate. We operated with the mindset that our design process should account for this and that a thorough testing procedure would assist in guaranteeing a safe experience for our users. Although we expect adults to use our device, children are a significant subset of our expected user base. Designing an appropriate digital product for children required special attention to data collection and retention; since the CHESSBOARD employs computer systems to collect data about any chess games that are played, which may involve child players, the CHESSBOARD was designed in accordance with the IEEE 2089-2021 standard. More information about IEEE 2089-2021 compliance is documented herein in Section VI.

Another important group to consider is the purchasers, more often than not being the parents of child users. If mass-produced, this product would likely still be prohibitively expensive for several intended purchase groups. This price point means that an average household likely could not purchase this product on a whim.

In lieu of use by individuals or their households, the device could appeal to school-affiliated and independent chess clubs who could integrate it into their activities and could be likelier to afford it based on their pooled resources. Although there were some concerns about equitable access with the assumption that only parents would be purchasing this, assuming this secondary group of potential purchasers lowers the barrier to access for the CHESSBOARD. The implications of this help to maintain the low barrier of entry for chess as a whole, further supporting the traditional game for anyone interested in playing it.

The technologies which constitute the CHESSBOARD have minimal potential for misuse and abuse. The computer systems involved can be arbitrarily reprogrammed assuming an individual has physical access to the device and sufficient time and resources; in fact, this is a risk for any consumer product that incorporates a computer system. Thus, the risks of physical access are, unfortunately, impossible to account for absent the incorporation of extreme and expensive security measures. However, the CHESSBOARD follows other, more reasonable measures when it can to be a secure product. Since the CHESSBOARD does not maintain an Internet connection and does not require an Internet connection to interface with an end-user's computer and dump transcriptions, fewer vectors exist for cyberattacks on the CHESSBOARD than exist for many other consumer technologies. Bugs and vulnerabilities likely exist within our code that we wrote to run on the CHESSBOARD; this is, unfortunately, not unusual, and it is also true that the libraries and operating systems we depend on may have vulnerabilities we could not have anticipated. Apart from reasonable testing for functionality, implementing additional security measures was not practical, nor would it have offset the greatest threat (physical access). Thus, the CHESSBOARD meets a similar ethical standard for protecting its intended application from unwanted interference that similar computer systems on the market meet.

VI. External Standards

In comparison to many other engineering projects, the CHESSBOARD has fewer standards it must uphold. Nevertheless, there are relevant standards worth discussing in relation to the product's design. These include standards relating to child safety [32], chessboard and piece regulations [33], LED brightness [34], and power safety [35].

A. IEEE 2089-2021: IEEE Standard for an Age Appropriate Digital Services Framework Based on the 5Rights Principles for Children

Since the CHESSBOARD was developed for a general audience that may include children, designing the computer systems and their code to interact ethically with child users was paramount. Specifically, the CHESSBOARD was designed in accordance with the IEEE 2089-2021 standard, which describes an "Age Appropriate Digital Services Framework" (AADSF) based on prior children's rights literature [32]. IEEE 2089-2021 was an appropriate choice since the burden of compliance was relatively low: a team or developer "is not required to engage an expert to conform with the standard" [32]. Rather, Square Dance needed to demonstrate that the outcomes, activities, and tasks in Clauses 7 through 15 of the standard had been achieved [32].

For the purposes of Clause 7 of the standard, the CHESSBOARD design features and data processing which interact with children are those features which support the transcription of moves that child players may make into chess notation. This notation is completely severed from a player's personally identifying characteristics like age or other status. Children's rights are thus inherently protected as users are not identified to begin with.

Pursuant to Clause 8, Square Dance has identified that the CHESSBOARD product and its associated services are appropriate for all children. Outside of standard variation in players' ability to understand the game of chess, there is no content within the product and services the CHESSBOARD provides that are appropriate for some children but not for others. Square Dance asserts that the product and its services are appropriate for all children and that, correspondingly, no engagement with an "age assurance" mechanism is needed on the part of the end user [32].

Pursuant to Clause 9, Square Dance has ensured that "children's rights are realized in the product or service" of the CHESSBOARD and that the terms of use which accompany the CHESSBOARD uphold children's rights [32]. For an

example of upholding children's rights, in accordance with Section 22580 of the California Business and Professions Code [36], the CHESSBOARD does not advertise a variety of drug- or weapons-related products to child users because the CHESSBOARD does not advertise any product to any user.

Pursuant to Clause 10, Square Dance does not subject child users of the CHESSBOARD to inappropriate commercial advertising techniques; completely separates player data from player identity and personally identifying characteristics; and collects the minimal amount of necessary data regarding chess moves themselves and the sequences in which they were made. Square Dance has not, and does not, put "commercial considerations" over children's rights [32].

Pursuant to Clause 11, Square Dance provides fair terms to child users that treat child users as equally capable as adult users. The actual CHESSBOARD products and its services uphold these terms, which are in the best interests of children [32]. The CHESSBOARD is a learning device that operates equally for child and adult users; since the CHESSBOARD serves to increase chess skills, which is an age-appropriate skill that engages with no adult concepts, the CHESSBOARD acts in the best interests of children.

Pursuant to Clause 12, if the CHESSBOARD is released for a consumer audience, Square Dance will publish CHESSBOARD terms of use to the extent necessary that are easily understood. Since the CHESSBOARD uses universal chess symbols and only involves culturally specific glyphs (*i.e.*, A-G for files and 1-8 for ranks) to mark different portions of the board, the device and its services are available to children of many different backgrounds. The underlying product of a chess game with anonymous transcription is age-appropriate; thus, the CHESSBOARD terms of use do "render an age appropriate service" [32].

Pursuant to Clause 13, Square Dance is "explicitly [adopting] and [implementing]" the AADSF standard, and the supply chain which provides the electronics, materials, and computer systems for the CHESSBOARD can be reasonably verified as offering "age appropriate digital services" [32]. Since the only digital services offered to the child end user are those which Square Dance create, there is practically no supply chain to verify. Correspondingly, the Square Dance member primarily responsible for the portions of the CHESSBOARD design which interact with children (Paul Karhnak) has read through the IEEE 2089-2021 standard and is making reasonable efforts to be trained in child compliant design. Lastly, Square Dance has consistently upheld children's interests in design and business decisions [32] and will continue to do so.

Pursuant to Clause 14, Square Dance is making a "best effort" attempt to incorporate the CHESSBOARD's impacts on children in the product and service design, noting various scenarios that child users may encounter, and prioritizing children's rights and development in the product [32]. The remaining risks in the product, which are hereby published, mainly encompass the inherent risks that unrestricted physical access to computer systems entail (see Section V for a detailed description). That is, the computer systems on the CHESSBOARD have the inherent risk of being reprogrammed with malicious software because anyone with enough time and access to such systems has unlimited control over them. Reasonable efforts have been taken to ensure that only with physical access is this the case, however; because network-based authentication on the system is based on a secure public key infrastructure rather than on passwords which are often easily cracked, any attacker would almost certainly need physical access. If the CHESSBOARD were developed into a mass-market device, network services would be disabled entirely; they were only enabled during prototyping for live testing and debugging purposes.

Lastly, pursuant to Clause 15, Square Dance has monitored and will, to the best of its ability, continue to monitor the CHESSBOARD prototyping and deployment, iterating wherever and whenever possible to best protect children's rights. If the device is developed into a mass-market consumer product and actually made available to children, Square Dance will publish a corresponding AAR. Furthermore, Square Dance by default shreds user data (completely overwriting it and deallocating the file system space associated with the data) once it is copied to an external flash drive. This respects the wishes of customers, including children, and their "right to be forgotten" [32].

B. Fédération Internationale des Échecs (FIDE) Handbook: General Rules and Technical Recommendations for Tournaments

In an ideal scenario, we would comply completely to FIDE's General Rules and Technical Recommendations for Tournaments [33]. However, many of our alterations from a standard chessboard, though vital to the functionality of our subsystems, violate the regulations outlined in this section of the FIDE Handbook. One such example is the divots that are to be incorporated into every square for our pieces. In our design, it is required that the piece be placed within a very specific proximity to the sensor to properly trigger the Hall effect sensor. These divots are all the same size, so the pieces we constructed all needed to be of the same base size. This violates regulation C.02.2.2, which states that piece's bases must be 40–50% of their heights. To comply with another element of the same regulation, dictating specific piece heights, this portion must be ignored.

The regulations also have several key features that we wished to include in our board design, including sufficient contrast between the two colors of pieces, a square size of at least 2in, and the previously mentioned piece heights.

With the understanding that these violations of chess tournament procedure were necessary to ensure the functionality of our product, we attempted to apply as many standards present as possible but accepted that violations were necessary. The intended goal of this product was to provide a tool for beginners moreso than an instrument for competitive play in tournaments. Adhering to as many standards as possible should produce a project that is ergonomic and comfortable for most users, so it is still important to consider these standards.

C. Other Standards to Consider

According to the Power Supply Safety Standards, Agencies, and Marks published by CUI, inc. [35], the device we are operating is Class III equipment, as we are supplying it with a 5V DC power supply. This is well below the 42V AC peak established by the Safety Extra Low Voltage (SELV), meaning hazardous voltages are unable to be generated by the system. These systems do not require extensive safety testing and are considered safe for operator access.

According to the IEEE Recommended Practices for Modulating Current in High-Brightness LEDs [34], flickering LEDs could pose a minor health risk to epileptic individuals when flickering at a rate below 90 Hz, especially considering that they would be expected to stare directly at the board for extended periods of time. Although the risks are minor considering that the view of the LED is obfuscated by the board's surface, it is still worth considering the recommendations outlined in the document to mitigate the adverse effects. Considering that 60 Hz is generally the limit for discernible flicker, it is important to be considerate when designing the system and guaranteeing that the power is sufficiently provided for each LED, while not being so bright as to be uncomfortable. Part of the way we addressed LED flickering was to ensure LEDs were connected in such a way that code written to consistently light up LEDs did not result in unexpected flickering: this is documented in Section III-F3. LED brightness is likely to be a subjective metric considering that we would be unable to measure the brightness across the board's surface. Though the majority of the document is dedicated to general LED lighting, it is still worth considering the implications of an LED that is expected to be stared at for an extended period of time.

VII. Intellectual Property Issues

In light of the more novel aspects of this design, the device's patentability is worth considering. The concept of a "smart chessboard" is already a widely commercially available product as well as a common electronics hobbyist project. Due to this lack of novelty, the idea of a smart chessboard itself likely has little or no patentability. In fact, a patent granted in 2022 features a smart chessboard that communicates with a computer by sensing chess pieces with RFID [37]; this patent is set to remain active until 2039. Since patents exist for similar "smart chessboards", this would have to be taken into consideration for a patent application for the CHESSBOARD.

