Networked Chessboard System

By:

Danny Guller

Payton Schutte

Quinn Athas

Final Report for ECE 445, Senior Design, Spring 2026
TA: Wenjing Song

5 May 2026
Project No. 51

Abstract

This project developed and verified a networked physical chessboard hardware platform designed to support customizable software experiences built around real-world chess interaction. The system enables remote players to play chess using physical pieces while maintaining board synchronization through wireless communication and magnetic piece detection. This document explores the problem being solved and the approach taken to develop a flexible hardware foundation for future software expansion. Furthermore, the requirements and designs used to meet those requirements for each subsystem are presented. The compatibility and accuracy of the sensors and analog-to-digital converters are validated through basic calculations and engineering analysis. Finally, the ethical considerations and societal impact of the project are discussed.

Table of Contents

1. Introduction

1.1 Problem

Chess is widely played recreationally, competitively, and educationally, with benefits for concentration, problem-solving, and strategic thinking. Digital chess platforms make remote play easier and more accessible, but they remove much of the physical experience that makes chess engaging. Players no longer interact with real pieces, develop the same spatial awareness of the board, or experience the sense of presence that comes from over-the-board play.

Physical electronic chessboards help bridge the gap between online chess and traditional chess, but many existing solutions are expensive, closed-source, and designed around a fixed software experience. This limits accessibility for casual users, students, educators, and hobbyists. It also limits the ability to modify the system for new features or different use cases.

A closed hardware and software ecosystem makes it difficult for users to build on top of the platform. For example, a user may want to add chess training tools, puzzle modes, custom timers, spectator features, replay systems, or support for other 8x8 board games. If the system is not open or flexible, these additions are difficult or impossible without redesigning the entire product. This creates a need for a lower-cost, customizable physical chessboard platform that preserves the benefits of real-piece interaction while supporting remote digital play and future software expansion.

1.2 Solution

Figure 1. Visual representation of the two-board networked chess system

This project creates a low-cost, open-source networked physical chessboard hardware platform that software can be built around. Rather than designing a single-purpose chess product, the system is intended to serve as a flexible foundation for future development. The hardware provides the core functionality needed for physical remote chess: piece detection, board-state tracking, wireless communication, and user interface.

The design uses magnetic chess pieces, hall-effect sensors, analog-to-digital converters, custom Printed circuit boards (PCBs), Wireless Fidelity (Wi-Fi) networking, and an embedded touchscreen interface. These subsystems work together to detect piece movement, synchronize board states between remote players, and display gameplay information to the user.

Because the platform is open-source and hardware-focused, users and developers can modify the software to fit their own goals. Future software could support chess training, puzzle solving, opening practice, timers, spectator mode, game replay, or other 8x8 board games such as checkers. This makes the project more than an electronic chessboard; it is a customizable physical-digital board platform designed for affordability, flexibility, and continued development.

2. Hardware Design

2.1 Hardware Introduction

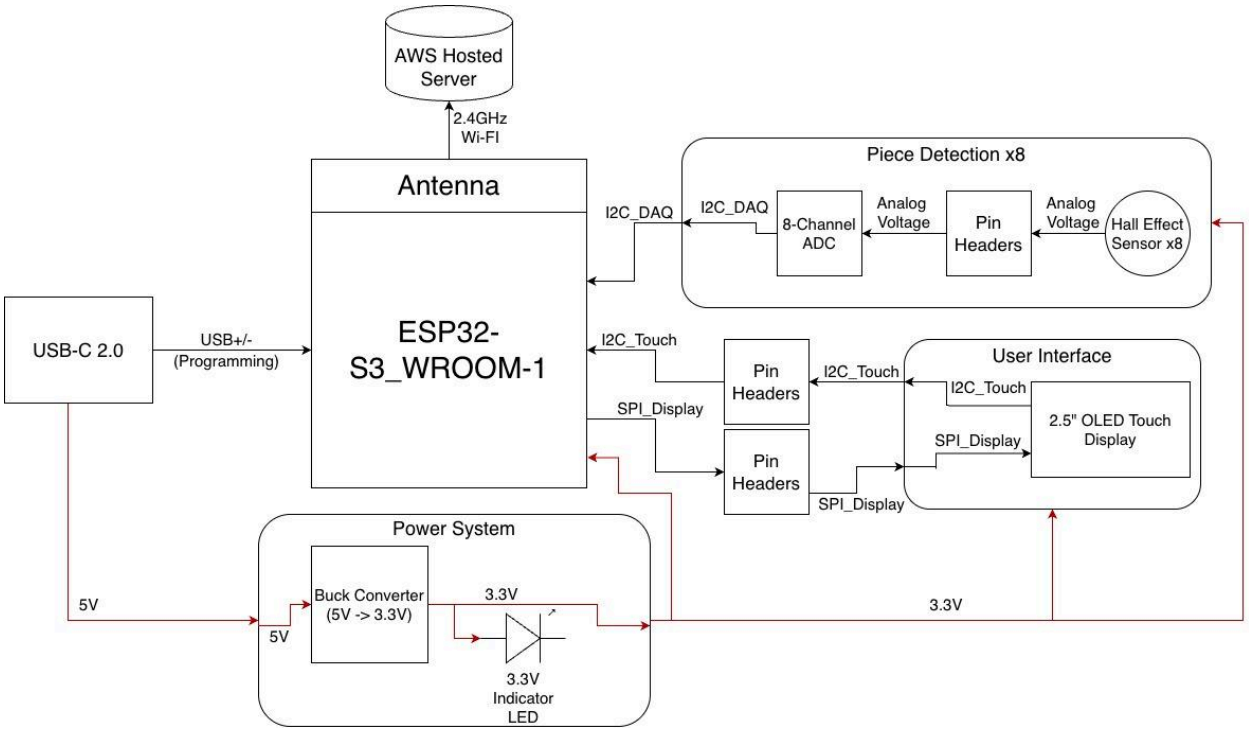


Figure 2. Block diagram of system

The hardware for this project can be broken into four subsystems, those being the power, piece detection, user interface, and compute subsystems as seen in Figure 2. Each hardware subsystem can function independently of each other, but need the compute subsystem to connect them all together and for any meaningful work to be done. All the schematics for these subsystems are found in Appendix E.

Two PCBs were designed and assembled for this project. First is the main PCB that holds every component except the Hall-effect sensors. This PCB acts as a main connection node and compute platform. Second are the tile PCBs that act as the chess board's game tiles. There are 64 tile PCBs per chess board and have a Hall-effect sensor attached to the bottom for magnetic field detection. Each column of eight tile PCBs are wired together and power is daisy chained through them.

For the chess board to function, the Hall-effect sensors need to detect the magnetic field of the magnets embedded into the game pieces. This will cause a voltage drop on the sensor's output channel which is read by the eight channel analog-to-digital converter. This analog-to-digital converter (ADC) then sends a packet over Inter-Integrated Circuit (I^2C) to the microcontroller. The microcontroller then checks game rules for a valid move and sends an updated user interface to the display of Serial Peripheral Interface (SPI). When the user confirms their move on the touchscreen display, data is sent over an independent I^2C

bus back to the microcontroller. This signals the microcontroller to use its built-in Wi-Fi antenna to send the new board state to the Amazon Web Services (AWS) server. This serves as the primary hardware loop for a player's turn. During this loop the power subsystem is supplying a steady 3.3 V to the rest of the subsystems.

