Internet Connected Chessboard

By

Jeffrey Ito

Joel Mathews

Ritish Raje

Final Report for ECE 445, Senior Design, Spring 2019

TA: Thomas Furlong

1 May, 2019

Project No. 61

Abstract

The Internet Connected Chessboard aims to combine the capabilities of an online chess application with the physical interface of a traditional chessboard. Pieces on the chessboard are detected using reed switches. Data from the chessboard is sent to a chess application using a WiFi module. The opponent makes a move on the chess application and this data is sent back to the chessboard. LEDs light up to indicate to the chessboard user what move was made by the opponent. We were able to successfully make a move on the chessboard, send this data to the chess application, make a move on the chess application, and send this data back to the chessboard.

Contents

1. Introduction 1
2. Design 3
    2.1 Sensing/Game Board Module 3
        2.1.1 Reed Switches 3
        2.1.2 Shift Registers 4
    2.2 Control Module 5
        2.2.1 Microcontroller 5
        2.2.2 WiFi Module 6
    2.3 Power Module 7
    2.3 PC Module 7
3. Design Verification 9
    3.1 Sensing/Game Board Module 9
        3.1.1 Reed Switches 9
        3.1.2 WiFi Module 11
    3.2 Power Module 11
    3.2 PC Module 12
4. Costs 13
    4.1 Parts 13
    4.2 Labor 13
5. Conclusion 14
    5.1 Accomplishments 14
    5.2 Uncertainties 14
    5.3 Ethical Considerations 14
    5.4 Future Work 15
References 16
Appendix A - Requirement and Verification Tables 17

1. Introduction

Chess is a board game that is centuries old. In the modern age of computers, there have been many online applications created to allow players to compete against one another despite being in different parts of the world. All of these implementations, however, involve a screen of some sort. Looking at a computer screen for extended periods of time can cause fatigue in the eyes and mind, both of which are essential tools for any chess player.

To solve these problems, we plan to create a chessboard that maintains the ability to play opponents over long distances while eliminating the need for a computer screen to play the game. We plan to create an internet-connected chessboard. This chessboard would connect to a computer, but remove the need to look at a screen, as the moves would be registered on the physical board itself. Our goal is to allow players to regain the physical interface of a chessboard to reduce strain on their eyes. The chessboard will have the same functionality as online chess applications but will remove the need of the screen for the player.

Figure 1 shows the high-level block diagram from our original design. The final implemented design remains true to the original design. The only change that is made is our communication protocol between the shift registers (SRs) and the microcontroller. Instead of using an SPI interface, we use GPIO pins to control the signals to and from the SRs.

The system is split up into four main modules. The power module provides power to the whole system. The sensing/game board module contains anything that pertains to components on the physical chessboard. This includes the reed switches and LEDs which were used to detect pieces and display information on the board respectively. The control module is used to determine where data is routed. The PC module manipulates and determines what happens with the data as well as interfacing with the opponent.

Our physical design was meant to mimic the feel of a traditional chessboard while still containing all the underlying electronics. The original design can be seen in figure 2 while the final, tested design is pictured in figure 3.

