Assistive Shogi

Team 48 — Rahul Rameshbabu and Maxim Papushin
TA: David Hanley

Spring 2019
# Table of Contents

**Introduction**  
Objective  3  
Background  3  
High-level requirements  4  

**Design**  
Physical Design Description  7  
Tolerance Analysis  9  
Functional Overview  11  
Requirements and Verification  13  

**Cost And Schedule**  
Cost  17  
Schedule  19  

**Ethics And Safety**  
20  

**References**  
21
Introduction

Objective

Shogi is a very difficult game for beginners to learn because of its complex set of rules. While anybody could find a rulebook and start trying to play shogi, it would be very simple to violate one of its rules, or not to be able to take advantage of some more complex rules. There are a number of guides meant for learning the game, and one can attempt to play online shogi. However, cross-referencing a long guide to verify that every move is correct gets considerably tedious, and playing online shogi first requires the player to at least be moderately capable to play against real opponents. The complexity of the game remains overwhelming due to the sheer variety of pieces, each with their own moveset, compounded with added complexity from the ability to replace captured pieces.

To rectify this issue, we propose a shogi board which helps to interactively teach a player all the valid moves that can be made using a piece that has been lifted from the board. This will help players know all possibilities of where to move the pieces and prevent any illegal moves from being made, as the board will keep track of its own game state, and thus be able to point out any errors to the players.

Background

While most Americans are at least vaguely familiar with the rules of chess, if not able to fully play it, far fewer are familiar with shogi. Whereas chess has an established community of players and mentors willing to teach the game, shogi has no such following, with few organized groups, if any. Thus, prospective players are left to either play unbalanced games online against far more experienced players, to try learning alone through a guide, or to start learning with other beginners, where neither player has enough experience to notice when a rule has been violated.

Shogi has a 9x9 board over chess’s 8x8, but the more significant difference is shogi has a heavier emphasis on counter attacking strategies. Unlike in chess, pieces in shogi are never out of play, and may be replaced on the board by the capturer, creating an immense amount of possible moves compared to chess. Finally, shogi has some obscure rules which beginners may easily forget about, such as optional promotion, except under certain conditions where it becomes mandatory. Even though shogi and chess pieces share similar names, they behave quite differently, which makes adapting from chess to shogi quite difficult [1].

Chess has quite an astonishing number of people playing the game, 35 million in the US [2]. This community helps support both high-level competition and learning players. However, shogi does not have such a strong community, especially outside East Asia. While it is difficult
to get an accurate count of the number of shogi players in the United States, an estimate can be made by comparing the number of players well-known enough to have articles on Wikipedia, which would suggest roughly 100 times as many chess players (315) as shogi players (3) [3][4].

By making the process of learning the game simpler, its popularity could rise significantly.

High-level requirements

- Board must recognize when a piece has been lifted and identify the possible moves for that piece.
- Board must keep a log of valid moves which have been made, which should be exportable to the connected computer.
- Board must display a warning if an invalid move is made, which needs to be corrected before play can continue.
- Connected computer must allow for manual error correction both for physical errors such as pieces moving after bumping the board, as well as manually inputting a move if the board misreads what has been played.
Design

Figure 1: Block diagram for implementation of shogi board

Legend
- Sensor Data
- Digital I/O
- Power over USB
- Serial Communication over USB

Figure 1: Block diagram for implementation of shogi board
The Game State/Compute Processing (GS/CP) subsystem that handles maintaining the game state for each move made and making the master logical decisions behind which LEDs should be illuminated and processing the sensor I/O. This unit is responsible for back and forth communication with the AtMega 328P in the Game I/O block. The computer device provides additional information to the players.

The Game I/O subsystem is responsible for handling I/O between the board electronics of the LEDs and photoresistors, as well as for sending data over serial to the computer device in the GS/CP subsystem. The PCB with the AtMega is powered over USB from the computer and serves as a voltage source for the other low-power components in the project, such as ICs and LEDs.