A Light-emitting diode (LED) subsystem was originally planned to be included as well. However, during assembly, it was realized that the connection points on the tile PCBs were wrong and that there was no room with the supports for the LEDs. The system was functional and each individual LED lit up based on the move received by the other player, but due to these issues this feature was cut.

2.2 Compute Subsystem

The compute subsystem is the heart of this project's hardware and consists of an ESP32-S3-Wroom-1-N8R8, Universal Serial Bus Type-C (USB-C) 2.0 port, and debug devices. The ESP32-S3 is the microcontroller of this system and was a clear choice for its built-in antenna and Wi-Fi capabilities as well as its large number of digital Input/output (I/O) pins. The Espressif 32-bit microcontroller platform (ESP32) is set up using the ECE445 wiki [1]. This includes strapping pins, reset and boot buttons, and Universal asynchronous receiver-transmitter (UART) programming. The other microcontrollers considered using were more powerful such as a Raspberry Pi or Arduino. These proved to be much more powerful than what the project needed and too expensive. Adding a USB-C 2.0 port for both programming and power was the next step made in the part selection for this subsystem. USB-C has become the most standard connection in consumer electronics and thus is a perfect choice for a consumer chess board.

As far as debug devices go, every power and communication line was broken out to pin headers for easy access and if a signal had no exposed pad/pin one was added to the design. These are important features for bringing up the PCB after assembly and can also help the user troubleshoot a new modification they may make to their chess board.

2.3 Piece Detection Subsystem

This subsystem is responsible for detecting the magnetic field on each game tile and sending the corresponding voltage value to the microcontroller. The magnetic fields of the chess pieces are detected through TI's DRV5055A4QLPG [2] Hall-effect sensors. These sensors are powered by 3.3 V and the output analog voltage is

$$V_{OUT} = V_Q + B \times Sensitivity_{(25℃)} \times (1 + S_{TC} \times (T_A - 25℃)) \quad (1)$$

where V_Q is half of V_{CC}, B is the applied magnetic flux density, $Sensitivity_{(25℃)}$ is 7.5 mV/mT, S_{TC} is 0.12%/℃, and T_A is ambient temperature. For this use case where the exact magnetic field value is not needed and only want to know polarity, distinguish black and white pieces, and the presence of a magnetic field. This characteristic of the Hall-effect sensor is found in Figure 3 where half of V_{CC} is no magnetic field, less than V_{CC} is a north pole field, and more than V_{CC} is south pole field.

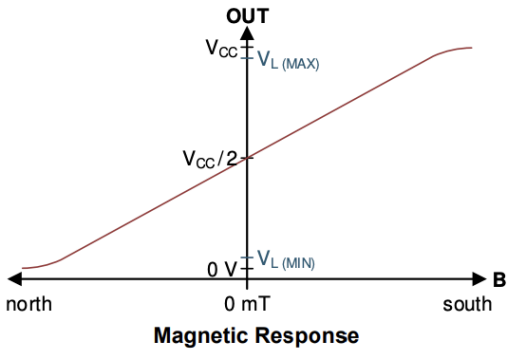


Figure 3. Magnetic response of DRV5055A4QLPG hall-effect sensor

The DRV5055A4QLPG was chosen due to its low cost of $0.54 in the quantities needed, simplicity, and small package. The 12.5 mV/mT, ±169-mT Range is more than accurate enough for our current use case and perfect for future expansion to exact piece detection. The sensor is directional so the through hole variant was chosen so that it could be positioned to read the magnetic field through the PCB. There was a concern that the copper in the PCB would block or disrupt the magnetic field, but there has been no verifiable proof that there is an issue.

The analog voltage of the Hall-effect sensor is wired via pin headers to a channel on the eight channel ADS7128IRTER [3] ADC. This ADC has 12-bits of precision which means it can detect a more than a 12.5 mV change on a channel. This means that the ADC can detect a single mT deviation coming from the Hall-effect. The ADC communicates to the microcontroller over I²C. Each ADC reads one column of the game board and thus eight ADCs are needed. They are each addressed with a resistor voltage divider between the Decap and Addr pins according to Figure 4.

RESISTORS		ADDRESS
R1(1)	R2(1)	
0 Ω	DNP(2)	001 0111b (17h)
11 kΩ	DNP(2)	001 0110b (16h)
33 kΩ	DNP(2)	001 0101b (15h)
100 kΩ	DNP(2)	001 0100b (14h)
DNP(2)	DNP(2)	001 0000b (10h)
DNP(2)	11 kΩ	001 0001b (11h)
DNP(2)	33 kΩ	001 0010b (12h)
DNP(2)	100 kΩ	001 0011b (13h)

(1) Tolerance for R1, R2 ≤ ±5%.
(2) DNP = Do not populate.

Figure 4. ADS7128IRTER I²C addressing table

2.4 User Interface Subsystem

The user interface consists of a 3.5” 480x320 In-plane switching (IPS) capacitive touch screen [4] that communicates over SPI for graphics and an individual I²C bus for touch data. There are other data lines that are connected to the ESP but not used for options like screen brightness, backlight, etc. The display sits on its own I²C line so that the piece detection does not rely on the screen functioning and vice versa. This compartmentalizes the two subsystems and makes repair and debugging easier. The connection from the main PCB to the display is just pin headers again for ease of repairability if something were to break.

2.5 Power Subsystem

The goal of the power system is to convert a 5 V input from the USB-C port or power pin headers to a 3.3±0.2 V at up to 1 A. This max amperage is not expected to be close to hitting 1 A, but the power system being able to handle up to 1A will protect it from shorts or current spikes, making for a robust system. The expected current draw is closer to 300 mA to 500 mA.

The way this is achieved is through a TI TLV62569 [5] synchronous buck converter. A buck converter was chosen over a low dropout regulator for its efficiency. Being a consumer product, power efficiency is a key factor when designing because most users do not want electronics that excessively heat up.

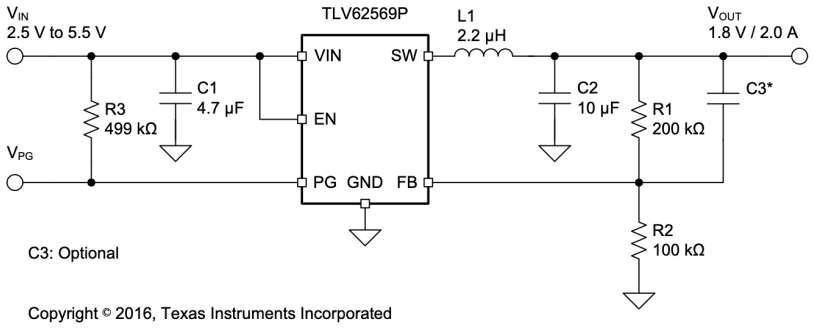


Figure 5. Typical application circuit of TLV62569P synchronous buck converter

The typical application circuit seen in Figure 5 does not match what our project needed. The most notable changes needed were the values of the resistors in the FB pin voltage divider. The output voltage based on resistors one and two is

$$V_{OUT} = V_{FB} \times (1 + \frac{R_1}{R_2}) = 0.6V \times (1 + \frac{R_1}{R_2}) \qquad (2)$$

where V_{OUT} is the output voltage, V_{FB} is the feedback pin voltage (in this case 0.6 V), R_1 is the resistance between the V_{OUT} and V_{FB} nodes, and R_2 is the resistance between the V_{FB} node and ground. The data sheet recommends R_2 to be 100kΩ for the best transient response time and current consumption. Solving the equation gets R_1 to be about 450kΩ. This value had to be changed from the original solution so that it is a standard value. The datasheet then outlines that for $1.8 \leq V_{OUT}$ a 2.2 µH inductor is needed and three output capacitors valued 10 µF each. The enable pin is shorted with the V_{IN} pin so that the regulator is always on.