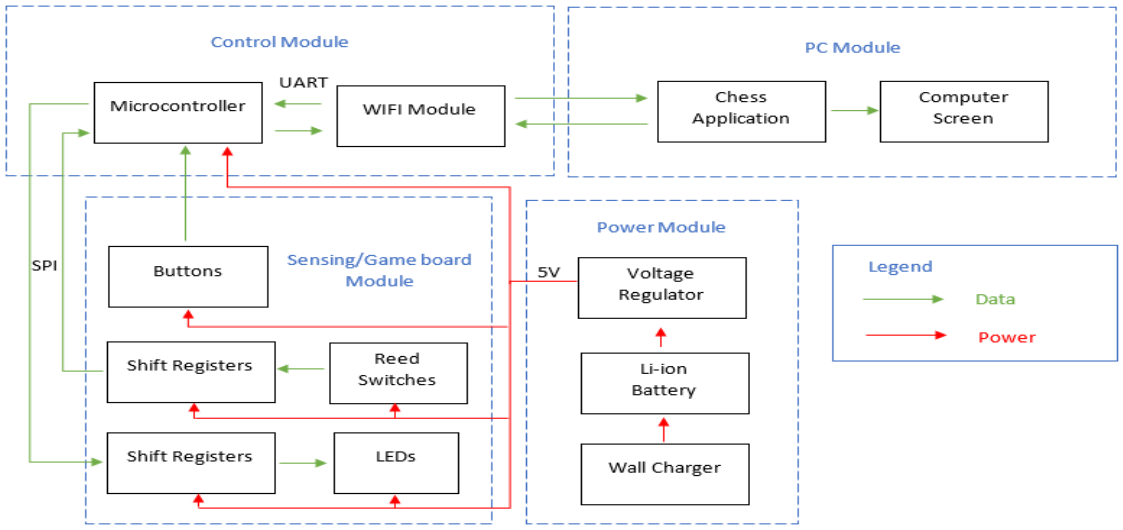

Figure 1. High-level block diagram of original design

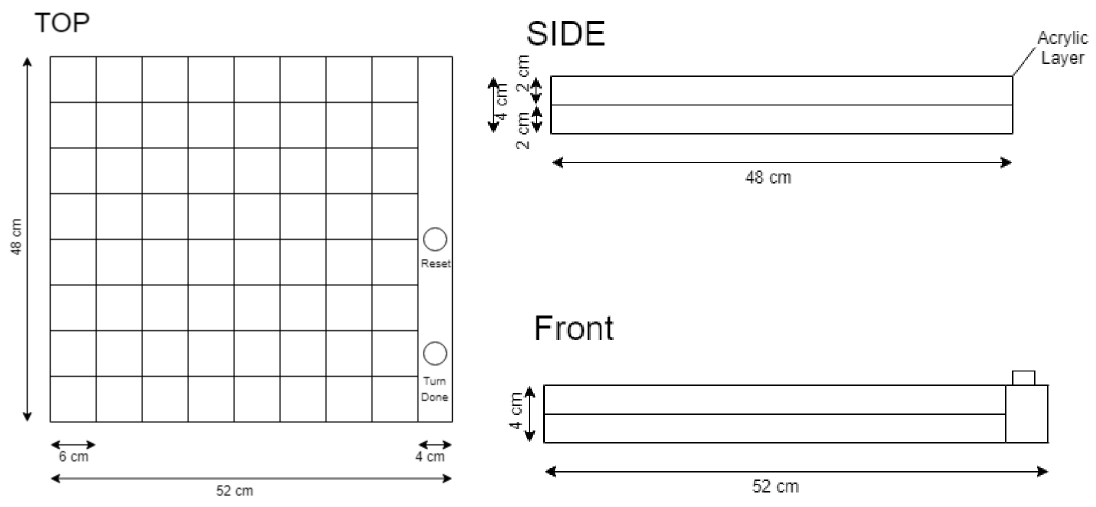

Figure 2. Original physical design of chessboard

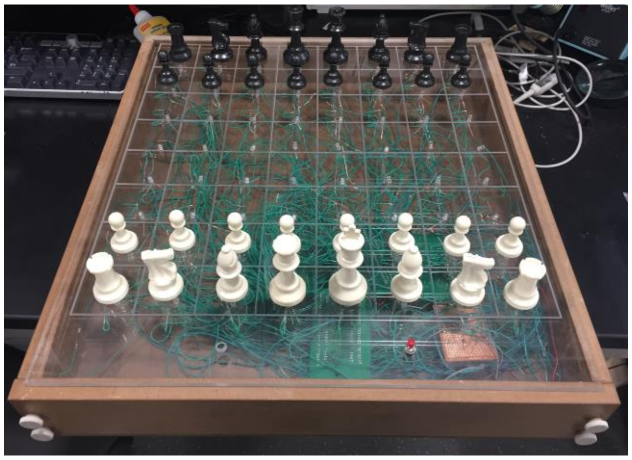

Figure 3. Final design of chessboard

2. Design

2.1 Sensing/Game Board Module

2.1.1 Reed Switches

Reed switches are magnetically activated switches [1]. This means they are like any other switch with an on/off state. However, reed switches differ because they can be activated using a magnet rather than physically flipping the switch. We decided to use reed switches to detect where pieces are on the board because of their simplicity and low cost. By handling the majority of game state tracking in the chess application, we only need a way to determine if there is a piece on a spot or not. One alternative we considered was hall-effect sensors. These are like reed switches because they change based on the presence of a magnetic field. However, these require at least 3 wires to gather data while the reed switches only require 2. Furthermore, hall-effect sensors provide more information about the magnetic field (intensity and direction) than is needed for our purposes. Another alternative we discussed was infrared sensors. These would not have required any extra hardware to be placed on the chess pieces. However, infrared sensors would also require at least 3 wires to gather data and can suffer from a lot of noise based on external lighting.

Figure 4 shows the circuit diagram we use with the reed switches to detect if a piece is on a spot. The circuit is a simple switching circuit with a pull-up resistor. The output of the circuit reads a high voltage until the reed switch is activated when it would detect a low voltage. The output of the circuit is connected to the input of the parallel-in serial-out SRs discussed in section 2.1.2.

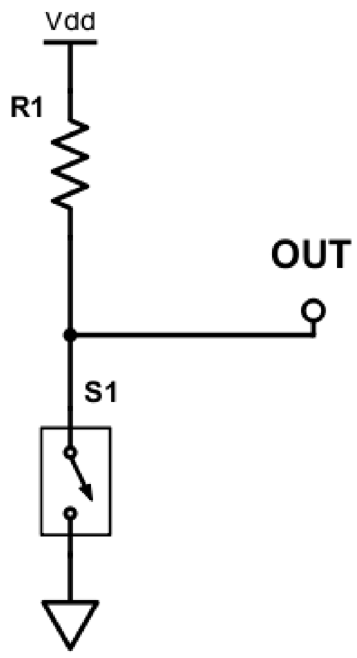

Figure 4. Circuit schematic for reed switches

2.1.2 Shift Registers

Shift registers are used to facilitate communication between the microcontroller (MCU) and the reed switches as well as facilitate communication between the MCU and the LEDs. We decided to use SRs because we had 64 reed switches and 64 LEDs. We did not have enough available pins on the MCU to connect to each one individually. Using 8 8-bit SRs for the reed switches and 8 more for the LEDs, we were able to successfully reduce the number of pins greatly.