The Game Board Sensor subsystem provides information about what pieces have been moved with the use of photoresistors. The sensors will require some additional logic to convert resistance into a digital voltage and will include some MUXes to reduce the number of I/O pins required, which require some control lines back from the AtMega chips.

The Game Board Indicators subsystem states what moves are valid to make with the piece currently selected. They are controlled by signals from the AtMega chips in the Game I/O Processing unit. The indicators will store some internal state logic in order to allow all LEDs to be individually controlled with a minimal number of I/O pins, likely in the form of shift registers and individual comparators.
Physical Design Description

The board will be modified to include photoresistors underneath each of the 81 squares on the board. For each square, a hole will be drilled through the center of the piece, and photoresistors will be attached into the hole, facing upwards. When a piece is present on that square, the photoresistor is covered; without a piece, the photoresistor is lit by the ambient light of the room. This difference in lighting will be used to detect when a piece is moved.

In addition to the photoresistors, each square will have two LEDs at opposite corners of the square. These LEDs will be allowed to protrude slightly from the top of the board in order to prevent them from being covered up when a piece is placed on the same tile.

All the electronics will be connected underneath the board, with the only externally accessible parts being a connection for the power supply, and a USB connection between the AtMega to the computer.

Figure 2: Photoresistor placement, side view

This simple render is intended to show a cross-section of one tile of the board, with a piece present on this square. The photoresistors are slightly inset into the surface of the board, such that the pieces smoothly sit on the board, blocking the ambient light to the photoresistor.
At any time during the game, the pieces being present will block certain photoresistors and reveal others. Each of these is being monitored by the Arduino chips and translated into a game state. Once a change is detected in one of the photoresistors, the system will determine whether the move which the player has made is indeed valid, and respond accordingly.
Tolerance Analysis

The sensors used for detecting whether a piece is present, photoresistors, or light-dependent resistors (LDRs), are nonlinear devices which vary in resistance by many orders of magnitude with a change in lighting. Using one example of a datasheet taken from an arbitrarily selected photoresistor, the resistance in darkness, such as one of the photoresistors which is being covered by a piece, is in the range of 5MΩ, while the ‘bright’ resistance is in the range of 50kΩ at 10 Lux [8], which is still well below ordinary ambient room lighting, at roughly 50 Lux [9].

Because of this wide variation in resistance, the tolerance for individual differences between photoresistors is quite large, as anything within an order of magnitude should be easily distinguishable. A simple analog-digital converter should be sufficient for this application, or even a plain voltage divider.

The most difficult part of the project construction will likely be the circuitry needed to read from the photoresistors. Since there are 81 inputs, the amount of wiring needed to connect each photoresistor to an input on the PCB will be extensive, and will likely cause errors. Furthermore, the amount of inputs needed are more than a single AtMega chip is able to support. As such, it will be necessary to add input multiplexers in order to reduce the number of inputs to a manageable amount, which will in turn add additional complexity to the wiring.

Similarly, each square on the board will have two LEDs, which will also have to be controlled individually by the AtMega chip. In order to avoid having to use so many output lines, it may be necessary to create simple LED controller circuits with latches to save a state, so that they can be programmed with fewer data output lines, instead of creating a 81-wide one-hot bus to control LEDs, which would further complicate wiring.

The circuit must be designed in such a way that only one AtMega chip is necessary. This will not only reduce the cost of the entire board, but also simplify the code needed for the AtMega to communicate with the computer. According to the AtMega 328P datasheet, each chip has 23 programmable I/O pins [5], which would have to be shared between the input photoresistors and the output LEDs.