For debugging and assembly, a solder jumper on the regulator's output and a LED on the output was added so that there is a visual indication there is power and power can be tested before passing it through to the rest of the board.

When fully assembled, the entire electronic assembly runs at 3.28 V at 312±10 mA. This is well within the bounds of each device's power consumption requirements and tolerances.

3. Software Design

3.1 Software Overview

The software for this project is divided into three main components: peripheral interfacing, the AWS-hosted web Application programming interface (API), and the game logic finite state machine. The peripheral interfacing software manages communication between the microcontroller and the hardware subsystems, including the hall-effect sensor array, ADCs, touchscreen display, and user feedback components. The AWS-hosted web API [6] provides the communication layer between remote boards, allowing each system to send moves, retrieve updated board states, and remain synchronized during gameplay. In addition to synchronization, the API also supports chess engine functionality and a messaging feature, expanding the system beyond basic move transfer. The game logic finite state machine controls the flow of the match by managing states such as waiting for a game to begin, detecting a local move, sending that move to the server, polling for an opponent's move, updating the display, and ending the game when appropriate. Together, these components allow the physical board to function as an internet-connected chess platform with room for future software expansion. The following sections break down each component in more detail, explaining its purpose, implementation, and role within the complete system.

3.2 Peripheral Interface Software

3.2.1 ADC Driver

The ADC driver is responsible for reading the hall-effect sensor array used to determine the current state of the chessboard. The board uses eight ADS7128 analog-to-digital converters, each connected over a dedicated I²C bus and addressed from 0x10 to 0x17. Each ADC is responsible for reading one column of the chessboard, with its eight input channels corresponding to the eight ranks in that column. Since the physical wiring order of the ADCs does not directly match the left-to-right file order of the chessboard as shown in Figure 6, the driver uses a column-to-chip mapping array to translate between physical board position and ADC address.

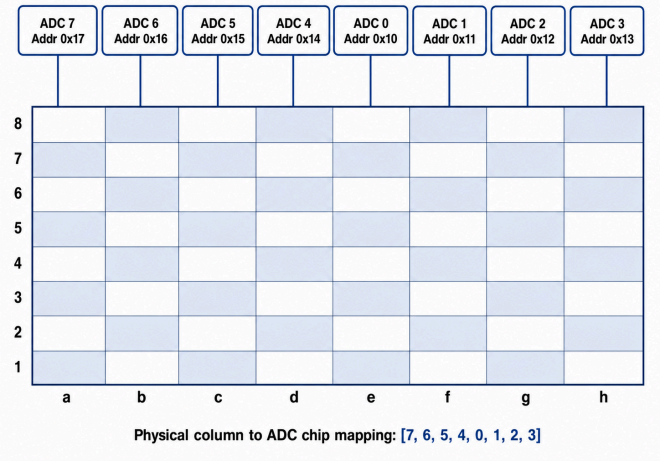


Figure 6. ADC to board mapping

During initialization, the driver starts the I²C bus on the assigned SDA and SCL pins and configures each ADS7128 for analog input operation and manual channel selection. Manual channel selection allows the software to choose a specific ADC channel, briefly wait for the signal to settle, and then read a 12-bit conversion result. The raw ADC result is compared against a fixed midpoint baseline of 2048, which represents the center of the 12-bit ADC range. A threshold of 300 counts is used to determine whether the magnetic field indicates a piece is present and which polarity is detected.

The readBoardFEN() function converts the sensor readings into a simplified Forsyth-Edwards Notation (FEN) style board representation [7]. It scans all 64 squares by selecting the correct ADC chip and channel for each board position, then compares the raw 12-bit ADC value to the fixed baseline. A full board scan and mapping takes approximately 3–4 ms, allowing the system to update the board state quickly during gameplay. Values above the threshold are represented as white pieces, values below the negative threshold are represented as black pieces, and values within the threshold range are treated as empty squares. The function also supports both white and black board orientations by mirroring the rank and file mapping when needed. This allows the same hardware and software to generate the correct board representation regardless of which side the local player is using.

The driver also includes a diagnostic function, testADCs(), which verifies that all ADC chips are present on the I²C bus and that each channel can return a valid reading. This provides a basic hardware validation step for confirming that the sensing subsystem is connected and functioning before gameplay begins. Overall, the ADC driver forms the main software interface between the physical chessboard and the game logic by converting analog hall-effect sensor outputs into a digital board-state representation.

3.2.2 Display Driver

The display interface software manages communication between the ESP32 and the 3.5-inch touchscreen display [8]. The display is initialized in landscape mode and updated using the DFRobot graphics library over SPI. This layer provides reusable drawing functions for clearing the screen, displaying headers, rendering buttons, writing text, updating status bars, and drawing the 8 × 8 chessboard layout. These functions allow the game logic to present information to the user without directly controlling individual display commands.

During operation, the display interface shows the current board state, Wi-Fi connection status, player orientation, move prompts, timers, chat messages, and game alerts. It also supports separate screens for menus, Wi-Fi setup, password entry, AI difficulty selection, and promotion choices. By isolating display-related code into its own driver, the software remains more organized and easier to modify. This makes it possible to update the user interface without changing the lower-level game logic or networking code.

3.2.3 Wi-Fi and API Communication

The Wi-Fi and API communication software provides the connection layer between the physical chessboard and the AWS-hosted server. The ESP32 first connects to a local wireless network in station

mode using the configured SSID and password. After a successful connection, the board can begin making requests to the API over Hypertext Transfer Protocol Secure (HTTPS).

Communication with the server is handled using Wi-FiClientSecure and HTTPClient. The board uses the Amazon Root CA certificate to establish a trusted HTTPS connection before sending or receiving data. Each API request is formatted as JavaScript Object Notation (JSON) using ArduinoJson, allowing the ESP32 to package values such as board identifiers, FEN strings, moves, and request parameters into a format the server can understand.

The software supports both GET and POST requests. GET requests are used when the board needs to retrieve information from the server, while POST requests are used when the board needs to send updated information. After each request, the response body is read as a string and parsed from JSON so the rest of the embedded software can use the returned values.

3.3 API Hosting

The API for this project is hosted through AWS[9] and acts as the cloud communication layer between the physical chessboards. The block diagram for the API interface is shown below in Figure 7. The server is implemented as a Python-based AWS Lambda function and stores game data in a DynamoDB table. This allows both boards to access a shared game state without needing to communicate directly with each other. Instead, each board sends requests to the API, and the API updates or returns the current state of the game.