Two different types of SRs are used for our project. Parallel-in serial-out (PISO) SRs take in parallel data and output serial data [2]. These are used to communicate with the reed switches. The output from the reed switch circuit found in figure 4 is connected to one of the inputs of the SR. When we want to gather the data from the board, a shift-in is performed on our MCU which reads the serial data from these SRs. Serial-in parallel-out (SIPO) SRs take in serial data and output parallel data [3]. These are used to communicate with the LEDs. When we want to activate an LED, we perform a shift-out on the MCU which serially transmits data to the SRs. The LEDs are connected to the outputs of the SRs and, based on if the output is high or low, the LED is activated or not. Figures 5 and 6 show how the SRs were connected in series to reduce the total number of pins required by the MCU to only 7.

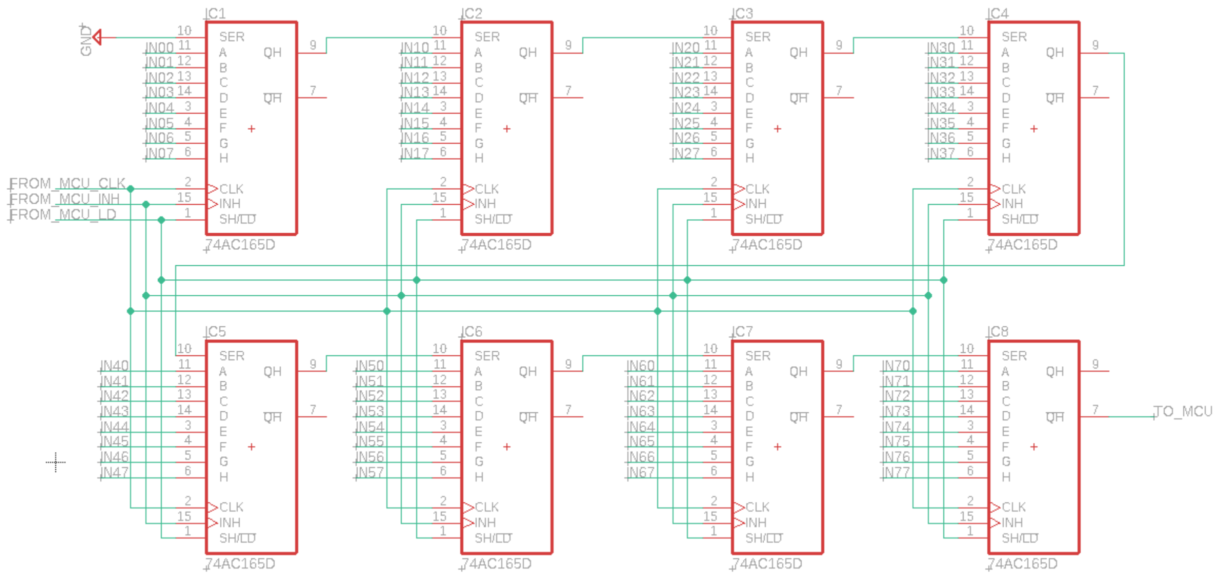
Figure 5. Parallel-in serial-out SRs connected in series

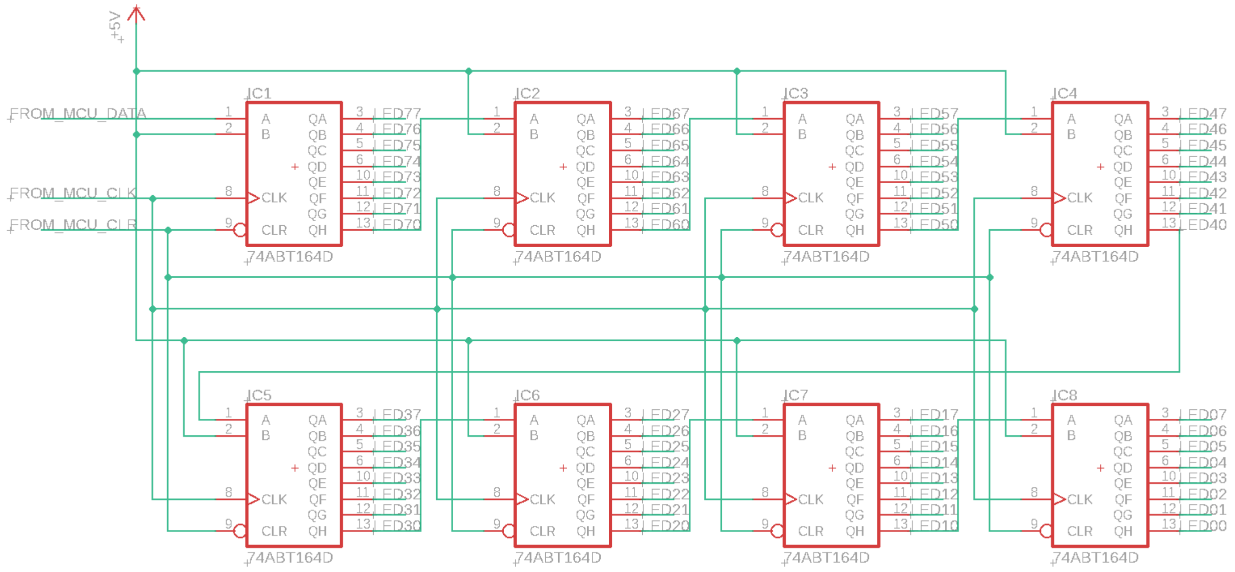
Figure 6. Serial-in parallel-out SRs connected in series