In addition to the concept of a smart chessboard, several aspects of the design originate from other projects. This includes the idea of using Hall effect sensors as the means by which to determine the piece type. A patent from 1974 uses similar magnetic based sensors in order to determine the presence of a piece on a specific square [38]. Although our method of sensing determines the type of piece in comparison to just the presence, the preexisting magnetic board patent can be used to demonstrate that several aspects of this project are not novel. Another patent approved in 1978 describes an electronic chessboard which uses a push-button display to indicate the location of a set of chess pieces for any given board state [39]. This provides the ability to display a similar degree of information as we are able to with our LEDs, albeit with a different form factor and technology.

Beyond the sensing and communication approaches, the project uses the open source Stockfish chess engine and chess piece modeling files from Thingiverse. The chess piece models are modified versions of the OpenSCAD Chess project by Tim Edwards [31]; this OpenSCAD project is licensed under the Creative Commons BY-SA license, which allows for the copying or modification of the material to be commercially distributed so long as attribution is given to the original creator. If modifications are made (the original material is "remixed"), the remixed material must be released under the same license. Through this, we would be able to include our remix versions of the chess pieces in a patent application of our own. Similarly, Stockfish is open source software released under the GNU General Public License version 3 (GPLv3) [40], so using Stockfish does not impede the creation of a patent because our system is a derivative work, rather than simply a redistribution, of Stockfish.

Despite these features that are reused from previous work, the CHESSBOARD's overall approach is nonetheless novel. The CHESSBOARD's use case and specific application—a smart chessboard designed for learners with supplemental features like a hint button, undo button, and LED feedback—is unique. Although many of these aspects have appeared individually within other patents, our overall combination to support our goal can be argued to be novel. Indeed, some of the prior projects have aspects that overlap with each other, and they could still each be patented. Approaching the patent with this in mind while emphasizing the more unique aspects of our CHESSBOARD could give a potential patent application more merit.

VIII. Timeline

Over the course of the semester, there were several points of divergence from our initial estimate of the project timeline that became increasingly apparent. Chief among these were assembly and manufacturing processes that were significantly longer than expected. Nonetheless, we allowed ourselves a period of nearly three spare weeks at the end of the semester in our timeline, so we still completed this project on time before our major deadlines.

Our hardware development schedule changed dramatically over the course of the design process. During this process, we realized that there were several less critical elements such as the enclosure design. This part of the design required modification after having a better understanding of the PCB dimensions, so pushing the completion of this aspect of the project back was an appropriate action to take. Even after the PCBs were designed, some issues with assembly required adjustments to the enclosure, as discussed in Section III-C. Aside from this, the design largely aligned with our initial expectations (see Fig. 17).

Our software design schedule also changed. Although the preliminary hardware design phase was completed early in the

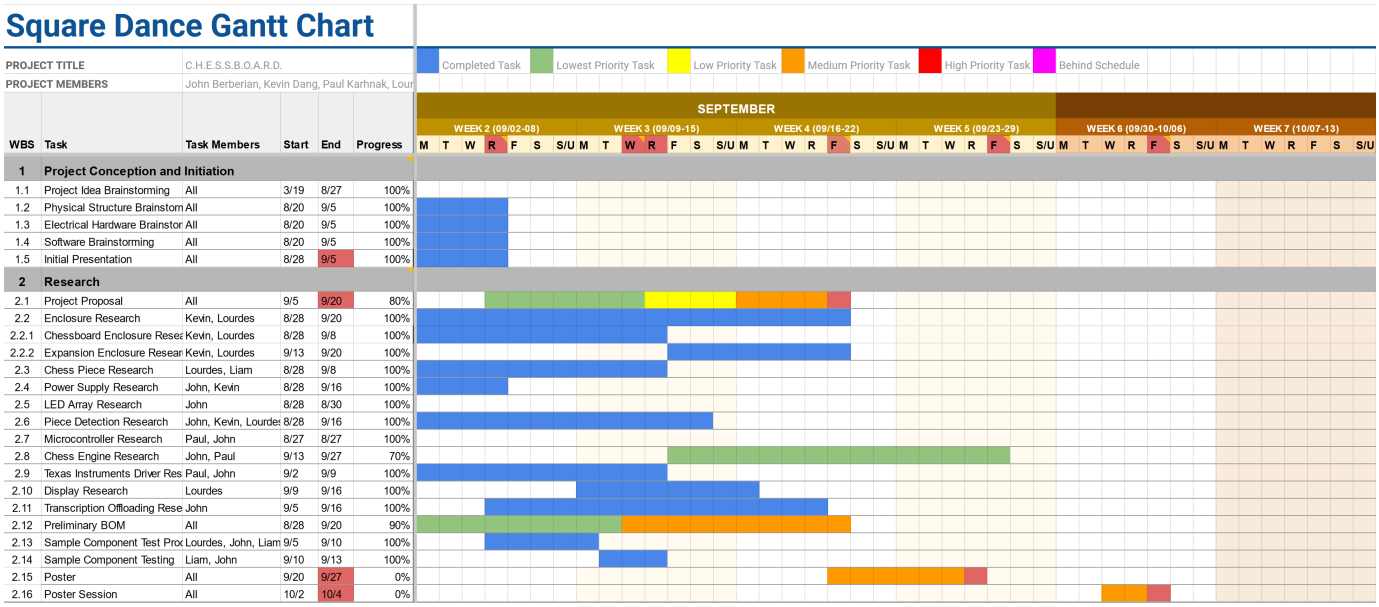


Fig. 16. Initial Gantt Chart of the Project Conception and Initiation and the Research Phases.

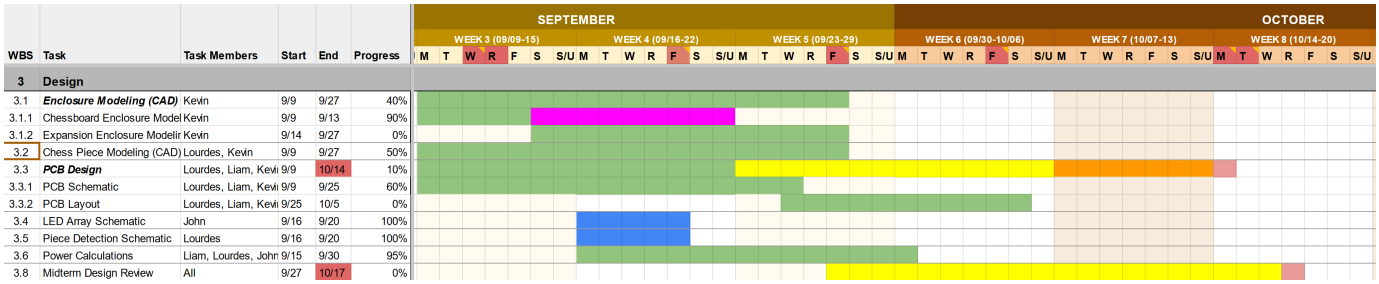


Fig. 17. Initial Gantt Chart of the Design Phase.

semester, later changes to the hardware systems necessitated restructuring and rewriting substantial sections of code. In addition, the delays caused by unexpected issues related to UART communication momentarily prevented the development of other sections that were dependent on it. Much of the software logic was completed early, but because the hardware was unavailable until later, we were unable to test large portions of the software until the last few weeks of the development period (see Fig. 21).

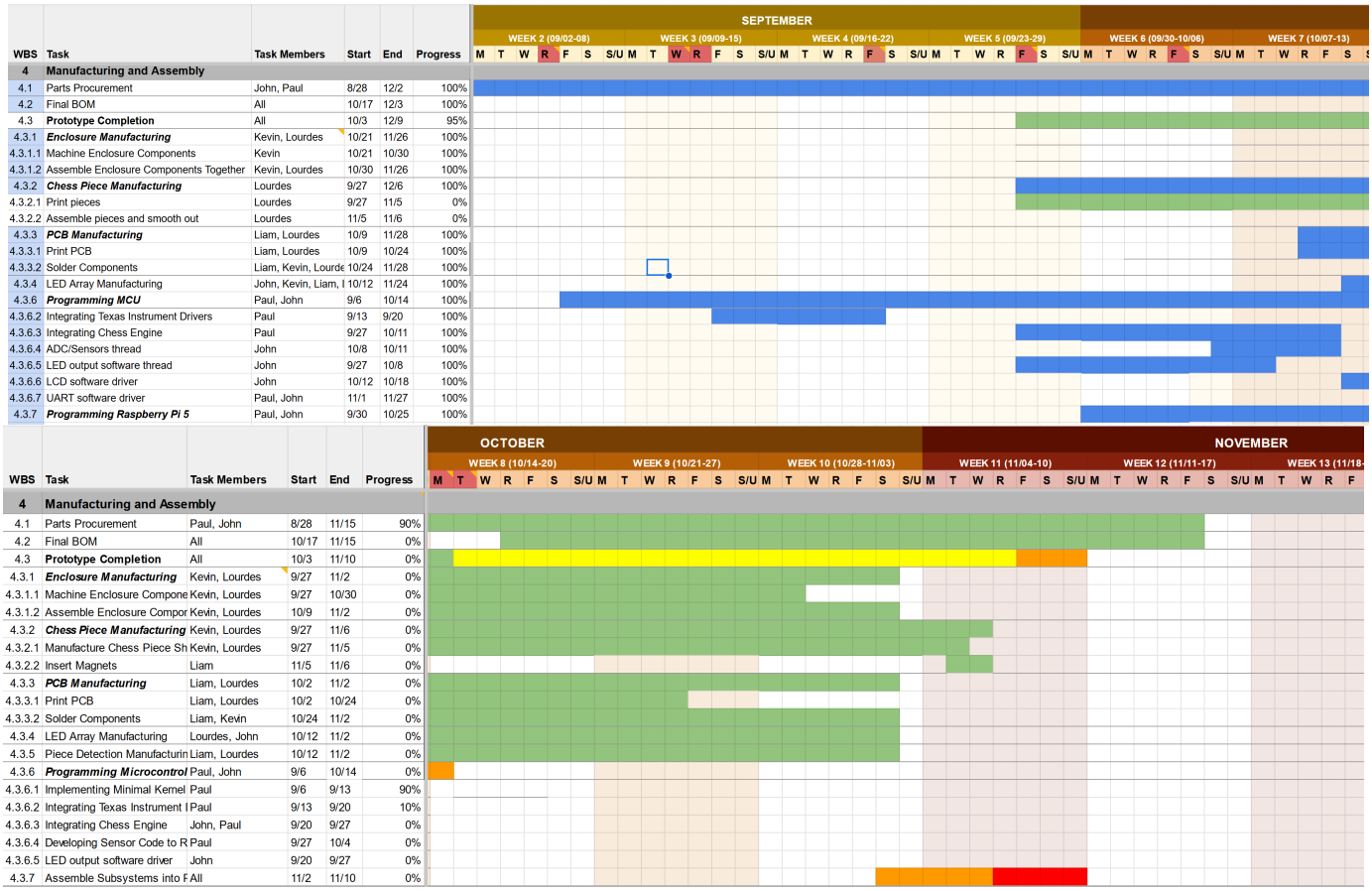
After the design phase, the assembly process experienced the most significant increase in time relative to our expectations. Initially, our expectations were to complete this over an approximately 4-week span in October and early November. However, this significantly underestimated the amount of soldering that needed to be conducted. The LED array and PCB soldering processes took until late November to be fully completed. As the team members doing the soldering were largely the same as those in charge of the enclosure manufacturing, the enclosure manufacturing was also impacted (see Fig. 20).