The physical construction of the board is not expected to be difficult, as a relatively large margin of error is acceptable for the photoresistors to fit in the holes while still being entirely covered by the pieces. This is not expected to require any more precision than what could be achieved with a simple drill press. However, depending on what sort of board is used for the design, some effort may be needed to fit electronics into a relatively small space, or otherwise to lift the board up with some supports in order to create more space underneath the board.
Figure 4 shows a simple circuit should be sufficient to convert the LDR’s resistance into a voltage which can be read by the microprocessor chips. Using the above example, with a resistance range of 50kΩ to 5MΩ, we take R1 to be 500kΩ.

\[
\begin{align*}
I_{out} = 0 \quad &\rightarrow \quad I_Rp = I_R1 \\
V_{out} / R_p = (5 - V_{out}) / R_1 \\
V_{out} / R_p + V_{out} / R_1 = 5 / R_1 \\
V_{out} \cdot (1 / R_p + 1 / R_1) = 5 / R_1 \\
V_{out} = 5 / (R_1 / R_p + 1)
\end{align*}
\]

Case 1: No piece present, bright light, Rp = 50kΩ
\[
V_{out} = 5 / (500k / 50k + 1) = 5 / 11 = 0.4545V
\]

Case 2: Piece present, dark light, Rp = 5MΩ
\[
V_{out} = 5 / (500k / 5M + 1) = 5 / 1.1 = 4.545V
\]

Equation 1: Photoresistor voltage divider calculations

As seen in Equation 1 from the two values for Vout based on the resistance of the photoresistor, the two voltages should be easily distinguishable as digital low/high, though some testing may be necessary to determine the ideal resistance for R1 for the particular photoresistors which end up being used in the design.
Functional Overview

**Game State/Compute Processing** - The goal of the Game State/Compute Processing subsystem is to receive process input from the AtMega 328P microcontroller in the Game I/O Processing block and also process any board state correctional data provided by the user to the program running on the computer, in the case of pieces being bumped accidentally. Once it receives game state updates, the program updates its UI and sends information to the AtMega on which LEDs should be updated to display valid moves that can be made.

**Game I/O Processing** - The Game I/O Processing subsystem's goal is to handle the large number of sensory inputs, and digital indicator outputs needs for the assistive shogi board. The AtMega microprocessor uses the sensor data to detect when pieces are moved, and controls LED illumination based on the commands it receives from the computer, but does not know about the setup of the game being played.

The AtMega communicates with the computer in the Game State/Compute Processing subsystem to update the computer program with information about piece movements that lets the program update the game state. Any detection of a lifted piece should be sent over to the computer, which sends signals in response to the AtMega indicating which LEDs should be illuminated, thus indicating to the players which spaces are valid moves given the piece which was lifted from the board.

**Game Board Indicators** - This block consists of LEDs used to indicate when each position is valid to move to on the 9x9 shogi board. These indicators are controlled by the AtMega, which is in turn signalled to set certain LEDs to light up by the computer.

In order to save power, the LEDs will be blinking at high frequencies, rather than always being on. This will require some logic internal to each LED, but should allow each of the AtMega chips to draw considerably less power to power peripherals, thus reducing the power requirements of the entire board.

**Game Board Sensors** - This block contains a series of photoresistors placed at each position on the 9x9 Shogi board. These sensors each contain a simple voltage divider to calculate an output voltage based on the internal resistance of the photoresistor. The sensors are connected to a series of MUXes within the Game I/O Processing subsystem, which pass the sensor data to the AtMega microcontroller.
## Requirements and Verification