2.2 Control Module

2.2.1 Microcontroller

The MCU is used to help control the flow of data between the WiFi module and components in the sensing/game board module. We decided to go with the ATmega328P because of its UART and SPI communication capabilities, 32 kB of flash programmable memory, and ability to operate at 5 V. Furthermore, the ATmega328P is a commonly used and very well documented microcontroller. One other MCU considered was the ATmega2560 but we decided that this provided more capabilities than was necessary by our system. Another MCU considered was the PIC18F25K22. We decided that this microcontroller did not contain enough documentation and tutorials to help guide us through the project given the time frame as we were not as familiar with this MCU.

Figure 7 gives a high-level block diagram of the programming implemented on our MCU. This represents the main game loop of the player on the chessboard making a move, reflecting this move on the chess application, the opponent on the chess application making a move, and this move being reflected on the chessboard. To get the board state, we had to implement a shift-in function that updated the data on the PISO SRs from the reed switches then performed 8 shifts that shifted in 1-byte (8-bits) at a time. A similar function was implemented for shifting data out to the SIPO SRs to update the LEDs.

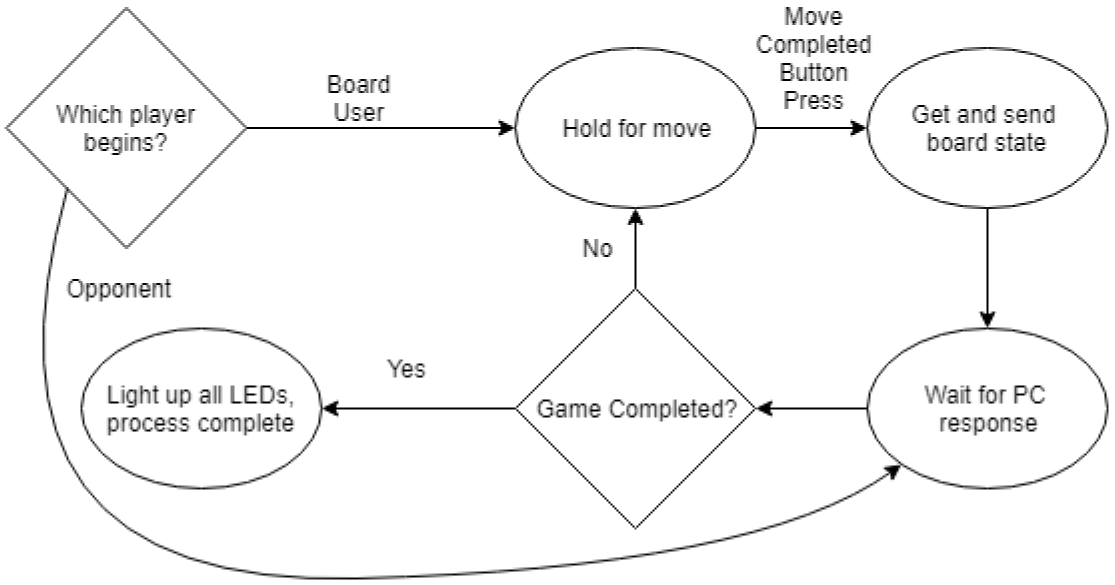

Figure 7. Block diagram of program implemented on MCU

2.2.2 WiFi Module

After researching methods of transferring data to the Chess Application, we decided the WiFi module was optimal for our project. Our top alternatives were USB and Bluetooth. However, both of those options would require an additional computer to be attached/connected to the chess board. The computer would then be responsible for relaying information across the internet. With the WiFi module, the board is self-contained, it can communicate across the globe without the need of peripheral connections.

The WiFi module was used to control the flow of data between the MCU and the online Chess Application. We decided to use the ESP8266-01s chip because it was relatively cheap ($6.99) and it had 4 Input/Output pins which suited our needs well. The MCU and the WiFi module communicated through Serial communication. The WiFi module communicated with the online Chess Application in 2 ways, when the WiFi module needed to send data, it makes a POST request to the endpoint on the Chess Application. Likewise, the WiFi module had its own server to receive data made by a POST request from the Chess Application.

2.3 Power Module

The power module provides power to our whole system. We decided to use a 9 V, rechargeable battery to provide power to our system because we did not want any external wires on the chessboard. To change this voltage to one that is useable by the rest of our system, we use a 5 V, fixed-voltage regulator. One alternative considered for our power module was to have our system connected to a wall outlet, but this would have required an external wire connected to the chessboard.

2.3 PC Module

The PC module is the brains of the whole project. It consists of solely a chess application running on the web through a python script. Our chess application does a significant part of our data processing. We could have distributed processing between the application and the microcontroller but decided not to for various reasons. Our microcontroller does almost no data processing, it continuously sends the state of the chess board. The chess application then receives data for LEDs to light up, but all the decision making is made on the python application. It is much easier to debug code on a python script than a microcontroller, for that reason, we decided to put as little code as possible on our microcontroller. Also, there is a python chess library built into python, this saved us from excessive coding, as most of the move validation was already implemented for us through this library.