Hardware testing was ongoing during the entirety of this process, from simple beep testing and visual inspection which verified proper soldering to power or sensor testing that guaranteed we were adhering to our calculations. Other kinds of testing were also performed. This testing was largely consistent between our initial (Fig. 19) and final (Fig. 21) Gantt charts: we expected the testing process to occur simultaneously with the assembly process. After assembly was completed, the sole focus could be placed on guaranteeing hardware functionality. This last objective was a continuous task that was not complete until our final demonstration, so the task is not marked completed on our attached Gantt charts.

For reference, initial Gantt charts that reflected our expectations for the timeline of the project's research, design, manufacturing, and testing phases are given in Fig. 16, Fig. 17, Fig. 18, and Fig. 19, respectively. Our final Gantt charts which measure our actual project timeline as it proceeded are given in Fig. 20 and Fig. 21.

IX. Costs

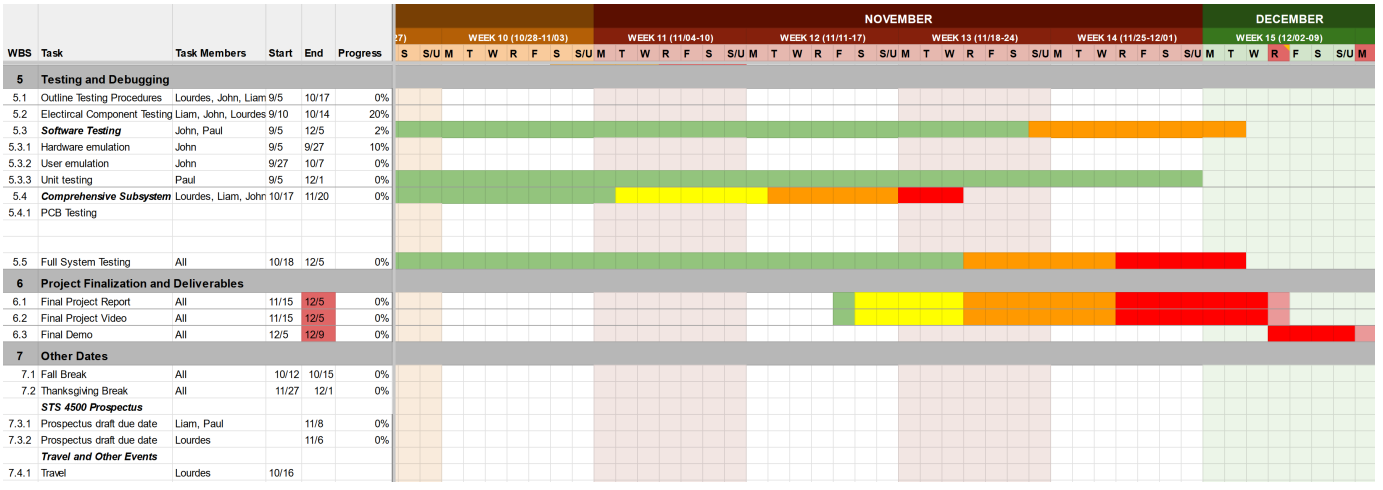
In this section we outline a potential mass production scenario for an order of 10,000 units. As mentioned in Section IV-D, there are many adjustments to the design that could be made to reduce costs. To summarize, we could



WBS	Task	Task Members	Start	End	Progress
4	**Manufacturing and Assembly**				
4.1	Parts Procurement	John, Paul	8/28	12/2	100%
4.2	Final BOM	All	10/17	12/3	100%
4.3	**Prototype Completion**	All	10/3	12/9	95%
4.3.1	***Enclosure Manufacturing***	Kevin, Lourdes	10/21	11/26	100%
4.3.1.1	Machine Enclosure Components	Kevin	10/21	10/30	100%
4.3.1.2	Assemble Enclosure Components Together	Kevin, Lourdes	10/30	11/26	100%
4.3.2	***Chess Piece Manufacturing***	Lourdes	9/27	12/6	100%
4.3.2.1	Print pieces	Lourdes	9/27	11/5	0%
4.3.2.2	Assemble pieces and smooth out	Lourdes	11/5	11/6	0%
4.3.3	***PCB Manufacturing***	Liam, Lourdes	10/9	11/28	100%
4.3.3.1	Print PCB	Liam, Lourdes	10/9	10/24	100%
4.3.3.2	Solder Components	Liam, Kevin, Lourde	10/24	11/28	100%
4.3.4	LED Array Manufacturing	John, Kevin, Liam, I	10/12	11/24	100%
4.3.6	***Programming MCU***	Paul, John	9/6	10/14	100%
4.3.6.2	Integrating Texas Instrument Drivers	Paul	9/13	9/20	100%
4.3.6.3	Integrating Chess Engine	Paul	9/27	10/11	100%
4.3.6.4	ADC/Sensors thread	John	10/8	10/11	100%
4.3.6.5	LED output software thread	John	9/27	10/8	100%
4.3.6.6	LCD software driver	John	10/12	10/18	100%
4.3.6.7	UART software driver	Paul, John	11/1	11/27	100%
4.3.7	***Programming Raspberry Pi 5***	Paul, John	9/30	10/25	100%

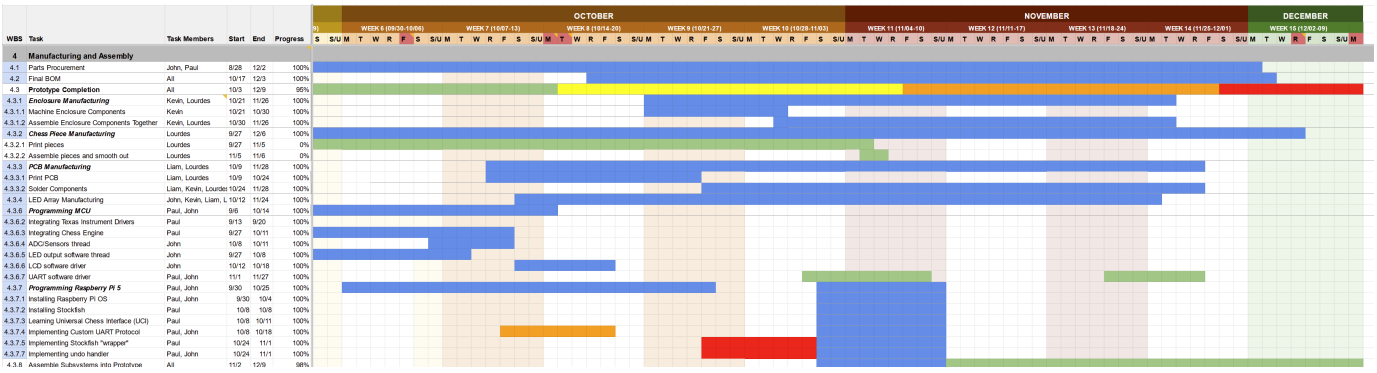
WBS	Task	Task Members	Start	End	Progress
4	**Manufacturing and Assembly**				
4.1	Parts Procurement	Paul, John	8/28	11/15	90%
4.2	Final BOM	All	10/17	11/15	0%
4.3	**Prototype Completion**	All	10/3	11/10	0%
4.3.1	***Enclosure Manufacturing***	Kevin, Lourdes	9/27	11/2	0%
4.3.1.1	Machine Enclosure Compone	Kevin, Lourdes	9/27	10/30	0%
4.3.1.2	Assemble Enclosure Compor	Kevin, Lourdes	10/9	11/2	0%
4.3.2	***Chess Piece Manufacturing***	Kevin, Lourdes	9/27	11/6	0%
4.3.2.1	Manufacture Chess Piece Sh	Kevin, Lourdes	9/27	11/5	0%
4.3.2.2	Insert Magnets	Liam	11/5	11/6	0%
4.3.3	***PCB Manufacturing***	Liam, Lourdes	10/2	11/2	0%
4.3.3.1	Print PCB	Liam, Lourdes	10/2	10/24	0%
4.3.3.2	Solder Components	Liam, Kevin	10/24	11/2	0%
4.3.4	LED Array Manufacturing	Lourdes, John	10/12	11/2	0%
4.3.5	Piece Detection Manufacturin	Liam, Lourdes	10/12	11/2	0%
4.3.6	***Programming Microcontrol***	Paul, John	9/6	10/14	0%
4.3.6.1	Implementing Minimal Kernel	Paul	9/6	9/13	90%
4.3.6.2	Integrating Texas Instrument I	Paul	9/13	9/20	10%
4.3.6.3	Integrating Chess Engine	John, Paul	9/20	9/27	0%
4.3.6.4	Developing Sensor Code to R	Paul	9/27	10/4	0%
4.3.6.5	LED output software driver	John	9/20	9/27	0%
4.3.7	Assemble Subsystems into F	All	11/2	11/10	0%

Fig. 18. Initial Gantt Chart of the Manufacturing Phase.