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Verification</th>
</tr>
</thead>
</table>
| Must continuously store the game state (piece positions and their validity), including both pieces on the board and whether a piece is currently being moved. | 1. Run the game program on the computer while connected to the board.  
2. While making several moves, use debug features in the program to dump the game state between each action (lifting or placing a piece).  
3. Verify that the state output correctly reflects the physical state of the board. |
| Program must allow the user to save or load a game state to/from a file on the computer. | 1. Run the game program on the computer while connected to the board.  
2. Save the game state to a file using the program.  
3. Make several valid moves in the game.  
4. Use the program to load the previous state.  
5. Move pieces on the board as indicated by the program to restore the previous state.  
6. Dump the game state to verify that the layout matches the physical layout of the board, and the layout at the time of the save. |
| Program must allow for captured pieces to be selected and replaced on the board during play. | 1. Run the game program on the computer while connected to the board.  
2. Load a game state with a captured piece.  
3. Use the program to select which piece to play, then physically play the piece.  
4. Dump the game state to verify that the state matches the physical layout with the played piece on the board. |
| Must detect invalid moves made during play and inform the user, requiring correction before allowing play to continue. | 1. Run the game program on the computer while connected to the board.  
2. Pick up a piece, then move it to an invalid tile for that piece.  
3. Verify that the program displays a warning indicating the invalid move.  
4. Move the piece to a valid location from its initial position.  
5. Verify that the error goes away and play is resumed. |
Table 2: R&V for Game I/O Processing Subsystem

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Verification</th>
</tr>
</thead>
</table>
| Must detect a piece being lifted or placed at any one of 81 spaces within 2.5 seconds of the move. | 1. Modify the computer program to output debug information to a console whenever a piece movement is received.  
2. Perform several piece movements:  
   a. Lift or place a piece, and simultaneously start a stopwatch  
   b. Verify that the debug message appears within 2.5 seconds.  
   c. Verify that the debug message contains the correct piece location, and the correct direction (placed or lifted) |

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Verification</th>
</tr>
</thead>
</table>
| Must act on any LED state commands to update the board LEDs within 0.5 seconds of the command being sent. | 1. Modify the computer program to send arbitrary LED state commands to the AtMega.  
2. Use the program to send a pattern of LEDs to the AtMega, and simultaneously start a stopwatch.  
3. Verify that within 0.5 seconds, the LEDs on the board match the state which was sent in the command. |

Table 5: R&V for Game Board Indicators Subsystem

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Verification</th>
</tr>
</thead>
</table>
| Each of 81 squares must be individually set as either green, red, or off, independent of all other squares. | 1. Connect LED control signals to a standalone Arduino board with equivalent chip.  
2. Run a program which provides control signals to the LEDs in order to draw some pattern on the board.  
3. Verify that the LEDs match intended patterns. |

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Verification</th>
</tr>
</thead>
</table>
| Total average power consumption for all LEDs on board should not exceed 250mW for any configuration of enabled LEDs. | 1. Connect GND and VCC to a metered DC power supply as 0V and +5V.  
2. Connect LED control signals to a standalone Arduino board with equivalent chip.  
3. Run a program which provides control signals to the LEDs in order to set all tiles to green.  
4. Verify that power usage recorded on power supply does not exceed required maximum value. |
Table 6: R&V for Game Board Sensors Subsystem

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor output voltage records greater than 4V when a piece is present on</td>
<td>1. Connect the Vout from one photoresistor on the board to a voltmeter.</td>
</tr>
<tr>
<td>the tile, and less than 1V when no piece is present on the tile.</td>
<td>2. Connect the GND and VCC to a DC power supply as 0V and +5V.</td>
</tr>
<tr>
<td></td>
<td>3. Record the voltage on the Vout wire when a piece is present at the center of the tile. Repeat when no piece is present.</td>
</tr>
<tr>
<td></td>
<td>4. Verify the voltage in both cases is past the threshold.</td>
</tr>
<tr>
<td>Sensor output can be accurately recorded as digital high (1) when a piece</td>
<td>1. Connect the Vout from one photoresistor to a standalone Arduino board.</td>
</tr>
<tr>
<td>is present on the tile, and digital low (0) when no piece is present on the</td>
<td>2. Run a program which reads the sensor input on the connected pin and outputs it to the console as either 1 or 0.</td>
</tr>
<tr>
<td>tile.</td>
<td>3. Verify that console output matches the actual state of the piece on the board.</td>
</tr>
<tr>
<td>Digital value of a sensor is correctly updated within 2 seconds of a piece</td>
<td>1. Connect the Vout from one photoresistor to a standalone Arduino board.</td>
</tr>
<tr>
<td>being either placed or removed.</td>
<td>2. Run a program which reads the sensor input on the connected pin and outputs it to the console as either 1 or 0.</td>
</tr>
<tr>
<td></td>
<td>3. Place a piece on an empty tile, and run a timer at the same time the piece is placed.</td>
</tr>
<tr>
<td></td>
<td>4. Verify that the console output becomes a steady 1 within 2 seconds.</td>
</tr>
<tr>
<td></td>
<td>5. Repeat steps 3-4, removing a piece instead of placing it.</td>
</tr>
</tbody>
</table>
Cost And Schedule