The chess application functions as a state machine. It communicates with the physical chess board and serves as the interface to make moves for an opponent player. The chess application receives data through the Wifi module through a global server, called ngrok. Once it processes the data, the chess application connects to the Wifi modules server to send data back (if a move is detected), in the form of LEDs to light up. If a wrong move is detected, the applications sends a flag to the physical chess board, which signals the chess board to light up all of its LEDs

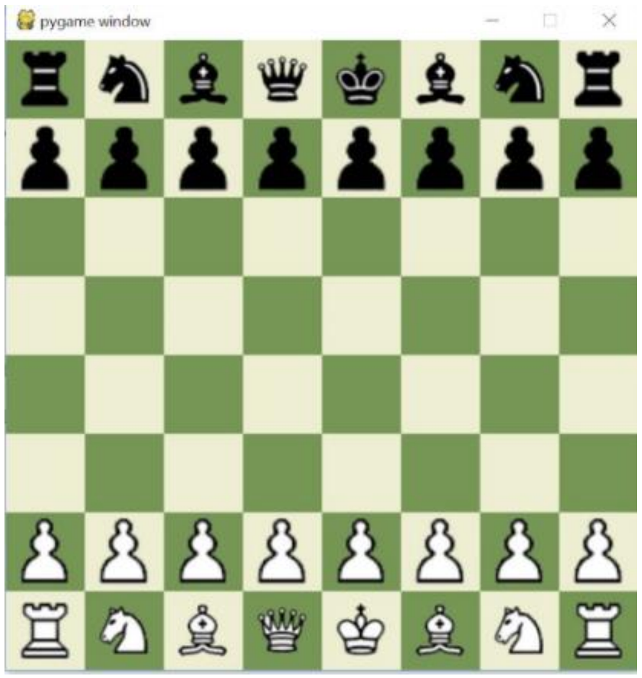

Figure 8: Chess application interface

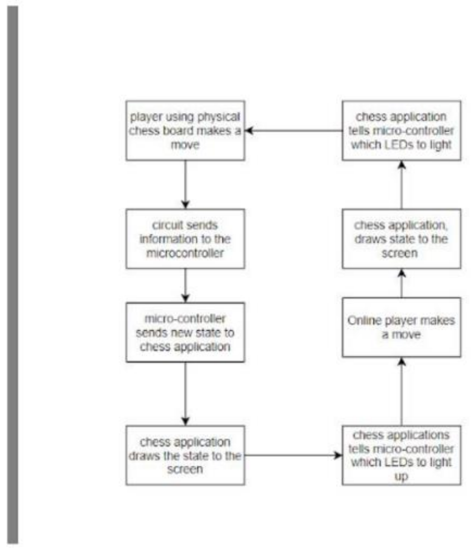
player using physical chess board makes a move
circuit sends information to the microcontroller
micro-controller sends new state to chess application
chess application draws the state to the screen
chess applications tells micro-controller which LEDs to light up
Online player makes a move
chess application, draws state to the screen
chess application tells micro-controller which LEDs to light

Figure 9: State diagram of our python application

3. Design Verification

3.1 Sensing/Game Board Module

The sensing/game board module was mostly verified through integration testing with the MCU. Two PISO and two SIPO SRs were used to verify the functionality of connecting SRs in series. For testing purposes, the inputs of the PISO SRs were either connected to power or ground to ensure that if there was a problem, it was due to the SRs and not the reed switches. LEDs were connected to the SIPO SRs as they would have been in our final product. The appropriate connections were made between our MCU and the SRs. Then, the MCU was programmed to perform a loop of shifting in data from the PISO SRs then shifting this data out to the SIPO SRs, so whatever data was input to the PISO SRs was immediately visible on the LEDs. Figure 10 shows the setup on a breadboard. Outlined in yellow is the first PISO SR connected to an input sequence of 01001010. Outlined in blue is the first SIPO SR and associated LEDs which show the same bit sequence by looking at the right side going from bottom to top then looking at the left side going from bottom to top. Similarly, outlined in red is the second PISO SR connected to an input sequence of 01101100 with the associated SIPO SR and LEDs outlined in green.

The SRs were verified to be able to take parallel and serial input data and output serial and parallel data. SRs were verified to be able to convert 16-bits of input into data that can be read by the MCU, but this was easily extended out to 64-bits of input. The MCU was verified to be able to send and receive data to and from the SR. This verified all the requirements for the SRs and all but 1 of the requirements for the MCU from our original design (see Appendix A).

3.1.1 Reed Switches

The reed switches are required to activate from a distance that is far enough that the layer of acrylic is not too thick to prevent them from activating. The reed switches must also not activate from the effects of a magnet on an adjacent square. To verify that this was true, several test cases were taken. Figure 11 shows various configurations as well as angles which were tested to activate the reed switch. Table 1 shows the distance at which the reed switch was activated for the various test cases. The distance was measured from the center of the reed switch to the closest edge of the magnet. As can be seen by the test results, the orientation, and angle that the reed switch approaches both affect the activation distance. We decided to go with orientation 2 because it provided a good range of activation distances that was larger than the thickness of our acrylic (0.64 cm) but smaller than the distance to the edge of a square (2.93 cm).