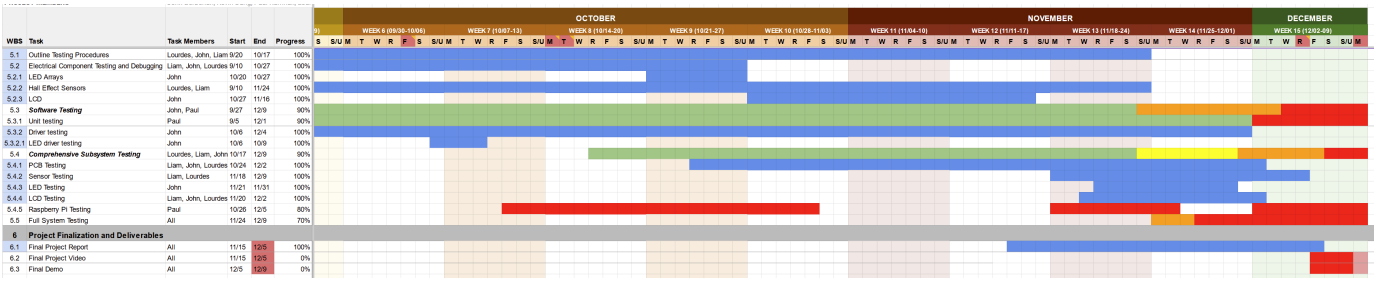
WBS	Task	Task Members	Start	End	Progress
5	**Testing and Debugging**				
5.1	Outline Testing Procedures	Lourdes, John, Liam	9/5	10/17	0%
5.2	Electircal Component Testing	Liam, John, Lourdes	9/10	10/14	20%
5.3	***Software Testing***	John, Paul	9/5	12/5	2%
5.3.1	Hardware emulation	John	9/5	9/27	10%
5.3.2	User emulation	John	9/27	10/7	0%
5.3.3	Unit testing	Paul	9/5	12/1	0%
5.4	***Comprehensive Subsystem***	Lourdes, Liam, John	10/17	11/20	0%
5.4.1	PCB Testing				
5.5	Full System Testing	All	10/18	12/5	0%
6	**Project Finalization and Deliverables**				
6.1	Final Project Report	All	11/15	12/5	0%
6.2	Final Project Video	All	11/15	12/5	0%
6.3	Final Demo	All	12/5	12/9	0%
7	**Other Dates**				
7.1	Fall Break	All	10/12	10/15	0%
7.2	Thanksgiving Break	All	11/27	12/1	0%
	STS 4500 Prospectus				
7.3.1	Prospectus draft due date	Liam, Paul		11/8	0%
7.3.2	Prospectus draft due date	Lourdes		11/6	0%
	Travel and Other Events				
7.4.1	Travel	Lourdes	10/16		

Fig. 19. Initial Gantt Chart of the Testing and Finalization Phases and Other Important Dates.



WBS	Task	Task Members	Start	End	Progress
4	**Manufacturing and Assembly**				
4.1	Parts Procurement	John, Paul	8/28	12/2	100%
4.2	Final BOM	All	10/17	12/3	100%
4.3	**Prototype Completion**	All	10/3	12/9	95%
4.3.1	***Enclosure Manufacturing***	Kevin, Lourdes	10/21	11/26	100%
4.3.1.1	Machine Enclosure Components	Kevin	10/21	10/30	100%
4.3.1.2	Assemble Enclosure Components Together	Kevin, Lourdes	10/30	11/26	100%
4.3.2	***Chess Piece Manufacturing***	Lourdes	9/27	12/6	100%
4.3.2.1	Print pieces	Lourdes	9/27	11/5	0%
4.3.2.2	Assemble pieces and smooth out	Lourdes	11/5	11/6	0%
4.3.3	***PCB Manufacturing***	Liam, Lourdes	10/9	11/28	100%
4.3.3.1	Print PCB	Liam, Lourdes	10/9	10/24	100%
4.3.3.2	Solder Components	Liam, Kevin, Lourdes	10/24	11/28	100%
4.3.4	LED Array Manufacturing	John, Kevin, Liam, L	10/12	11/24	100%
4.3.6	***Programming MCU***	Paul, John	9/6	10/14	100%
4.3.6.2	Integrating Texas Instrument Drivers	Paul	9/13	9/20	100%
4.3.6.3	Integrating Chess Engine	Paul	9/27	10/11	100%
4.3.6.4	ADC/Sensors thread	John	10/8	10/11	100%
4.3.6.5	LED output software thread	John	9/27	10/8	100%
4.3.6.6	LCD software driver	John	10/12	10/18	100%
4.3.6.7	UART software driver	Paul, John	11/1	11/27	100%
4.3.7	***Programming Raspberry Pi 5***	Paul, John	9/30	10/25	100%
4.3.7.1	Installing Raspberry Pi OS	Paul, John	9/30	10/4	100%
4.3.7.2	Installing Stockfish	Paul	10/8	10/8	100%
4.3.7.3	Learning Universal Chess Interface (UCI)	Paul	10/8	10/11	100%
4.3.7.4	Implementing Custom UART Protocol	Paul, John	10/8	10/18	100%
4.3.7.5	Implementing Stockfish "wrapper"	Paul	10/24	11/1	100%
4.3.7.7	Implementing undo handler	Paul, John	10/24	11/1	100%
4.3.8	Assemble Subsystems into Prototype	All	11/2	12/9	98%

Fig. 20. Final version of the manufacturing Gantt chart. .



WBS	Task	Task Members	Start	End	Progress
5.1	Outline Testing Procedures	Lourdes, John, Liam	9/20	10/17	100%
5.2	Electrical Component Testing and Debugging	Liam, John, Lourdes	9/10	10/27	100%
5.2.1	LED Arrays	John	10/20	10/27	100%
5.2.2	Hall Effect Sensors	Lourdes, Liam	9/10	11/24	100%
5.2.3	LCD	John	10/27	11/16	100%
5.3	***Software Testing***	John, Paul	9/27	12/9	90%
5.3.1	Unit testing	Paul	9/5	12/1	90%
5.3.2	Driver testing	John	10/6	12/4	100%
5.3.2.1	LED driver testing	John	10/6	10/9	100%
5.4	***Comprehensive Subsystem Testing***	Lourdes, Liam, John	10/17	12/9	90%
5.4.1	PCB Testing	Liam, John, Lourdes	10/24	12/2	100%
5.4.2	Sensor Testing	Liam, Lourdes	11/18	12/9	100%
5.4.3	LED Testing	John	11/21	11/31	100%
5.4.4	LCD Testing	Liam, John, Lourdes	11/20	12/2	100%
5.4.5	Raspberry Pi Testing	Paul	10/26	12/5	80%
5.5	Full System Testing	All	11/24	12/9	70%
6	**Project Finalization and Deliverables**				
6.1	Final Project Report	All	11/15	12/5	100%
6.2	Final Project Video	All	11/15	12/5	0%
6.3	Final Demo	All	12/3	12/9	0%

Fig. 21. Final version of the testing Gantt chart.

consolidate subsystems to reduce the number of components to assemble, simplify parts logistics by using a narrower variety of parts with similar functions, and more loosely follow FIDE guidance to reduce the size of the device and its material costs. Automated manufacturing would also have a significant impact on the effective cost of the product. Using stencils and solder paste to assemble the PCBs and injection molds for the chess pieces would all allow for much more rapid production times.

Conservative estimates of costs for a 10,000-unit production order may thus be made. In Tab. IV, the far right column shows the ordering the parts in volume provides a substantial reduction in cost from $346.05 per unit to $262.55, a 24% reduction. This could be only be reduced further with the manufacturing changes we previously discussed.

X. Final Results

On December 6, 2024, the original publication date of this report, the functionality of the project was as follows: when the user plugged in the power cable into the I/O board, the game automatically began if the pieces were in the correct starting position. If this was not the case, the lights would not turn on until the board was in the correct position. In this state, the white player's movable pieces were highlighted. When any piece was picked up, all available moves for that piece were indicated by the LEDs. For pieces that were not movable, no LEDs were highlighted. An issue prevented the turns from swapping to the other player. The piece movement is visible in Fig. 23 and Fig. 24.

However, what was present at that point indicated that we had a complete understanding of the board state and

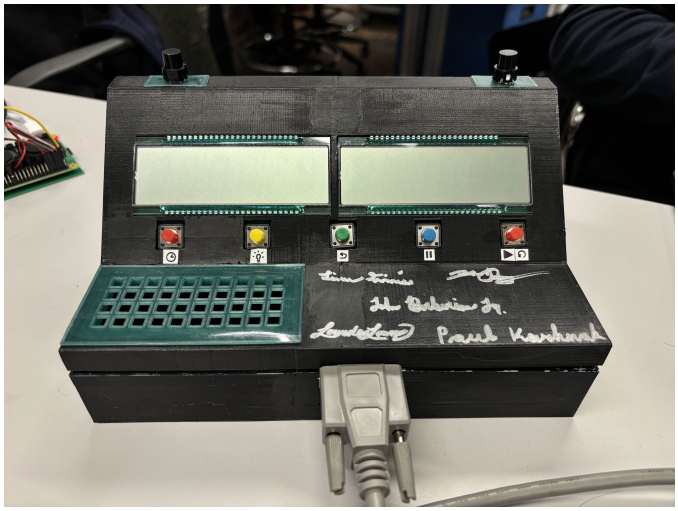


Fig. 22. Final version of the clock box. The top two buttons correspond to each of the players' turn swap, the while the others correspond to clock, hint, undo, pause, and restart, respectively.

accurate readings of our sensors. Additionally, the LEDs were functioning as expected. Further testing and debugging showed errors in the Stockfish wrapper on the Raspberry Pi that caused the code to crash and prevented the entire CHESSBOARD system from functioning as intended as a result.

For the project proposal, we outlined five items to aim for completing by the day of the demonstration to achieve full marks. Firstly, we needed a proper enclosure and all physical

Fig. 23. The start state of the untimed board. All available moves are highlighted with the LEDs.

Fig. 24. The white D pawn being moved. The LEDs light to indicate which spaces are available for movement.

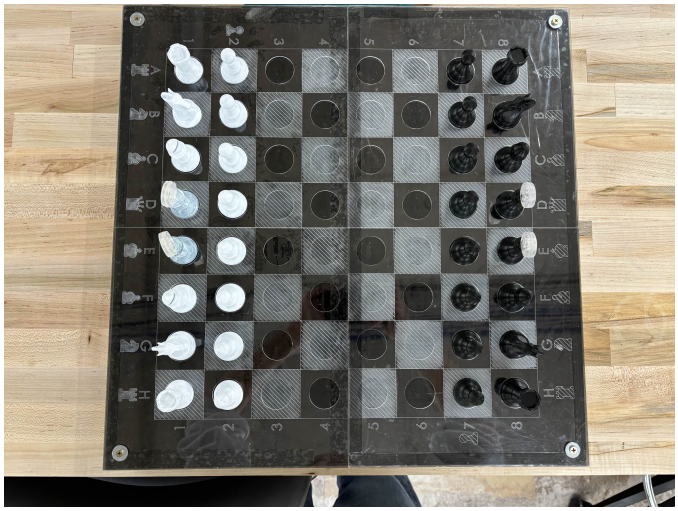


Fig. 25. A top down view of the completed chessboard.

Fig. 26. A side view of the chessboard and clock box.

elements required to play chess. Secondly, we needed an LED array that allows us to communicate with the players to indicate what pieces they can interact with. We also decided to include the ability to transmit a complete game transcript via USB to study and import into other chess software. Additionally, to facilitate newer players, hint and undo buttons were designed and included. Lastly, we decided it was important to include a chess clock to encourage newer players to approach the game with a different mindset.