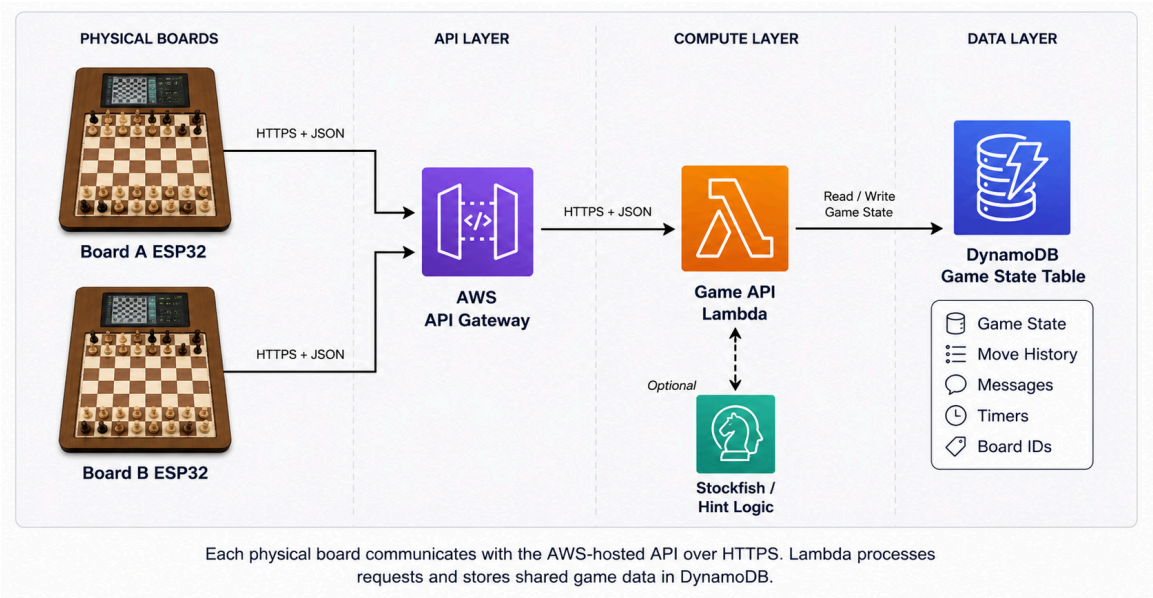


Figure 7. API block diagram

The API is organized around a single game route, /api/v1/games/1, with additional endpoints for moves, messages, reset, heartbeat, hints, and timeout events. Each request is processed by the Lambda function based on the HTTP method and path. GET requests are used to retrieve information such as the current

game state, move history, or recent messages, while POST requests are used to submit new information such as moves, chat messages, resets, heartbeat updates, hint requests, and timeout results.

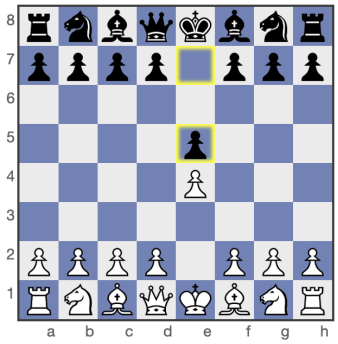
Game state is stored in DynamoDB using fields such as the current FEN, turn, version number, move history, player IDs, timer settings, messages, and game result. The API uses the board's MAC address as a board identifier, allowing the server to distinguish between the two physical boards and assign player color. Version tracking is also used when moves are submitted, helping prevent conflicting updates if two requests are made from an outdated board state.

The API also supports higher-level gameplay features beyond simple move synchronization. A messaging endpoint stores and returns recent chat messages between players. A hint endpoint uses Stockfish through python-chess to calculate a suggested move from the current FEN position. The reset endpoint initializes a new game with optional timer and AI settings, while the timeout endpoint records a game-ending clock expiration. Overall, the hosted API provides the shared backend needed for synchronization, game management, messaging, and chess engine support.

3.4 Move Validation

The move validation section is responsible for checking whether a detected board change is a legal chess move. The system stores the current confirmed board position and compares it to the new position read from the sensors. Both positions are represented using FEN strings. It is a compact text format that describes a chessboard position. Each row of the board is written from rank 8 to rank 1. FEN strings were chosen because they are compact, standard, and easy to compare. An alternative approach would have been to send all 64 board squares as individual values. FEN was simpler because the whole position could be stored and transmitted as one string. Letters represent pieces and numbers represent empty squares.

In this format, uppercase letters represent white pieces and lowercase letters represent black pieces. A number such as 8 means that the entire row is empty. In Figure 8 an example FEN string and associated board representation is seen [7]. This format is useful because the board state can be stored, compared, displayed, and sent over Wi-Fi as a short string.



rnbqkbnr/pppp1ppp/8/4p3/4P3/8/PPPP1PPP/RNBQKBNR

Figure 8. Example FEN string and matching chessboard position

To validate a move, the system compares the previous confirmed FEN string with the newly detected FEN string. From this comparison, the software determines the starting square and ending square of the moved piece. This allows the system to infer the attempted move before checking whether that move follows the rules of chess.

Once the move is detected, the system checks the chess rules for that piece. Pawns, rooks, bishops, queens, knights, and kings are each checked using their own movement rules. For rooks, bishops, and queens, the system also checks that the path between the starting square and ending square is clear. This prevents a piece from moving through another piece. The system also checks that the moving piece belongs to the local player and that the destination square does not contain a piece of the same color.

The system supports special chess moves as well. This includes castling, pawn promotion, en passant, check, checkmate, and stalemate. Castling rights are tracked during the game so the system knows whether each side can still castle. Pawn promotion is handled when a pawn reaches the opposite end of the board. En passant is checked using the previous board state and the current move.

After the piece movement rules are checked, the system verifies that the move does not leave the local king in check. This is necessary because a move can follow the movement pattern of a piece and still be illegal. If the king would be left in check, the move is rejected. If the move is legal, the system prepares the new board position for player confirmation. If the move is illegal, the system keeps the previous confirmed board position and prompts the player to fix the physical board.

The sensors detect piece presence and magnetic polarity rather than the exact identity of every piece. Because of this, the software compares the new physical board reading to the last confirmed FEN string. The physical reading shows where pieces are present. The logical FEN string stores which chess pieces those physical pieces represent. This allows the system to infer which actual chess piece moved. It also helps ignore incomplete moves while a piece is still lifted above the board. The move is only checked after the board settles into a clear new position.

3.5 Gameloop

A finite state machine was chosen because the chessboard needs to move through clear gameplay stages. The system must know whether it is waiting for setup, reading a local move, validating a move, sending a move, waiting for the opponent, or ending the game. This prevents unfinished or illegal board states from being sent to the server.

The game loop controls the overall flow of the chessboard during play. It connects sensor readings, touchscreen input, display updates, Wi-Fi communication, move validation, and game state into one organized process. Instead of performing every task at once, the system moves through a set of clear states. Each state has one main job. In Figure 9, the Finite state machine (FSM) and the specific game loop states are shown.