Cost

Labor Analysis

Labor Cost = Average Electrical Engineer Salary ($33) x 2.5 multiplier for some safe tolerance for going over estimate x Estimated Amount of Hours Worked

Rahul - $33 x 2.5 x 44 hrs = $3630
Max - $33 x 2.5 x 42 hrs = $3465
Total - $7095
### Parts Cost

*Table 8: Cost Estimate of Required Parts*

<table>
<thead>
<tr>
<th>Part Name</th>
<th>Quantity</th>
<th>Unit Cost</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATMega 328P</td>
<td>1</td>
<td>$4.99</td>
<td>$4.99</td>
</tr>
<tr>
<td>Photoresistors (100 pieces)</td>
<td>1</td>
<td>$8.95</td>
<td>$8.95</td>
</tr>
<tr>
<td>FT232RL USB to Serial Cable</td>
<td>1</td>
<td>$7.99</td>
<td>$7.99</td>
</tr>
<tr>
<td>16 MHz Clock Crystal</td>
<td>1</td>
<td>$7.99 per 20</td>
<td>$7.99</td>
</tr>
<tr>
<td>Red+Green LEDs</td>
<td>81 each</td>
<td>$6.68 per 50</td>
<td>$13.36</td>
</tr>
<tr>
<td>74LS00 NAND</td>
<td>42</td>
<td>$1.09 per unit</td>
<td>$45.78</td>
</tr>
<tr>
<td>74LS02 NOR</td>
<td>41</td>
<td>$0.54 per unit</td>
<td>$22.14</td>
</tr>
<tr>
<td>74LS04 Inverter</td>
<td>41</td>
<td>$0.61 per unit</td>
<td>$25.01</td>
</tr>
<tr>
<td>74LS06 O-C Inverter</td>
<td>41</td>
<td>$0.73 per unit</td>
<td>$29.93</td>
</tr>
<tr>
<td>74LS85 Comparator</td>
<td>18</td>
<td>$1.27 per unit</td>
<td>$22.86</td>
</tr>
<tr>
<td>74LS191 Synchronous Counter</td>
<td>2</td>
<td>$1.09 per unit</td>
<td>$2.18</td>
</tr>
<tr>
<td>73LS279 Quad S-R Latch</td>
<td>41</td>
<td>$1.07 per unit</td>
<td>$43.87</td>
</tr>
<tr>
<td>Perf Boards</td>
<td>40</td>
<td>$6.49 per 20</td>
<td>$12.98</td>
</tr>
<tr>
<td><strong>Total parts cost</strong></td>
<td></td>
<td></td>
<td><strong>$247.85</strong></td>
</tr>
</tbody>
</table>

Some parts listed in Table 8 require purchasing relatively large quantities of parts even when only a single item is required, since the item is not sold in smaller quantities.