First, it is apparent based on Fig. 25 or Fig. 26 that all of the physical materials necessary for chess are present. All pieces, as well as additional pieces to accommodate pawn promotion, are present. Additionally, based upon Fig. 23 and Fig. 24, the LEDs can be seen to function as intended, highlighting available moves and guiding the players into legal moves.

The remainder of the features (hint, undo, clock, and transcript) were implemented in software but had bugs as of this report's initial publication. All the hardware required for these features was functional, however. We expected that these bugs would be resolved during the period from December 6, 2024, to December 9, 2024, between this report's initial submission and the final demonstration of the project. Since transcription depended on a fully functional game to be tested, the transcription feature also could not be verified to work as of December 6, 2024.

Based on the outline established, the work that we achieved as of December 6 would be deserving of a B. However, considering our proximity to full functionality at that point and our confidence in achieving these aspects of the project by the time of the prototype demonstration, we still believed that

the effort that had been placed into the project was deserving of an A.

We have preserved the above paragraphs in order to document the state of the project when we turned in this report in Fall 2024, three days before the demonstration. Over the weekend that followed, we worked ceaselessly toward full functionality. We are very pleased to report that we successfully achieved every goal laid out in this project and scored an A+ for our efforts. Every system worked completely on demo day[4]—though one of the turn-switching buttons needed to be re-glued after an enthusiastic child hit it too hard. A quick demonstration video, filmed and edited by Lourdes, may be found here.

XI. Engineering Insights

A. Reinventing the Wheel

One of the foremost lessons from this project was to not "reinvent the wheel" and instead employ existing stable tools as project components wherever possible. For example, software tasks were progressively offloaded to well-tested external libraries, packages, and projects. Initially, the team planned to write a bespoke RTOS kernel and manually port a chess engine to the MSPM0's limited embedded environment. Using the stable, professionally maintained FreeRTOS for the MSPM0, however, proved a superior alternative. With FreeRTOS's comprehensive interfaces for task management, data structures, and synchronization, among other features, the team saved countless hours of development time by using existing "off-the-shelf" products instead of opting for a "do-it-yourself" approach. Moreover, when the more powerful Raspberry Pi emerged as a platform to run a chess engine on, the team elected to run Stockfish on the Raspberry Pi and engineer communication between the Raspberry Pi and MSPM0 rather than engineer a full-fledged stripped-down chess engine on the MSPM0 itself. Yet more time was saved by using Python's `chess` and `serial` modules to use trusted code for chess game management and UART communication, respectively. In all, reducing the scope of novel software we developed allowed for easier testing and debugging. If existing libraries and projects could simply be assumed to work *a priori* (often, though not always, justified), we could, and did, ultimately concern ourselves with simply testing the bespoke code written to interface with those software tools rather than testing all software involved in the project.

In comparison to software, the hardware side of the project largely avoided this issue by prioritizing the aspects of design that only we could implement, such as the filter values and the overall block diagram. Even in these aspects, we utilized tools to make them as simple as possible, such as the Texas Instruments analog filter designer. For other aspects, we used widely available and common standards to have as much information and references to take from as possible. For instance, we used a commonly available DE-15 adapter and cable to transmit power and information between the clock box and chessboard. These actions meaningfully avoided "reinventing the wheel" with respect to the hardware.

[4]Punctuated by the successful application of a "hotfix" for the Stockfish wrapper's Python code approximately 15 minutes before our demonstration.

B. Trust, but Verify

Another critical lesson we learned through this experience was to trust our judgment but also verify our work. Assumptions that critical project software worked were always justified with extensive testing and, where needed, debugging. One of the foremost software examples to demonstrate this was the UART testing (see Section III-F4). Initially, UART was configured properly and appeared to work on the Raspberry Pi. When a new microSD card had to be prepared to replace a broken one, however, the new kernel version bundled with the installation contained a patch that disrupted the ability to access memory necessary for UART communication. While further kernel patches have since addressed this problem, the original solution was to revert the kernel version to one issued before the patch. This put into practice the principle that no software was treated as infallible.

This lesson was especially relevant for the hardware. Testing our system, much like with the software, was a critical part of guaranteeing correct functionality. To this end, we took great care to check our work throughout the assembly and design process. However, there were still aspects of the project in which our overconfidence led to problems that could have been prevented with more careful reflection. One such issue was the faulty footprints on our PCBs. This issue was largely preventable—indeed, it should have been caught during the design phase of the project—but was able to go unnoticed until the assembly phase, resulting in the need for several unexpected modifications to the components to allow them to fit within their expected positions. In the case of the I/O PCB, a second revision needed to be ordered to resolve the footprint issues. Although trust in our capabilities was critical to actually creating a functional end product, verifying our work thoroughly was a key factor we needed to pay more attention to during the development of the project. A hardware problem with UART on the MSPM0 also demonstrated this principle. Like with the Raspberry Pi, basic examples were used to verify that the UART functioned as expected for simple TI-provided code snippets. This helped diagnose a UART issue on the MSPM0 where a hardware defect led to two of the GPIO pins failing to propagate the intended UART messages electrically. A hasty fix somewhat analogous to the kernel reversion on the Raspberry Pi was soon put in place to address the underlying problem.

One way we were able to proactively follow this lesson was by providing ample surplus parts during prototyping. This was initially avoided because we trusted our soldering ability. Although our soldering ability by itself was generally sufficient, other design issues meant that additional multiplexers and sensors needed to be ordered individually, which incurred fairly expensive shipping costs. This shipping is what caused us to exceed the budget cap of $500, and it could have been avoided if additional parts were ordered from the beginning. A

better approach would have been to order at least 20% more of every part than we really needed, to account for adjustments that needed to be performed; this would have become valuable as certain aspects of the design demonstrated that they would not function as intended without alteration.

For both software and hardware, thorough testing that implemented conservative assumptions allowed the team to catch potentially product-breaking errors. Especially for UART, discarding the assumption that UART drivers and hardware simply worked as-is enabled the team to catch critical errors. While the team could not always anticipate these errors ahead of time (especially for UART, where the errors were unusual), the team could nonetheless pinpoint the source of the problem immediately rather than have to repeatedly vet the entire project's hardware and software every time an error presented itself.

C. Nights and Weekends

Lastly, an important lesson emerged from the pace at which this project was developed. Initially, we expected this project would proceed at a constant pace. To start from the best place possible, we all put ourselves into this project completely and get a great work pace even before the start of this semester. However, we failed to anticipate our external obligations and struggled to properly balance this project with many of our competing priorities. By the midpoint of the semester, this project was just meeting the deadlines established by the class. With the addition of underestimations for the assembly period, this project was completed only with great time commitments during the last few weeks of the semester.

The main takeaway from this section is that even with intentions of maintaining a good development pace, without the will to act and a thorough plan of doing so, said pace can fall below the established expectations.

XII. Future Work

Many of the changes the team would have made to the design have been outlined in Section IV-D. The main changes would be to reduce the number of parts and simplify manufacturing. The sensor boards should be turned horizontally in future versions of the product and combined together with the I/O board into fewer, larger parts. Components should be standardized across PCBs in the system and be SMD only rather than a mix of SMD and through-hole. The D-Bus connector should be swapped for the low density variant with only two rows to help ensure it is easy to run properly sized traces to each pin. Compression of the acrylic should be done mechanically with rivets rather than with adhesives.

For future student teams who may want to tackle a similar project concept, we would like to preface that some of these changes anticipate industrial manufacturing capacities that student teams may not have. The choice of SMD components, for example, is great for package size and manufacturing time if a designer has access to a system that allows for large scale solder paste and stencil methods. One might be able to achieve similar results with a heat gun but may deem it more practical to use through-hole component since many students are more likely to be comfortable soldering through-hole components.

The aesthetics and interface could also be improved upon to enhance user experience during gameplay. Since we spent the majority of the semester manufacturing and debugging the project (even up until the day of our showcase), we did not have time to make all the redesigns and implement all the features we wanted to. Features and components the team would like to add or change beyond those geared toward mass production include adding puzzle modes, using a more complex display option, trying out omnidirectional Hall effect sensors, and trying out a resistor-based piece identification system. With the limited time we had, we did not expect to be able to implement many more features than those outlined in our original proposal. However, that does not mean these additional features are unimportant. For instance, the team thinks a commercial learning aid with the option for puzzles to give the user more options for learning how to strategize and think about playing chess. Other improvements could be made to the aesthetics and interfaces to enhance user experience during game play. For example, the current end player buttons should be replaced by larger buttons that are also possibly external to the clock box module to prevent the wearing down of both the buttons and the clock box enclosure. This would be important as players may try to hit the buttons to end their turn as soon as possible. As such, durability of the buttons is important to consider, especially for Blitz games.

The more complex display idea of the project is driven by a desire to make this product more accessible. Currently, the only way to access the transcription feature is to have a USB flash drive and a computer to read the output file. This makes owning a computer a limiting factor for the accessibility of our device. A more complex display could offer the ability to use the transcription feature without additional financial investments (owning a computer).

Omnidirectional Hall effect sensors and the resistor concept are all ideas the team thought of as we progressed further into the project and ran into concerns with piece detection consistency. Needing to bend the through-hole sensors to reduce the space between the sensors' face and the ceiling of the enclosure allows for deviations in distances and angles relative to the enclosure ceiling. The omnidirectional sensor would help with issue of the relative angle.

The resistor option would be an attempt to completely circumvent the magnetic system. Magnets may still be desirable to snap the pieces into the same position every time, but they would not be essential for the core piece detection system. In a resistive design, two circular, concentric contacts could be put on the board, with pieces containing corresponding contacts and an internal resistor of a value unique to the piece's color and type. The probability of misidentification with this system is far smaller than with the magnetic system, where any physical difference in the position of the piece relative to the sensor could result in misidentification. One downside would be the aesthetic of the board, which would have visible ring contacts on every tile.

The team generally advises future teams to put more thought into manufacturing and system integration. When reviewing PCB layouts, read the datasheet for major components and make sure the connections align with what the pinout is on the datasheet. Avoid soldering to wires wherever possible and use standard, pre-made wire harnesses if this type of soldering is still needed. Try to use harness or connector standards with locking mechanisms to make electrical connections more reliable. Minimize the use of adhesives and use mechanical joining systems instead. When there are repeated manufacturing processes, try making rigs to help with consistency and reduce time spent on the task. Include tolerance in manufactured components that give room for small mistakes and individual differences between parts. Lastly, buy backups of critical components. Planning ahead might save money since bulk orders result in reduced costs per unit.

In the Spring 2025 semester, following the completion of this initial project, Liam, Kevin, Lourdes, and John worked on a second revision of the CHESSBOARD with an entirely different sensing system based on resistors and contact pads. This engaged with one of the major areas of future work outlined here. The simplicity of manufacturing the design was also improved substantially: we designed it such that the majority of the soldering could be performed with a single bake in a reflow oven. A more detailed technical description of the second revision may be found here.