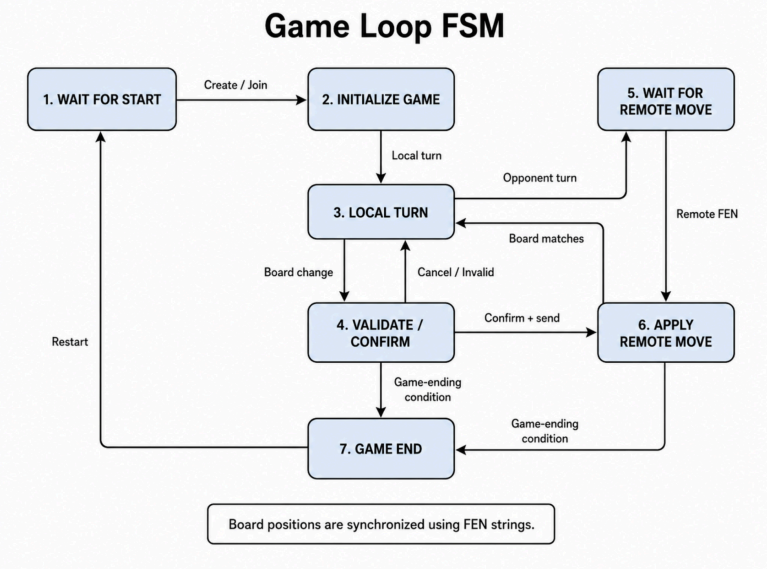


Figure 9. FSM of states the backend uses to control game flow

At the start of a game, the system initializes the board and assigns the local player as white or black. Created games begin from the standard starting position, while joined games receive the current position from the server. Board positions are sent as FEN strings, which keep synchronization simple because the full chess position fits in one compact message.

After setup, the system checks the active turn. During the local player's turn, the board reads the sensors until it detects a stable board change. The new position is then compared to the last confirmed FEN string and checked for move legality. If the move is invalid, the display asks the player to correct the board. If the move is valid, the player can confirm or cancel it. Confirmed moves are sent to the server as a new FEN string, while canceled moves return to the previous board state.

During the opponent's turn, the board waits for an updated FEN string from the server. Once received, the display shows the move to copy. The player moves the physical piece, and the system waits until the physical board matches the expected FEN position. After the match is confirmed, the local player can take the next turn.

The game loop also checks for checkmate, stalemate, and timeout. This structure keeps the physical board, local digital board, and remote server board synchronized. A move is not complete until all three match the same FEN state.

4. Design Verification

4.1 ADC Reading Verification

Accurate and responsive readings from the analog-to-digital converters were critical because the ADC subsystem directly determined whether the board could detect piece movement and reconstruct the physical board state. Delayed or incorrect ADC readings would cause missed moves, incorrect board states, or noticeable lag during gameplay.

To verify ADC performance, a test script continuously polled all eight ADCs at 30 ms intervals. Each ADC sampled one column of the chessboard, allowing the software to reconstruct the full 64-square board state. During testing, chess pieces were placed and removed across the board while the reconstructed digital board state was compared against the expected physical configuration.

The system achieved a piece detection accuracy of 98.7% under normal operating conditions. The missed detections occurred when pieces were placed too close to the edge or corner of a tile, where the magnet was not centered over the Hall-effect sensor. These errors were therefore attributed to physical placement tolerance rather than ADC communication or software failure.

The system also detected piece pickup and placement events within less than 5 ms after a board-state change. This response time was primarily limited by the full-board polling process, which took approximately 3–5 ms. These results verified that the ADC subsystem met the accuracy and response-time requirements listed in Table B.1.

4.2 API Interfacing Verification

Reliable API communication was required for the board to synchronize with a remote opponent. Without successful communication with the AWS-hosted API, the board could not retrieve the current game state, submit local moves, or maintain consistency between the two physical boards.

To verify API communication, a test function was created to send and receive game-state data over HTTPS. The test confirmed that the ESP32 could connect to Wi-Fi, establish communication with the AWS endpoint, send correctly formatted JSON payloads, and parse the server response. The transmitted data included the current FEN string, game ID, move information, and board-state version.

Testing showed that API communication completed within the 2.5 s requirement. Under normal testing conditions, Wi-Fi connection, HTTPS communication, JSON transmission, and response parsing typically completed in approximately 300–750 ms. The largest source of delay was the client-side polling rate rather than the API request itself. These results verified that the API interface met the communication and latency requirements listed in Table B.2.

5. Cost and Schedule

5.1 Cost Per Board

The total cost for a single game board breaks down into three categories: components cost, PCB cost, and labor cost. The total component cost is shown in Appendix C Table C.1 and totals an estimated $254.38 for a single game board. The total PCB cost is shown in Appendix C Table C.2 and totals an estimated $60.00, with much of the cost coming from shipping. This means the total estimated raw material cost is $314.38 for a single game board.

The labor cost is one of the most expensive parts of any consumer tech product. The equation to find the cost of labor for this project is

$$Total\ Cost\ = 2.5 \times n \times C \times T \qquad (3)$$

where n is the number of partners, C is cost in dollars per hour, and T is total time spent per partner. A reasonable starting salary for a UIUC Computer Engineering bachelor's graduate is about $77,653/year [10]. This equates to $37.33/hour for 40 hours a week and 52 weeks a year. There were three engineers on this project so n is three. The total time spent per person is an estimated 100 hours. This means that the total labor cost is an estimated $28,000.

5.2 Cost for Production

However, this could be a commercially viable product and the price of one game board will significantly decrease if quantities of raw components are increased. If production goes from one unit to 10,000 units, there is an estimated 46.7% decrease in production cost [11]. This means applying a 46.7% decrease to our original cost of one game board. Our new cost per game board is $167.36 at a quantity of 10,000 units.

If 10,000 units were intended to sell that splits the labor cost to $2.80 per unit. So, the effective cost per unit is $169.36. To have a reasonable profit margin of 30% to 50% a single unit could be priced between $220 and $260. This far exceeds what typical boards like this cost and would be accessible to intermediate and enthusiast chess players.

5.3 Schedule

The week by week schedule can be found in Appendix D Table D.1. This schedule was followed for the most part and led to the project being completed before the final demonstration. It split the work evenly, but by ability, between the three group members.

6. Conclusion

During the course of this project, a low-cost, open-source chessboard was successfully developed as a more cost-effective solution than similar products currently available on the market. The board is able to successfully detect pieces, connect to Wi-Fi, transmit and receive moves over Hypertext Transfer Protocol (HTTP) requests, and provides the user with a simple and easy to use interface. The design meets all of the core goals set forward at the beginning of our development cycle.

6.1 Hardware Conclusions

From a hardware perspective, the use of Hall-effect sensors was one of the most important design decisions for this project. Unlike reed switches, which only provide a simple on/off response, Hall-effect sensors provide analog magnetic field measurements. This allows the board to detect piece presence while leaving room for future improvements, such as exact piece identification.

The hardware system was also able to reliably read all 64 sensors, reconstruct the physical board state digitally, and support touchscreen interaction for improved user feedback. Many hardware choices were made with flexibility in mind, allowing future developers to add features primarily through software rather than requiring major hardware changes. Overall, the hardware design met the needs of the project while providing a strong foundation for future improvements.

6.2 Software Conclusions

From a software perspective, the system successfully connected the physical chessboard to the remote game state. The ESP32 firmware was able to connect to Wi-Fi, communicate with the AWS API, and transmit and receive board-state updates using HTTP requests. This allowed the board to synchronize moves between remote players.

The software was also able to reconstruct the board state from sensor readings, detect piece movement, and provide feedback through the touchscreen display. Using a modular structure made the system easier to test, debug, and expand. Overall, the software met the core requirements of the project while leaving room for future improvements such as additional game modes[12], stronger move validation, and exact piece detection.