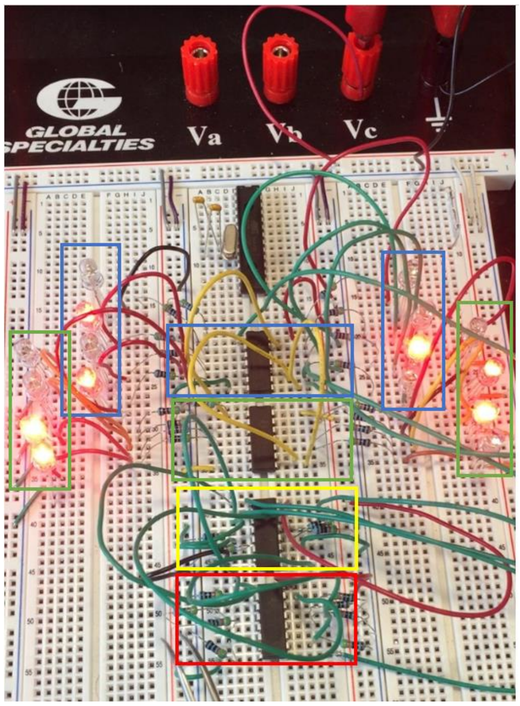

Figure 10. Setup used to verify SRs, MCU, and LEDs

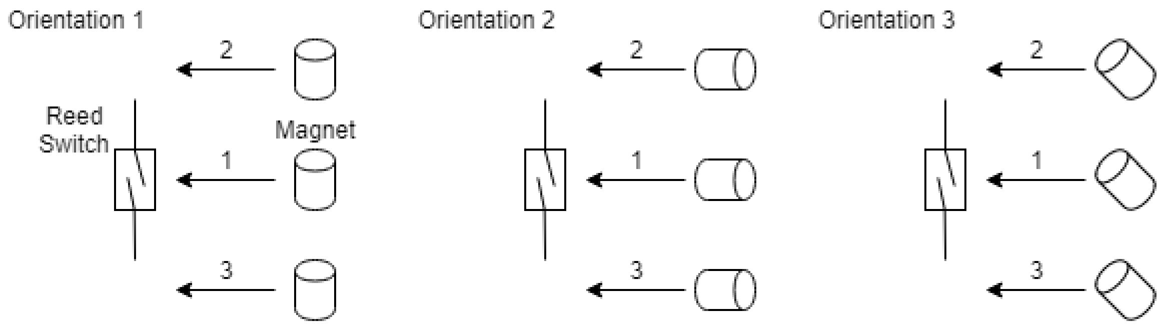

Figure 11. Test orientations and angle of approach for magnets with reed switches

Table 1. Activation distance of reed switch based on orientation and angle of approach

Orientation	Angle of Approach	Activation Distance (cm)
Orientation 1	1	0.74
	2	2.54
	3	0.71
Orientation 2	1	0.65

	2	2.13
	3	0.69
Orientation 3	1	2.78
	2	0.24
	3	1.12

3.1.2 WiFi Module

It was essential for our Wifi module to successfully send and receive data. To verify sending information, we connected our module to the server made by our python script. If the python script could print out the correct message sent by the module, we knew the message was sent correctly. Verifying received data was done in a similar way. Our python application connected to the server made by the ESP, and sent a message, we used Serial write to determine whether the sent message was correct.

3.2 Power Module

The power module was only required to provide 5 V for our system to utilize. To ensure this was accomplished, the battery needed to output a voltage greater than 5 V and the voltage regulator needed to output a voltage at 5 V ± 5%. This was verified by fully charging our 9 V battery and measuring the output voltage by connecting it to a multimeter. The output observed on the multimeter was at 8.3 V which was more than enough. The output on the voltage regulator was also measured with the multimeter and verified to be at 5 V ± 5%. It should be noted that batteries are known to output less voltage as the power is depleted. While we were unable to test our batteries due to time constraints, similar batteries are known to output above 5 V even after many hours of use as seen in Figure 12.

Another power analysis we were unable to perform in the given time frame was how long our battery would last given the power consumption of our system. However, through observation and estimation we can determine an approximate time. The component that draws most of the current for our system is the WiFi module, which draws approximately 80 mA while active. Taking all the other components into account, the total current draw of our system may be approximately 100 mA. Our battery is rated at 600 mAh meaning we can estimate about 6 hours of operation for our system from a fully charged battery.

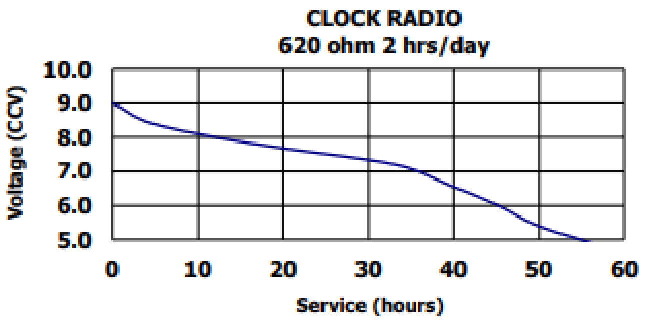

Figure 12. Voltage output of Energizer 9 V battery for usage in clock radio [4]

3.2 PC Module

The PC module needed to correctly detect and validate moves on the chess board. We decided to verify move validation manually instead of coding our own test cases. It would be much faster to manually test out if every piece moved correctly. Our test cases did not have to be very comprehensive as the python chess library already took care of move validation. We basically tested whether each piece moves the correct way and if turns alternate, but this testing was done with two players locally playing on the chess application (i.e. the physical chess board wasn’t connected). The full unit tests for each type of piece are given in the appendix r and v table for chess application.