XIII. Using This Report

The technical team (John Berberian Jr., Kevin Dang, Paul Karhnak, Lourdes Leung, and Liam Timmins) wish to license this technical report under a Creative Commons BY-NC-SA license to allow noncommercial use that attributes the authors so long as works which derive from our report reproduce our licensing conditions. While the SA provision of our license may introduce some hurdles, we believe this provides the greatest protection against the commercialization of our work without our permission or involvement. We do not wish our license terms to be an undue burden on scholarly use, however, and invite any questions about using this report to any of the authors via the email addresses given on the title page.

References

[1] M. Mcilyar, "Chess Automation for Accessibility / Female Gamers and Their Struggles with Online Gaming," University of Virginia, May 9, 2023. DOI: 10.18130/TFPH-YN34. [Online]. Available: https://libraetd.lib.virginia.edu/public_view/3b5919733 (visited on 09/17/2024).

[2] G. Portillo, "Autonomous Chess Robot; The Semiconductor Industry," University of Virginia, School of Engineering and Applied Science, BS (Bachelor of Science), 2023, Charlottesville, VA, May 12, 2023. [Online]. Available: https://doi.org/10.18130/1wkr-h829 (visited on 09/17/2024).

[3] S. Chittari, "Assistive Chessboard; The Struggle Over Artificial Intelligence in Healthcare," University of Virginia, School of Engineering and Applied Science, BS (Bachelor of Science), 2023, Charlottesville, VA, May 9, 2023. [Online]. Available: https://doi.org/10.18130/s56k-f620 (visited on 09/17/2024).

[4] J. M. Mukundh Balajee Edison Aviles, "Tactile Chess," Carnegie Mellon University, Apr. 29, 2023. [Online]. Available: https://course.ece.cmu.edu/~ece500/projects/s23-teama0/wp-content/uploads/sites/218/2023/05/Team_A0_Aviles_Balajee_Mejia_final_report.pdf (visited on 05/04/2025).

[5] NVE Corporation, *Ad024-10e: 2.8 mt digital switch, rohs tdfn*. [Online]. Available: https://www.nve.com/webstore/catalog/product_info.php?cPath=27_31_41&products_id=561 (visited on 09/19/2024).

[6] DIY Machines, *Diy super smart chessboard — play online or against raspberry pi*. [Online]. Available: https://www.instructables.com/DIY-Super-Smart-Chessboard-Play-Online-or-Against-/ (visited on 09/19/2024).

[7] *Mixed-signal microcontrollers with can-fd interface*, MSPM0G3507, Rev. C, Texas Instruments, Oct. 2023. [Online]. Available: https://www.ti.com/lit/ds/symlink/mspm0g3507.pdf.

[8] "Mspm0-sdk: Mspm0 software development kit." (Aug. 2024), [Online]. Available: https://www.ti.com/tool/MSPM0-SDK (visited on 09/20/2024).

[9] "Lp-mspm0g3507: Mspm0g3507 launchpad™ development kit for 80-mhz arm cortex-m0+ mcu." (2024), [Online]. Available: https://www.ti.com/tool/LP-MSPM0G3507?keyMatch=mspm0g3507%20launchpad&tisearch=universal_search (visited on 09/20/2024).

[10] *Raspberry pi 4*, Raspberry Pi Ltd, Feb. 2025. [Online]. Available: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf.

[11] *Ratiometric linear hall effect sensor*, DRV5055, Rev. B, Texas Instruments, Jan. 2021. [Online]. Available: https://www.ti.com/lit/ds/symlink/drv5055.pdf.

[12] *Debounce a switch*, SCEA094, Texas Instruments, Oct. 2022. [Online]. Available: https://www.ti.com/lit/ab/scea094/scea094.pdf.

[13] *5-v bidirectional 8:1, 1-channel multiplexer*, TMUX1208, Rev. C, Texas Instruments, Dec. 2018. [Online]. Available: https://www.ti.com/lit/ds/symlink/tmux1208.pdf.

[14] *Disk neodymium magnets*, N35-8195, Radial Magnets. [Online]. Available: https://radialmagnet.com/wp-content/uploads/2017/01/Disk%20Neodymium%20Magnets%20N35-8195.pdf.

[15] *APA102-2020 SuperLED*, APA102, Shenzhen LED Color Opto Electronic CO., LTD, Jul. 2021. [Online]. Available: https://www.mouser.com/datasheet/2/737/APA102_2020_SMD_LED-2487271.pdf.

[16] *Intelligent control led integrated light source*, WS2812, WorldSemi. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/WS2812.pdf.

[17] *Dotstar leds*, DotStar, Adafruit, Jun. 2024. [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-dotstar-leds.pdf.

[18] *Integrated Light Source Intelligent Control (Double line transmission) of chip-on-top SMD type LED*, SK9822, Opsco Optoelectronics, Mar. 2016. [Online]. Available: https://www.digikey.com/htmldatasheets/production/1843597/0/0/1/sk9822.pdf.

[19] *General-purpose triple-channel digital isolator with robust emc*, ISO6731, Rev. B, Texas Instruments, Feb. 2023. [Online]. Available: https://www.ti.com/lit/ds/symlink/iso6731.pdf.

[20] *High power dc connectors*, JACK-C-PC-10A-RA(R), Rev. F, GlobTek, Mar. 2024. [Online]. Available: https://spec.globtek.info/spec/spec_misc?id=01t0c000008OOuxAAG.

[21] *Regulated converters*, RS6-053.3S, RECOM Power, Oct. 2024. [Online]. Available: https://www.mouser.com/datasheet/2/468/RS6-1006283.pdf.

[22] *D-subminiature product catalog*, ICD15S13E4GX00LF, Amphenol ICC. [Online]. Available: https://cdn.amphenol-cs.com/media/wysiwyg/files/documentation/datasheet/inputoutput/io_dsub_brochure.pdf.

[23] *Raspberry pi 5*, Raspberry Pi Ltd, Jan. 2025. [Online]. Available: https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-datasheet.pdf.

[24] *5mhz, 15v/µs high slew-rate, rrio op amp*, TLV9052IDR, Rev. J, Texas Instruments, Feb. 2024. [Online]. Available: https://www.ti.com/lit/ds/symlink/tlv9052.pdf.

[25] *0.71" character height, six digit lcd glass, tn, reflective, 50 pins*, LCD-S601C71TR, Lumex Opto/Components Inc., Nov. 1998. [Online]. Available: https://www.lumex.com/spec/LCD-S601C71TR.pdf.

[26] *32-segment cmos lcd driver*, AY0438/P, Texas Instruments, 1995. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/80438a.pdf.

[27] *Octal buffers and line drivers with schmit trigger inputs, 3-state outputs, and flow-through pinout*, SN74HCS541, Rev. B, Texas In-

struments, Oct. 2021. [Online]. Available: https://www.ti.com/lit/ds/symlink/sn74hcs541.pdf.
[28] *5v 10a switching power supply*, 658, Adafruit, Jan. 2016. [Online]. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/2322/658_Web.pdf.
[29] R. Barry, "Freertos-a free rtos for small embedded real time systems," 2006.
[30] A. Gerrand. "Share memory by communicating – the go programming language." (Jul. 2010), [Online]. Available: https://go.dev/blog/codelab-share (visited on 12/06/2024).
[31] TimEdwards, *Openscad_chess*. [Online]. Available: https://www.thingiverse.com/thing:585218 (visited on 09/19/2024).
[32] "Ieee standard for an age appropriate digital services framework based on the 5rights principles for children," *IEEE Std 2089-2021*, pp. 1–54, 2021. DOI: 10.1109/IEEESTD.2021.9627644.
[33] "Fide handbook," *General Rules and Technical Recommendations for Tournaments*, 2017. [Online]. Available: https://www.fide.com/FIDE/handbook/Standards_of_Chess_Equipment_and_tournament_venue.pdf/.
[34] "Ieee recommended practices for modulating current in high-brightness leds for mitigating health risks to viewers," *IEEE Std 1789-2015*, pp. 1–80, 2015. DOI: 10.1109/IEEESTD.2015.7118618.
[35] CUI, Inc., "Power supply safety standards, agencies, and marks," Jan. 1, 2020. [Online]. Available: https://www.cui.com/catalog/resource/power-supply-safety-standards-agencies-and-marks/.
[36] *Privacy rights for california minors in the digital world*, 2013. [Online]. Available: https://leginfo.legislature.ca.gov/faces/codes_displayText.xhtml?lawCode=BPC&division=8.&title=&part=&chapter=22.1.&article.
[37] R. Socorregut, "Sensory chessboard and method for detecting positions of chess pieces on a chessboard and transmitting those positions to a computer or other electronic recording device," US11369862B2, Jul. 2017. [Online]. Available: https://patents.google.com/patent/US11369862B2/en?q=(chessboard)&oq=chessboard&page=1 (visited on 12/05/2024).
[38] D. Ferguson, "Board game move recording system," US3843132A, Apr. 1973. [Online]. Available: https://patents.google.com/patent/US3843132A/en?q=(chessboard)&oq=chessboard&page=1 (visited on 12/05/2024).
[39] D. B. Bathurst, "Electronic chess game," US4082285A, Nov. 1976. [Online]. Available: https://patents.google.com/patent/US4082285A/en?q=(chessboard)&oq=chessboard&page=1 (visited on 12/05/2024).
[40] D. Yang, *About*, 2022. [Online]. Available: https://stockfishchess.org/about/ (visited on 05/08/2025).

APPENDIX

A. 32-bit Protocol Specification

The protocol has four modes. They are detailed in their respective subsections. The general protocol details are described below. Example packets (with packing order) are shown in Fig. 27, on the next page.