6.3 Remaining Limitations

The largest limitation of our project is the necessity for somewhat centered piece placement. There is a significant amount of tolerance allowed but putting a piece closer to the corners impacted accurate reading of the board. This may cause issues down the line when exact piece detection is implemented. Potential fixes for this issue may include stronger magnets, mechanical guides to center pieces, and multiple sensors per tile.

Another limitation was the setup time and cost of each board. Because the design used 64 individual tile PCBs to make debugging and assembly more manageable, the project required a large amount of wiring and individual soldering. While this modular approach helped with testing and replacing individual sections, it also increased assembly time, wiring complexity, and overall cost.In a future revision, the tile PCB design could be consolidated into fewer larger boards or a single integrated sensor PCB. This would reduce wiring, simplify assembly, improve reliability, and make the system easier to reproduce at scale.

6.4 Ethical Considerations

This project follows the IEEE Code of Ethics [13] by prioritizing safety, honesty, and responsible design. The board operates at low voltage, keeps electronics enclosed, and does not collect unnecessary user data. Only chess moves and board-state information is transmitted to the API.

The system should also be presented honestly. While it supports remote physical chess, it should not be marketed as a perfect anti-cheating system or a certified competitive chessboard. Its limitations, including occasional missed detections from off-center piece placement, should be clearly stated. Economically, the lower-cost and open-source design could improve access to physical remote chess systems, while future revisions should reduce wiring and PCB waste to improve manufacturability and environmental impact.

References

[1] “ESP32 Example Board,” ECE 445 Senior Design Laboratory, University of Illinois Urbana-Champaign. [Online]. Available: https://courses.grainger.illinois.edu/ece445/wiki/#/esp32_example/index

[2] Texas Instruments, “DRV5055 Ratiometric Linear Hall Effect Sensor,” Datasheet, Jan. 2018. [Online]. Available: https://www.ti.com/lit/ds/symlink/drv5055.pdf

[3] Texas Instruments, “ADS7128 Small, 8-Channel, 12-Bit ADC With I²C Interface, GPIOs, and CRC,” Datasheet, Rev. A, May 2020. [Online]. Available: https://www.ti.com/lit/ds/symlink/ads7128.pdf

[4] DFRobot, “DFR1092 3.5-inch 480×320 IPS Screen Datasheet,” Datasheet. [Online]. Available: https://dfimg.dfrobot.com/wiki/23471/DFR1092_480-320-ips-screen_datasheet_1.0.pdf

[5] Texas Instruments, “TLV62569 2-A High Efficiency Synchronous Buck Converter in SOT Package,” Datasheet, Rev. C, Oct. 2017. [Online]. Available: https://www.ti.com/lit/ds/symlink/tlv62569.pdf

[6] Amazon Web Services, “Getting started with Amazon API Gateway,” AWS Documentation. [Online]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/getting-started.html

[7] Chess.com, “FEN,” Chess.com Terms. Accessed: May 6, 2026. [Online]. Available: https://www.chess.com/terms/fen-chess

[8] DFRobot, “DFRobot_GDL,” GitHub Repository. [Online]. Available: https://github.com/dfrobot/DFRobot_GDL

[9] Amazon Web Services, “Developing REST APIs in API Gateway,” AWS Documentation. [Online]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/rest-api-develop.html

[10] “Become an Electrical or Computer Engineer,” University of Illinois Electrical & Computer Engineering. [Online]. Available: https://ece.illinois.edu/admissions/ece-majors

[11] PCB Sync, “High Volume PCB Assembly,” PCB Sync. [Online]. Available: https://pcbsync.com/high-volume-pcb-assembly/

[12] 365Chess, “Chess Openings for White: Full Guide,” 365Chess. Accessed: May 6, 2026. [Online]. Available: https://www.365chess.com/view/chess-openings-for-white-full-guide/

[13] IEEE, “IEEE Code of Ethics,” IEEE. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html

Appendix A Abbreviations

Abbreviation	Meaning	Abbreviation	Meaning
ADC	Analog-to-digital converter	API	Application programming interface
AWS	Amazon Web Services	ESP32	Espressif 32-bit microcontroller platform
FEN	Forsyth-Edwards Notation	FSM	Finite state machine
HTTP	Hypertext Transfer Protocol	HTTPS	Hypertext Transfer Protocol Secure
I²C	Inter-Integrated Circuit	I/O	Input/output
IPS	In-plane switching	JSON	JavaScript Object Notation
LED	Light-emitting diode	MCU	Microcontroller unit
PCB	Printed circuit board	RFA	Request for Approval
SPI	Serial Peripheral Interface	TA	Teaching assistant
TLS	Transport Layer Security	TOC	Table of contents
UART	Universal asynchronous receiver-transmitter	UI	User interface
USB-C	Universal Serial Bus Type-C	Wi-Fi	Wireless Fidelity

Appendix B Requirement Verification Tables

Table B.1: Piece Detection/ADC Reading - Requirements and Verification

Requirements	**Verification**
The system shall detect chess piece presence on all 64 squares under normal operating conditions.	A test script that continuously polled all eight ADCs, with each ADC sampling one column of the board. Pieces were placed and removed across the board, and a digital reconstruction of board state was compared against the expected physical configuration.
The sensing system shall achieve at least 95% piece detection accuracy.	Testing showed a 98.7% piece detection accuracy. Missed detections were caused exclusively by pieces being placed too far from the center of a tile and notably not through ADC or software failures.
The system shall detect piece pickup and placement quickly enough for real time gameplay.	Pickup and placement events were detected in less than 5 ms after a board state change. Full board polling takes approximately 3-5ms.
The ADC interface shall correctly select the proper ADC chip and channel for each board square.	The firmware scanned the board using ADC column to chip mapping and manual channel selection. Each square was tested by placing a piece on the physical tile and confirming that the corresponding digital square changed to the correct state.

Table B.2: API Interfacing - Requirements and Verification

Requirements	**Verification**
The ESP32 shall connect to Wi-Fi and communicate with the AWS hosted API	A test function and helper website verified that the ESP32 could connect to Wi-Fi and communicate with the AWS endpoint using HTTPS requests. This was shown through the board state changes being received by the API and the FEN changing accordingly.
The API interface shall send correctly formatted game-state data.	Test requests proved that the ESP32 could send JSON payloads containing the current FEN string, gameID, move information, and board state version.
The API interface shall receive and parse game-state data from the server.	API testing confirmed that the ESP32 could receive the server response body and returned JSON for use by the embedded software.
API communication shall complete fast enough to avoid disrupting gameplay.	Testing confirmed through 2.4 Ghz Wi-Fi connection, AWS communication, JSON transmission, and response parsing completing on average between 300 - 750 ms and being mostly affected by client polling rate of the API.

Appendix C Cost Tables

Table C.1: Component Cost per Game Board

Item	Manufacturer	Part #	Qty	Unit Cost	Total Cost
Hall-effect Sensor	Texas Instruments	DRV5055A4QLPG	64	$0.5658	$36.21
USB-C 2.0 Connector	GCT	USB4125-GF-A-0190	1	$0.61	$0.61
Buck Regulator	Texas Instruments	TLV62569PDDCR	1	$0.25	$0.25
Microcontroller	Espressif Systems	ESP32-S3-WROOM-1-N8R8	1	$6.32	$6.32
8-Channel ADC	Texas Instruments	ADS7128IRTER	10[1]	$4.50	$45.00
3.5" IPS Capacitive Touch Screen	DFRobot	DFR1092	1	NA	$36.90
Magnet	Radial Magnets, Inc.	8019	32	$0.4404	$14.09
Passives and connectors[2]	NA	NA	NA	NA	$40.00*
Part Attrition[3]	NA	NA	NA	NA	$10.00*
Shipping and Taxes	NA	NA	NA	NA	$25.00*
3d Print Filament	Bambu Labs	NA	NA	NA	$10.00*
Wires and Screws	NA	NA	NA	NA	$30.00*

*Estimated pricing.