Our chess application needed to get and receive the correct information. To test receiving information, we unit tested each reed sensor and checked the output our MCU sent us. Fortunately, all the reed sensors which were working properly sent the application the right data. We experienced a few bugs sending data and could not properly unit test it. The chess application sent the MCU which LEDs to light up whenever it needed to send data. Sometimes this worked and sometimes this didn’t. We could not figure out whether this was a bug in the python script, or the code running on the MCU. Although, we do think that it was a bug in the MCU code, as we could print out data right before it was sent by the python application, and it appeared to be right.

4. Costs

4.1 Parts

Table 2. Parts cost

Part	**Manufacturer**	**Retail Cost ($)**	**Bulk Purchase Cost($)**	**Actual Cost ($)**
2x 9 V Li-ion rechargeable batteries + charger	EBL	16.99	N/A	16.99
ESP8266-01 WiFi Module	Espressif Systems	6.99	N/A	6.99
64x reed switches	Cylewet	48.93	N/A	48.93
32x neodymium magnets	McMaster-Carr	37.44	27.84	27.84
64x RGB LEDs	EDGELEC	8.99	N/A	8.99
8x PISO SRs	Texas Instruments	3.76	3.18	3.18
8x SIPO SRs	Texas Instruments	3.52	2.86	2.86
ATmega328P	Atmel	2.14	N/A	2.14
2x Buttons	E-Switch	1.85	N/A	1.85
PCB	PCB Way	4.00	N/A	4.00
Total				**123.77**

4.2 Labor

Labor costs contributed to most of the cost of the project. Assuming an ideal wage of $40/hr, working 10 hrs/week, and 16 weeks in the course we get a total labor cost of $48,000 for 3 partners as shown in equation (1).

$$3 * \$40/hr * 10\ hrs/week * 16\ weeks * 2.5 = \$48000 \tag{1}$$

5. Conclusion

5.1 Accomplishments

We were successfully able to make a move on our chessboard, send this data to the chess application, and display the move on the chess application. We were also successfully able to make a move on the chess application, send this data back to the chessboard, and display the information on the chessboard.

5.2 Uncertainties

One of the biggest uncertainties with our current implementation is the latency. It takes approximately 5-10 seconds for a move made on the chessboard to be reflected on the chess application and a similar time to send the data back to the chessboard. This is not conducive for any chess game that needs to be played at a particular pace. Latency also causes our chess application to sometimes miss a detection of moves made on the chessboard.

The other major uncertainty of our current implementation is we were not able to completely connect all the reed switches and LEDs to test a complete game. We were able to make one successful round of moves as described in section 5.1. However, due to the wiring complexity that was associated with the current implementation, there was too much clutter that could be reasonably organized in the given time frame. In addition, many wires were openly exposed which caused them to short out against one another.

One final uncertainty of our project was the power consumption. As mentioned in section 3.2, we were not able to perform an exact analysis on the power consumption of our system. Making exact measurements of the current draw from all the components of our system then testing our battery at that current draw would give us an exact amount of time that the battery would last. A more in-depth power analysis would also give insight into the exact voltage that the battery would drop down to when it is not fully charged, which would tell us when we need to indicate that the battery needs to be charged.

5.3 Ethical Considerations

Our internet connected chess board is limited by the sensors on board. For this reason, there are not many ethical concerns for our project. If we were to continue our project we would want to make a dedicated website for our users. This website might be prone to attacks by hackers, which would violate section 1.6 of ACM's code of ethics to respect privacy. We would also have to worry people cheating through the manipulation of software. This would violate section 1.4 of the ACM code of ethics, to be fair and take action not to discriminate [5].

5.4 Future Work

One thing that can be done to build upon our project is to redesign the wiring system. The current wiring complexity cost too much time to put together and organize. In addition, there were many open wires that would short each other out. A ground plate could be attached to the acrylic so fewer wires need to be run from the reed switches and LEDs all the way to PCBs. Furthermore, a ribbon cable could be used to run wires from the reed switches and LEDs to the PCB which would make the wiring much more organized as well as reducing the number of openly exposed wires.

Another thing that could be done to improve on our project would be to reduce latency and fine-tune software algorithms. As mentioned in section 5.2 the latency of our system can cause several issues. We believe the main cause of latency is the way we implemented our WiFi endpoint in software. Looking into better solutions could reduce latency significantly as well as ensure a more reliable transfer of data. Fine tuning our software algorithms could also help with latency. Currently, both our software algorithms implemented on the MCU and in Python both utilize delays during data transfer. Eliminating these delays could help improve the latency issue and improve the reliability of data.

One final thing that could be done to improve our project would be to reduce the power consumption. The WiFi module was the component that was drawing the largest amount of power from our system. Currently, it is always active. Utilizing the built-in sleep modes for the WiFi module while it is not in use could help to reduce the power consumption and increase the battery life of our system.