The ranks and files of the chessboard are encoded as 3-bit integers, starting at 0 (*e.g.*File A$\rightarrow$ 0, Rank 1$\rightarrow$ 0). This allows us to encode a square in 6 bits: file, then rank. Each move includes a source and destination square. Some moves also include a "second move," if a second piece is moved as a result of the chess move (*e.g.*en passant, castle in normal mode or undoing a take in undo mode). In order to support piece promotion, the protocol also sometimes includes a field to indicate the type of piece expected to be placed on the destination square. The piece type is encoded as a 3-bit integer, with the mapping below:

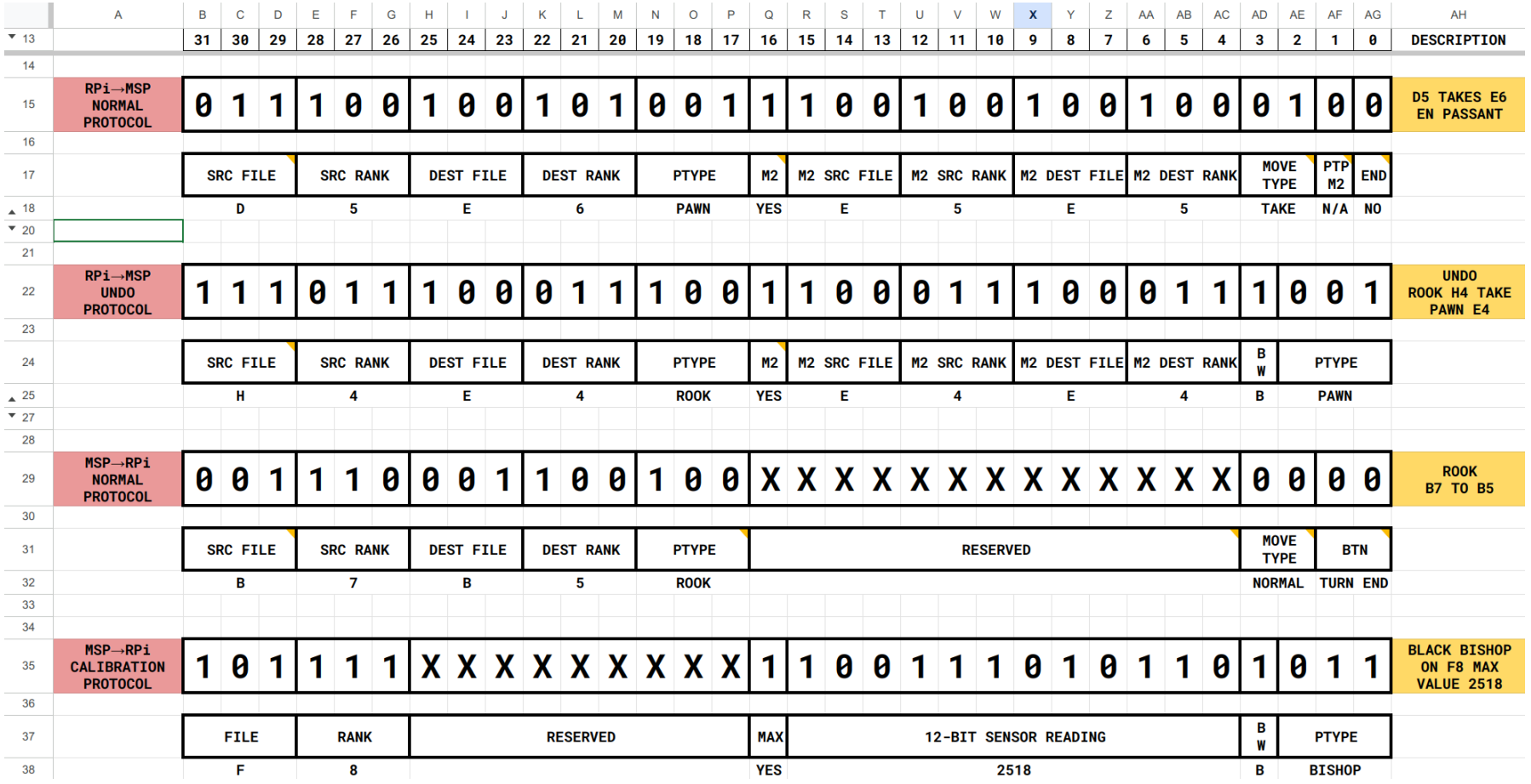
0) Empty square
1) Pawn
2) Knight
3) Bishop
4) Rook
5) Queen
6) King
7) Empty square

1) RPi→MSP normal: The normal protocol includes two additional fields not described above: move type (normal, check, capture, castle/promote), second move piece type (empty or rook), and last (not last or last). Move type is encoded as a two-bit integer, with the options numbered from 0 to 3 in the order listed above. The move type is required for the differences in LED rendering (red for takes, purple for promotion, etc.). Similarly, the second move piece type, describing the expected piece type on the destination square of the second move, is either nothing (en passant) or rook (castle). The "last" bit indicates whether a move is the last one in the sequence of legal moves that are being sent.

2) RPi→MSP undo: The undo packet type must also encode the information required to reverse a take, so we expanded the piece type specifications by removing the last-move and move type fields. Because the second move piece color could be either the same (castle) or different (take) from the color of the main move, we also added a single bit to indicate whether the second move piece was white (0) or black (1). The encoding and packing is otherwise identical to allow for maximum compatibility with the tools developed to parse the normal protocol.

3) MSP→RPi normal: Valid moves may be uniquely identified by the source and destination square along with the type of piece placed on the destination square. The moves are validated on the MSP side, so every move encoded with this scheme is guaranteed to be correct. This obviates the need for much of the information in the move packet, freeing up enough space for the MSP to encode button presses. Again, the fields were aligned to provide maximum compatibility with the other protocol modes' parsing code.

4) MSP→RPi calibration: The calibration code needs to send only a few pieces of information: for each square and each piece type of each color, we should record the maximum and minimum sensor values measured. The sensor value is a 12-bit integer, so encoding all of this information required splitting up the data into two packets: one for max and one for min. There is an indicator bit in the same position as the second-move indicator to discriminate between these two packet types.



	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	DESCRIPTION
RPi→MSP NORMAL PROTOCOL	0	1	1	1	0	0	1	0	0	1	0	1	0	0	1	1	1	0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	D5 TAKES E6 EN PASSANT
	SRC FILE			SRC RANK			DEST FILE			DEST RANK			PTYPE			M2	M2 SRC FILE			M2 SRC RANK			M2 DEST FILE			M2 DEST RANK			MOVE TYPE		PTP M2	END	
	D			5			E			6			PAWN			YES	E			5			E			5			TAKE		N/A	NO	
RPi→MSP UNDO PROTOCOL	1	1	1	0	1	1	1	0	0	0	1	1	1	0	0	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	1	UNDO ROOK H4 TAKE PAWN E4
	SRC FILE			SRC RANK			DEST FILE			DEST RANK			PTYPE			M2	M2 SRC FILE			M2 SRC RANK			M2 DEST FILE			M2 DEST RANK			B W	PTYPE			
	H			4			E			4			ROOK			YES	E			4			E			4			B	PAWN			
MSP→RPi NORMAL PROTOCOL	0	0	1	1	1	0	0	0	1	1	0	0	1	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0	ROOK B7 TO B5
	SRC FILE			SRC RANK			DEST FILE			DEST RANK			PTYPE			RESERVED													MOVE TYPE		BTN		
	B			7			B			5			ROOK																NORMAL		TURN END		
MSP→RPi CALIBRATION PROTOCOL	1	0	1	1	1	1	X	X	X	X	X	X	X	X	X	1	1	0	0	1	1	1	0	1	0	1	1	0	1	0	1	1	BLACK BISHOP ON F8 MAX VALUE 2518
	FILE			RANK			RESERVED									MAX	12-BIT SENSOR READING												B W	PTYPE				
	F			8												YES	2518												B	BISHOP				

Fig. 27. Sample 32-bit Protocol Field Packing.

B. Cost Breakdown Tables

TABLE IV
COST BREAKDOWN FOR FULL ORDER COST, COST OF PROTOTYPE UNIT, AND BULK MANUFACTURING COSTS.