[1] Cheaper to buy in quantities of 10.

[2] Group all components that are less than $0.10 a piece (resistors, capacitors, pin headers, etc.).

[3] Pricing for replacing parts that break or get lost during assembly.

Table C.2: PCB Cost per game Board

Item	Manufacturer	Qty	Unit Cost	Total Cost
Main PCB	JLCPCB	5**	$1.00	$5.00*
Tile PCB (White)	JLCPCB	50**	$0.10	$5.00*
Tile PCB (Black)	JLCPCB	50**	$0.10	$5.00*
Shipping	NA	NA	NA	$45.00*

*Estimated pricing.

**smallest available quantity at best cost

Appendix D Schedule

Table D.1: Semester week by week schedule

Week	Dates	Team / Milestone	Payton	Danny	Quinn
1	Jan 19-25	Project selected; concept finalized	Originated chessboard concept; led sensing/network UX discussion	Proposed alternate idea; reviewed hardware feasibility	Proposed alternate idea; reviewed scope/feasibility
2	Jan 26-Feb 1	Submitted RFA with problem, solution, subsystems, success criteria	Led component research: Hall sensor, ESP32, ADC, display	Reviewed sensor/ADC options; hardware feasibility	Subsystem breakdown; display/Wi-Fi research
3	Feb 2-8	Ordered breadboard parts; replaced TLA2528 with ADS7128	Sourced parts; selected ADS7128; placed DigiKey order	Verified ADC compatibility and I²C addressability	Researched magnet strength vs. distance
4	Feb 9-15	Prepared block diagram; TA identified sensing distance as main risk	Fixed order issue; chose SPI display wiring; redesigned PCB architecture	Block diagram support; USB-C connector research	Proposal writing; display connector research
5	Feb 16-22	Proposal/schematic review	Completed main PCB schematic: ESP32, power, ADCs, USB-C	Reviewed ADC addressing, alert routing, display signals	Reviewed schematic and connector plan
6	Feb 23-Mar 1	Built breadboard; debugged I²C/soldering; found magnets too weak	Routed main PCB and submitted first fabrication order	Helped assemble/debug breadboard; researched magnets	Breadboard setup; display library research
7	Mar 2-9	Design Review and Breadboard Demo; ADC sensing demonstrated	Demo firmware: I²C polling, thresholds, LEDs; PCB BOM/parts order	Wired Hall array/LEDs; demo/debug support	Demo support; tile PCB planning
8	Mar 10-22	Spring break development	Initial ADC/LED firmware for PCB bringup	Move-checking logic from board-state diffs	Designed and ordered tile PCBs
9	Mar 23-29	First main PCB assembled by stencil/reflow	Validated PCB; fixed USB-C issue; flashed board; wrote Wi-Fi/API layer	PCB assembly, inspection, through-hole soldering	PCB validation; confirmed tile approach
10	Mar 30-Apr 5	Firmware and tile board development	Refactored firmware into modular drivers/game logic	Refined move-checking logic for integration	Designed/ordered updated tile PCBs and headers
11	Apr 6-12	UI, tiles, and mechanical supports	Cleaned interfaces; built display renderer, Wi-Fi UI, touch transform	Soldered tile PCBs and connectors	Modeled/printed tile supports; revised baseplate approach
12	Apr 13-19	First physical board assembled	Added board test mode; integrated validation; finalized ADC driver	Tile soldering; first board assembly support	Modeled baseplates/pieces; wired board; replaced faulty ADC
13	Apr 20-26	Mock demo/presentation; system integration	Integrated 12-state FSM; FEN translation; clocks, AI hints, chat	Completed board assembly; demo/debug; FEN display fixes	Mock demo/presentation; improved slide readability
14	Apr 27-May 1	Final demo and presentation	Fixed edge cases; scripted demo; validated second board; software slides	Demo sequence; hardware slides; final demo/presentation	Second board assembly; UI/enclosure slides; final presentation