References

[1] HSI Sensing, "Reed Switch Basics," [Online]. Available: https://www.hsisensing.com/wp-content/uploads/2016/03/HSI_Sensing_-_Reed_Switch_Basics_v100512.pdf. [Accessed 28 April 2019].

[2] Texas Instruments, "SNx4HC165 8-Bit Parallel-Load Shift Registers," December 2015. [Online]. Available: http://www.ti.com/lit/ds/symlink/sn74hc165.pdf. [Accessed 29 April 2019].

[3] Texas Instruments, "SNx4HC164 8-Bit Parallel-Out Serial Shift Registers," September 2015. [Online]. Available: http://www.ti.com/lit/ds/symlink/sn74hc164.pdf. [Accessed 29 April 2019].

[4] Energizer, "ENERGIZER 522," [Online]. Available: http://data.energizer.com/pdfs/522.pdf. [Accessed 26 Apri 2019].

[5] ACM, "ACM Code of Ethics and Professional Conduct," 2018. [Online]. Available: https://www.acm.org/code-of-ethics. [Accessed 29 April 2019].

Appendix A - Requirement and Verification Tables

MCU

Table 3. MCU R&V

Requirements	Verification	Verification Status
1. Can transmit data to PC through USB 2. Can receive data over SPI from shift registers 3. Can send data over SPI to shift registers	1. A. Form a packet of data that contains the bits 10010011. B. Transmit this packet to the PC and ensure that the received packet matches the sent packet. 2. A. Load the shift registers with the bit sequence 10010011. B. Perform a shift-in on the MCU and ensure the received data is correct 3. A. Perform a shift-out on the MCU with the bit sequence 10010011. B. With a DMM, probe each parallel-out pin to ensure that the bit sequence is correct.	Y Y Y

WiFi Module

Table 4. WiFi Module R&V

Requirements	Verification	Verification Status
1. Can facilitate data transmission between the MCU and the connected PC	1. A. Form a packet of data that contains the bits 10010011. B. Transmit this packet to the PC and ensure that the received packet matches the sent packet.	Y

Reed Switches

Table 5. Reed Switches R&V

Requirements	Verification	Verification Status
Activate from the effects of a magnet that is 0-3 cm away.	A. Connect read switch to test circuit, placing the switch at the 0 cm mark on a ruler. B. Slowly move a magnet closer to the switch, starting at the 10 cm mark. C. Make note of the voltage across the switch every .5 cm. D. Ensure that the measured voltage is 5 V ± 5% until the magnet is 3 cm away, at which point the voltage should drop to 0 V.	Y

Shift Registers

Table 6. Shift Registers R&V

Requirements	Verification	Verification Status
1. Allows for parallel input, serial output. 2. Allows for serial input, and parallel output. 3. Convert 64 reed switch readings to digital data for the MCU to process.	1. A. Connect each parallel input pin to a GPIO pin on the microcontroller, setting each pin so the bit sequence is 10010011. B. Connect the serial output pin to the serial input of the microcontroller. C. Send a load signal from the MCU to the shift register. D. Perform a serial shift in on the MCU ensuring that the bit sequence read was 10010011. 2. A. Connect the serial output pin of the MCU to the serial input pin of the register.	Y Y

	B. Perform a serial shift out on the MCU with the bit sequence 10010011. C. Probe each of the parallel output pins with a DMM ensuring that the final bit sequence is 10010011. 3. A. Connect reed switch outputs to parallel input pins on SRs. B. Send a load signal to the SRs. C. Perform serial shifts in on the MCU ensuring that the bit sequence read matches which switches were activated when the load signal was sent.	 Y

Voltage Regulator

Table 7. Voltage Regulator R&V

Requirements	Verification	Verification Status
Converts DC input to 5 V ± 5% output.	A. Connect input of voltage regulator to a power supply. B. Power on the supply and adjust it to provide 12 V ± 5%. C. Measure output with DMM to ensure it remains steady at 5 V ± 5%.	Y

Battery

Table 8. Battery R&V

Requirements	Verification	Verification Status
Outputs greater than 5 V DC.	A. Recharge battery to full. B. Connect terminals to DMM and measure the output voltage ensuring it is greater than 5 V.	Y

Chess Application

Table 9. Chess Application R&V

Requirements	Verification	Verification Status

1. Chess application receives the correct information regarding the game state every time it is changed. 2. Chess application sends the correct data over the cloud	1. Change the game state. Print out the packet of information right before it is sent to the cloud 2. Connect a second computer to the cloud. Print out the data it receives from the sender. -All pieces are able to move the way they are defined to move, invalid moves are not possible -list of all possible checks are not written out because they are too extensive, but a summary is given below on everything checked: -horse moves correctly -queen moves correctly -rook moves correctly -pawn moves correctly -pawn promoted to queen correctly -white and black alternate turns -checkmate is detected -stalemate is detected	Y N – data displayed on the chessboard does not always match up with data being sent by application
3. Chess Application only allows valid moves	-All pieces are able to move the way they are defined to move, invalid moves are not possible -list of all possible checks are not written out because they are too extensive, but a summary is given below on everything checked: -horse moves correctly -queen moves correctly -rook moves correctly -pawn moves correctly -pawn promoted to queen correctly -checkmate is detected -stalemate is detected -white and black turn alternate	Y