	Brand/Supplier	Distributor	Product Name	Part Number	Part Number (Distributor)	# in Stock	Quantity Ordered	Quantity Needed	Unit Cost	Cost/ Prototype	Total (including tax and shipping)	10000 Unit Quantity	Bulk Purchase Quantity	Bulk Unit Cost	Production Cost	Savings for 10000 units
2	Texas Instrument	Texas Instrumen	Launchpad dev	LP-MSPM0G350	x	x	2	1	$16.99	$16.99	$47.78	10000	1	$16.99	$169,900.00	$0.00
3	Radial Magnets,	Digi-Key	Magnet	8195	469-1005-ND	x	10	40	$0.19	$7.48	$1.87	400000	5000	$0.09	$35,140.00	$39,660.00
4	Radial Magnets,	Digi-Key	Magnet	9093	469-1072-ND	x	10	0	$0.33	$0.00	$3.28	x	x	x	x	x
5	Texas Instrument	Digi-Key	Linear hall effect	DRV5055A1QLF	296-49441-ND	x	3	8	$0.83	$6.64	$2.49	80000	1000	$0.34	$26,884.00	$39,516.00
6	Texas Instrument	Digi-Key	Linear hall effect	DRV5055A3QLF	296-49443-ND	x	3	0	$1.33	$0.00	$3.99	x	x	x	x	x
7	Texas Instrument	Digi-Key	Linear hall effect	DRV5055A4QLF	296-49444-ND	x	3	0	$1.92	$0.00	$5.76	x	x	x	x	x
8	Extra Shipping+T	Digi-Key	x	x	x	x	1	x	$3.62	$0.00	$3.62	x	x	x	x	x
9	Adafruit	Digi-Key	5V 50W AC ada	658	1528-1519-ND	59	1	1	$29.95	$29.95	$29.95	10000	1	$29.95	$299,500.00	$0.00
10	Adafruit	Digi-Key	144-LED strip	2241	1528-1481-ND	32	1	0.5694444444	$49.95	$28.44	$49.95	5694.444444	1	$0.35	$1,975.26	$282,462.24
11	RECOM Power	Mouser	Isolated DC/DC	RS6-053.3S	919-RS6-053.3S	24	1	1	$14.54	$14.54	$14.54	10000	264	$11.98	$119,800.00	$25,600.00
12	GlobTek	Digi-Key	10A barrel plug	JACK-C-PC-10A	1939-1086-ND	1,437	1	1	$3.61	$3.61	$3.61	10000	3000	$1.79	$17,875.00	$18,225.00
13	Lumex Opto/Com	Digi-Key	7-segment LCD	LCD-S601C71T	67-1796-ND	2,015	3	2	$5.150	$10.30	$15.45	20000	2268	$3.46	$69,200.00	$33,800.00
14	Texas Instrument	Digi-Key	Shift Register	SN74AHC59N	296-4619-5-ND	1,805	14	0	$0.527	$0.00	$7.38	x	x	x	x	x
15	Radial Magnets,	Digi-Key	Magnet	8195	469-1005-ND	9,704	40	x	$0.187	$0.00	$7.48	x	x	x	x	x
16	Texas Instrument	Digi-Key	Linear hall effect	DRV5055A1QLF	296-49441-ND	1,033	70	x	$0.543	$0.00	$38.01	x	x	x	x	x
17	Texas Instrument	Mouser	Isolated Level Si	ICD15S13E4GX	595-ISO6731FD	3,146	3	2	$2.010	$4.02	$6.03	20000	4000	$1.01	$20,200.00	$20,000.00
18	Amphenol	Digi-Key	DB-15 Connector		609-5181-ND	8,714	3	1	$1.950	$1.95	$5.85	10000	2520	$0.72	$7,246.80	$12,253.20
19	Adam Tech	Digi-Key	3-pin header	RS1-03-G	2057-RS1-03-G	10,584	70	18	$0.180	$3.24	$12.60	180000	10020	$0.09	$16,668.00	$15,732.00
20	Texas Instrument	Digi-Key	Analog Mux	TMUX1208	296-51847-1-ND	9,422	10	9	$0.507	$4.56	$5.07	90000	10000	$0.27	$24,653.70	$20,976.30
21	KEMET	Digi-Key	0.1uF Ceramic C	C320C104K5R5	399-4264-ND	36,086	100	0	$0.084	$0.00	$8.44	x	x	x	x	x
22	C&K	Digi-Key	Pushbutton	D6R90 F1 LFS	401-1978-ND	53,467	2	2	$1.590	$3.18	$3.18	20000	1000	$1.04	$20,828.80	$10,971.20
23	Microchip Tech.	Digi-Key	LCD Driver	AY0438/P	AY0438/P-ND	597	3	3	$4.530	$13.59	$13.59	30000	100	$3.78	$113,400.00	$22,500.00
24	Diotec Semicond	Digi-Key	Bidirectional Dio	P4SMAJ15CA	4878-P4SMAJ15	10,776	2	1	$0.140	$0.14	$0.28	10000	15000	$0.05	$459.30	$940.70
25	Bel Fuse Inc.	Digi-Key	3A Fuse	0685T3000-01	507-1896-2-ND	104,344	5	4	$0.296	$1.18	$1.48	40000	1000	$0.16	$6,504.00	$5,336.00
26	Würth Elektronik	Digi-Key	2x10 2.54mm Fe	61302021821	732-6130202182	1,224	2	2	$0.830	$1.66	$1.66	20000	1000	$0.49	$9,792.00	$6,808.00
27	Amphenol ICC (F	Digi-Key	HD DB15 Male S	10090926-P154	609-4022-ND	1,769	2	1	$2.530	$2.53	$5.06	10000	2430	$0.96	$9,572.40	$15,727.60
28	Amphenol Cables	Digi-Key	HD DB15 Cable	CS-DSDHD15M	CS-DSDHD15M	795	1	1	$15.070	$15.07	$15.07	10000	500	$8.49	$84,850.00	$65,850.00
29	Raspberry Pi	Digi-Key	Power Adapter	SC1153	2648-SC1153-N	1,745	1	0	$12.000	$0.00	$12.00	x	x	x	x	x
30	GlobTek	Digi-Key	USB-C Cable	USBA2C1MOUS	1939-USBA2C1	2,194	1	0	$2.590	$0.00	$2.59	x	x	x	x	x
31	McMaster Carr	McMaster Carr	24"x24" Acrylic	8560K174	8560K174	x	3	3	$19.9200	$59.76	$59.76	30000	1	$19.92	$597,600.00	$0.00
32	McMaster Carr	McMaster Carr	24"x24" Plywood	1125T513	1125T513	x	1	0	$9.5200	$0.00	$9.52	x	x	x	x	x
33	McMaster Carr	McMaster Carr	72" 2x4 Plank	3577N134	3577N134	x	2	2	$5.7700	$11.54	$11.54	20000	1	$5.77	$115,400.00	$0.00
34	JLCPCB	JLCPCB	10xSensor, 5xIO	x	x	x	x		x	$0.00	$42.47	x	x	x	x	x
35	Texas Instrument	Mouser	Low-offset dual (	TLV9052IDR	595-TLV9052IDI	12,060	6	5	$0.5500	$2.75	$3.30	50000	10000	$0.28	$13,900.00	$13,600.00
36	KEMET	DigiKey	100pF oscillator	C0603C101J5G	399-C0603C101	195,268	2	64	$0.1000	$6.40	$0.20	640000	100000	$0.01	$6,828.80	$57,171.20
37	YAGEO	DigiKey	10k 0603 resisto	RC0603FR-101	13-RC0603FR-1	12,936	70	64	$0.0086	$0.55	$0.60	640000	50000	$0.00	$1,305.60	$4,198.40
38	Samsung	DigiKey	2.2uF 0603 cap	CL10A225KQ8N	1276-1183-1-ND	515,134	70	8	$0.0214	$0.17	$1.50	80000	100000	$0.00	$369.60	$1,342.40
39	YAGEO	DigiKey	1.5k 0603 resist	RC0603FR-071	311-1.50KHRCT	411,660	10	8	$0.0120	$0.10	$0.12	80000	25000	$0.00	$152.00	$808.00
40	YAGEO	DigiKey	2.7k 0603 resist	RC0603FR-102	13-RC0603FR-1	15,187	10	8	$0.0120	$0.10	$0.12	80000	70000	$0.00	$143.20	$816.80
41	KYOCERA	DigiKey	15nF 0603 cap	KGM15AR71E1	478-KGM15AR7	33,741	10	8	$0.0360	$0.29	$0.36	80000	100000	$0.01	$400.00	$2,480.00
42	Cal-Chip	DigiKey	10nF 0603 cap	GMC10X7R103	4713-GMC10X7	4,000	10	1	$0.0300	$0.03	$0.30	10000	12000	$0.01	$71.90	$228.10
43	YAGEO	DigiKey	4.7k 0603 resist	RC0603FR-074	311-4.70KHRCT	1,256,839	10	1	$0.0120	$0.01	$0.12	10000	10000	$0.00	$26.30	$93.70
44	YAGEO	DigiKey	6.2k 0603 resist	RC0603FR-076	311-6.20KHRCT	366,692	10	1	$0.0120	$0.01	$0.12	10000	10000	$0.00	$25.50	$94.50
45	YAGEO	DigiKey	2.4k 0603 resist	RC0603FR-072	311-2.40KHRCT	401,168	10	1	$0.0120	$0.01	$0.12	10000	10000	$0.00	$22.10	$97.90
46	YAGEO	DigiKey	3.9k 0603 resist	RC0603FR-073	311-3.90KHRCT	112,250	10	1	$0.0120	$0.01	$0.12	10000	10000	$0.00	$35.70	$84.30
47	Murata	DigiKey	1.1nF 0603 cap	GRM1885C1H1	490-1452-1-ND	25,014	2	1	$0.1000	$0.10	$0.20	10000	12000	$0.01	$109.00	$891.00
48	Samsung	DigiKey	1nF 0603 cap	CL10C102JB8N	1276-1091-1-ND	862,584	10	1	$0.0260	$0.03	$0.26	10000	12000	$0.00	$36.00	$224.00
49	Murata	DigiKey	2.7nF 0603 cap	GRM1885C1H2	490-8274-1-ND	27,663	2	2	$0.1000	$0.20	$0.20	20000	20000	$0.01	$243.80	$1,756.20
50	Texas Instrument	Mouser	Schmitt Triggers	SN74HCS541D	595-SN74HCS5	9,526	2	1	$0.5100	$0.51	$1.02	10000	10000	$0.12	$1,223.00	$3,877.00
51	DFRobot	DigiKey	RPi4 cooling fan		1738-FIT0817-N	44	1	1	$6.5000	$6.50	$6.50	10000	1	$6.50	$65,000.00	$0.00
52	Texas Instrument	Digi-Key	Analog Mux	TMUX1208	296-51847-1-ND	5,938	10	x	$0.450	$0.00	$17.73	x	x	x	x	x
53	Texas Instrument	Digi-Key	Linear hall effect	DRV5055A1QLF	296-49441-ND	572	20	x	$0.644	$0.00	$18.28	x	x	x	x	x
54	JLCPCB	JLCPCB	5xIO_rev2	x	x	x	5	1	$1.280	$1.28	$15.22	10000	10000	$0.24	$2,402.10	$10,397.90
55	Weld On	Delvie's Plastics	5oz Acrylic glue	x	weld_on_16	x	0	0.04	$7.80	$0.31	$8.58	400	1	$7.80	$3,120.00	$0.00
56	McMaster Carr	McMaster Carr	Black-Oxide 18-(	91249A047	91249A047	x	0	0.22	$6.76	$1.49	$7.44	2200	1	$6.76	$14,872.00	$0.00
57	Scotch	Amazon	Double sided tap	667	B00006IF63	x	0	1	$7.53	$7.53	$8.28	10000	6	$2.71	$27,083.33	$48,216.67
58	AdaFruit	AdaFruit	Female-female j	1919	1919	x	0	0.625	$6.95	$4.34	$7.65	6250	100	$5.56	$34,750.00	$8,687.50
59	Ryobi	Home Depot	Hot glue	A1931203	1007794405	x	0	1	$0.66	$0.66	$0.73	10000	120	$0.48	$4,780.00	$1,861.67
60	McMaster Carr	McMaster Carr	L-bracket	1556A26	1556A26	x	0	2	$1.06	$2.12	$1.17	20000	50	$0.79	$15,800.00	$5,400.00
61	McMaster Carr	McMaster Carr	Low-Strength St	90480A003	90480A003	x	0	0.02	$3.33	$0.07	$3.66	200	1	$3.33	$666.00	$0.00
62	Inland	Microcenter	Perforated boar	270256	422790	x	0	0.025	$4.99	$0.12	$5.49	250	1	$4.99	$1,247.50	$0.00
63	BambuLab	BambuLab	PLA spool	x	x	x	0	1	$19.99	$19.99	$21.99	10000	8	$12.99	$129,900.00	$70,000.00
64	Raspberry Pi	Microcenter	RPi4	SC0193(6A)	87320	x	0	1	$44.99	$44.99	$49.49	10000	1	$44.99	$449,900.00	$0.00
65	South Wire	Home Depot	500ft 22 gauge 4	57572444	249571	x	0	0.03333333333	$109.00	$3.63	$119.90	334	1	$109.00	$36,333.33	$0.00
66	McMaster Carr	McMaster Carr	18-8 Stainless S	92141A005	92141A005	x	0	0.04	$1.60	$0.06	$1.76	400	1	$1.60	$640.00	$0.00
67	WPCTEC	Amazon	Window tint	x	B0CMCR2DCM	x	0	0.05	$4.00	$0.20	$4.40	500	1	$80.00	$40,000.00	$3.00
68	TE Connectivity	DigiKey	Square push but	1825910-6	450-1650-ND	x	0	5	$0.16	$0.80	$0.18	50000	5250	$0.07	$3,727.00	$4,273.00
69	McMaster Carr	McMaster Carr	Wood screws	98685A220	98685A220	x	0	0.04	$7.38	$0.30	$8.12	400	1	$7.38	$2,952.00	$0.00
70															$2,625,515.03	$872,961.47
71										$346.05					$262.55	
72																
73												Protoype Unit Cost		346.05		
74												Bulk Unit Cost		262.55		