Appendix E Schematics

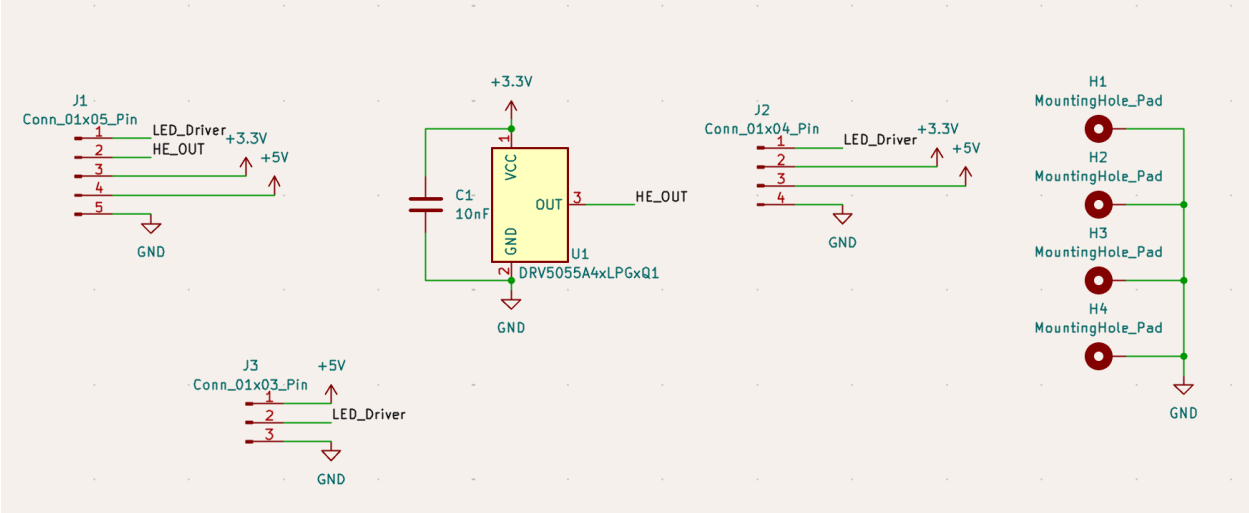


Figure E.1: Tile PCB schematic

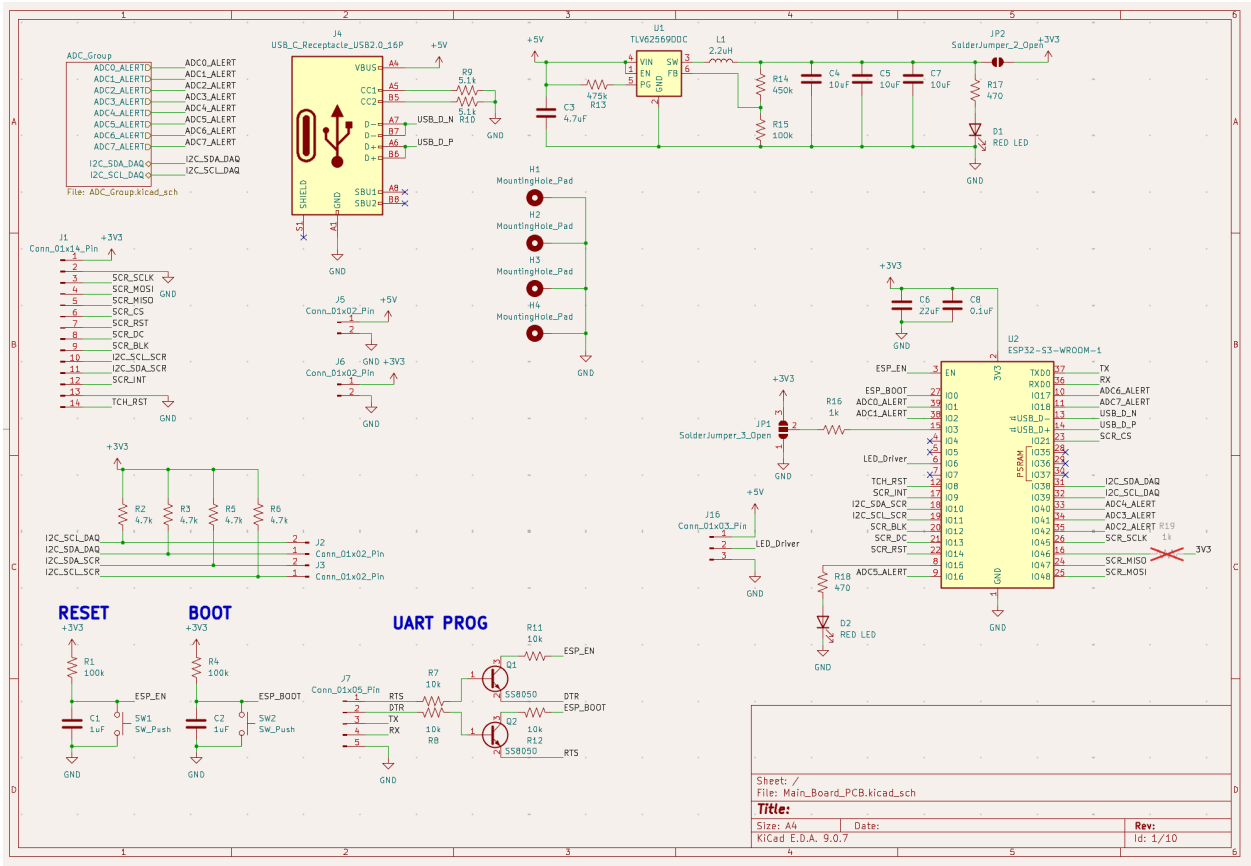


Figure E.2: Top level main PCB schematic

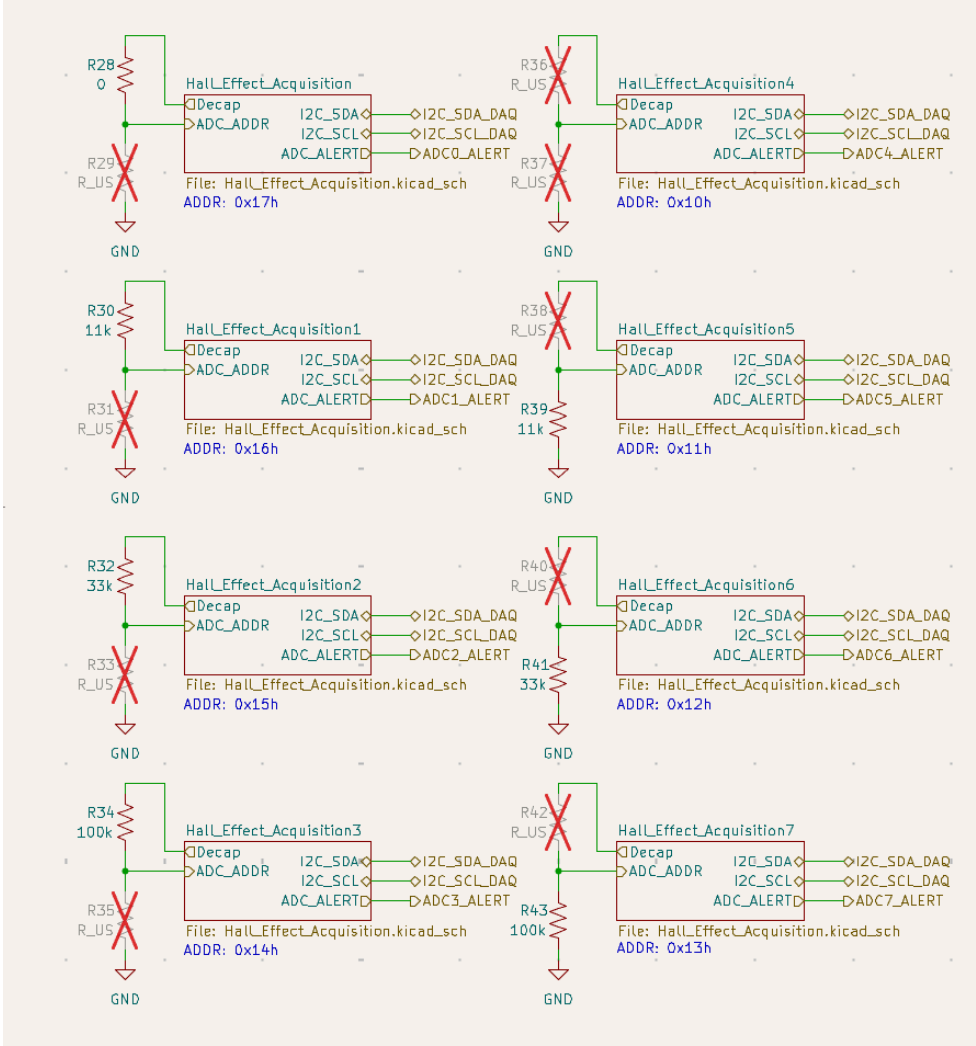


Figure E.3: ADC group main PCB hierarchical schematic

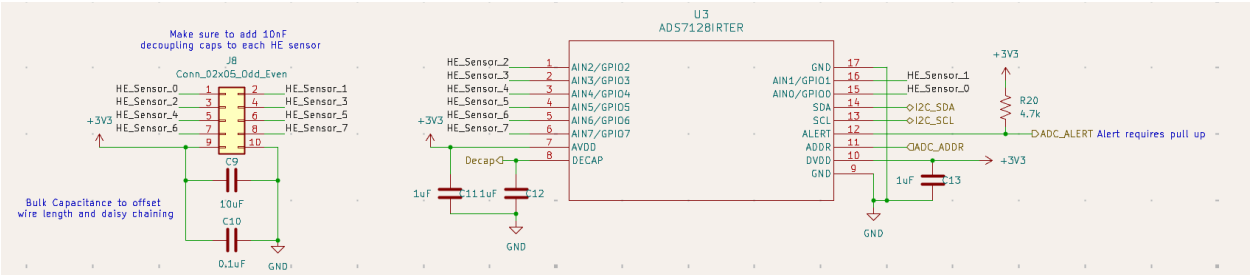


Figure E.4: Single ADC main PCB hierarchical schematic