These IC quantities are based on a certain simple logic layout - It may be possible to optimize the logic to require fewer IC chips than this, such as by combining multiple adjacent tiles into one circuit to share latches between adjacent tiles. This is likely to be revised after testing the circuit logic.
**Bulk Cost Analysis**

Resistors and capacitors are trivial in cost when mass purchased in bulk in terms of cents per unit so we will neglect them from the calculations. Let's assume per IC used we can estimate around 50 cents and PCB fabrication to be 3 dollars per unit. Let's assume we can produce the boards and pieces for 10 dollars in bulk estimating the cost for milling. This would lead to a bulk cost of $3 PCB + $10 Board + ($0.10 an IC x 226 ICs) = $35.60

**Total Cost Analysis for the Project**

The total cost is $7095 in labor plus $247.85 in parts, which leads to a total cost of $7342.85. The vast majority of the cost is dominated by labor cost, while the parts are relatively inexpensive. If the individuals working on this project were actually getting paid, the cost would increase dramatically over the current real cost, which is only parts.
## Schedule

Table 9: Predicted Schedule for Project Work

<table>
<thead>
<tr>
<th>Week</th>
<th>Tasks Completed (Time Taken) - Rahul</th>
<th>Tasks Completed (Time Taken) - Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/18</td>
<td>Revise Design Document - (3 hrs)</td>
<td>Revise Design Document - (3 hrs)</td>
</tr>
<tr>
<td></td>
<td>Begin ordering basic components for simple prototyping - (1 hour)</td>
<td>Begin ordering basic components for simple prototyping - (1 hour)</td>
</tr>
<tr>
<td>2/25</td>
<td>Begin prototyping and debugging basic components of the design - (5 hours)</td>
<td>Begin prototyping and debugging basic components of the design - (5 hours)</td>
</tr>
<tr>
<td>3/4</td>
<td>Begin PCB designs - (6 hrs)</td>
<td>Construct Perf circuit design for voltage dividers - (4 hrs)</td>
</tr>
<tr>
<td>3/11</td>
<td>Finalize PCB design - (3 hrs)</td>
<td>Initial software implementation working - (3 hrs)</td>
</tr>
<tr>
<td>3/18</td>
<td>Construct physical board prototype - (4 hrs)</td>
<td>Begin working on touchscreen interface - (4 hrs)</td>
</tr>
<tr>
<td></td>
<td>(Break)</td>
<td></td>
</tr>
<tr>
<td>3/25</td>
<td>Integrate pieces together and begin unified testing - (8 hrs)</td>
<td>Integrate pieces together and begin unified testing - (8 hrs)</td>
</tr>
<tr>
<td>4/1</td>
<td>Debug and make needed design revisions - (8 hrs)</td>
<td>Debug and make needed design revisions - (8 hrs)</td>
</tr>
<tr>
<td>4/8</td>
<td>Work on presentation material - (6 hrs)</td>
<td>Work on presentation material - (6 hrs)</td>
</tr>
<tr>
<td>4/15</td>
<td>(Nothing currently scheduled for this week)</td>
<td>(Nothing currently scheduled for this week)</td>
</tr>
</tbody>
</table>
Ethics And Safety

Although the board will be designed to record the game which is being played by keeping track of the moves made, all game data is made available only to the players themselves, and are not distributed or saved if the players themselves do not want to keep said data.

If this product were to be marketed, some experimentation would be necessary to determine the how effective the board is to assisting learning, however this is far outside the scope of this class, and is not a consideration at this time.

Our project operates under a low voltage environment, minimizing many electric safety concerns, though not removing them completely.

One possible issue is the issue of undervoltage in the case of large loads [6]. We need to be careful when developing the PSU to be sure we accurately accounted for the power consumption needs of the entire design. If not, we might damage our hardware over time due to operating electronics at inconsistent voltages. If we did so, we would be delivering a product that is not capable of sustaining itself, which would be unacceptable as a consumer product.

We should also be very of the case of using incorrect or unregulated power supplies that may result in an overvoltage and thus burn or damage the ICs and other electronic components used in this project. However, aside from power supply concerns, this project operates using mostly low voltage systems and does not take in an special input or sensitive data, reducing the amount of ethical and safety concern significantly. Even though our project itself does not seem to violate any ethical or safety concerns, we as developers should be sure to follow a code such as the IEEE Code of Ethics [7].
References


