Final Project: JanPongšević

Thomas George, Sophia Klymchuk, Eugene Wang, Aidan Wong
Prof. Janjusevic
ECE 150: Digital Logic Design

December 24, 2025

1 Introduction

The objective of this project is to design and implement a digital system with meaningful user interaction and memory functionality, comparable in scope to the Animated Drawing Canvas assignment. For this project, we chose the design-your-own option and implemented a two-ball Pong game that satisfies the functional and hardware constraints of the assignment. The game is displayed on an Light Emitting Diode (LED) array, uses user inputs to control paddles, and maintains internal state information, such as ball positions, movement directions, and collision outcomes, using combinatorial logic and memory elements.

This report first describes the overall design approach and decisions made in the Design and Methods section. The Implementation section then details the completed circuit, including input/output interfaces and physical realization. Finally, the Conclusion summarizes the results, discusses limitations of the current design, and outlines potential extensions of the system in more complex digital applications.

2 Design and Methods

2.1 System Decomposition and Functional Blocks

The project utilizes sub-circuits that can be set, triggered, and cleared to form a modular and scalable design. The user interface (UI) consists of a start button to begin the game, a 2x5 array of 5x5 LED matrices forming a total resolution of 10 x 25, with a used resolution of 10 x 17, and two de-bounced buttons per side that shift each respective side's Pong paddles up or down by one pixel. The functional block diagram consists of:

1. A master clock which is subdivided to create slower frequency clocks hereby referred to as "game clocks."

2. A ball coordinate handler block that takes in a game clock signal, a RESET (RST) signal to restore balls to their default position, and outputs 4-bit x and y positions for two balls.

3. Paddle blocks which take in an RST signal and debounced button signals, and outputs directly to the LED matrix.

4. A ball display handler block that takes in ball positions from the ball coordinate handler block and outputs HI and LO signals to the rows and columns of the LED matrix.

5. A collision logic block which takes in information from the ball display handler block and the paddle to output an RST signal.

6. A score handler block which takes in an RST signal and information from the ball coordinate handler block to determine which side has scored at an RST signal.

7. A ball acceleration block which records how many bounces of the balls have occurred and changes the signal of a multiplexer (MUX) to change which game clock is selected.

8. The 10 x 25 LED matrix which takes in information from the paddle blocks and the ball display handler blocks to provide visual feedback to the user.

Due to the complexity of the circuit, abstraction was a necessary step for the circuit to be implemented in a comprehensive manner, thus, much of the following description of the circuit will be describing the cross sub circuit interactions, then developing the sub circuit internals.

2.2 Functional Block Diagram

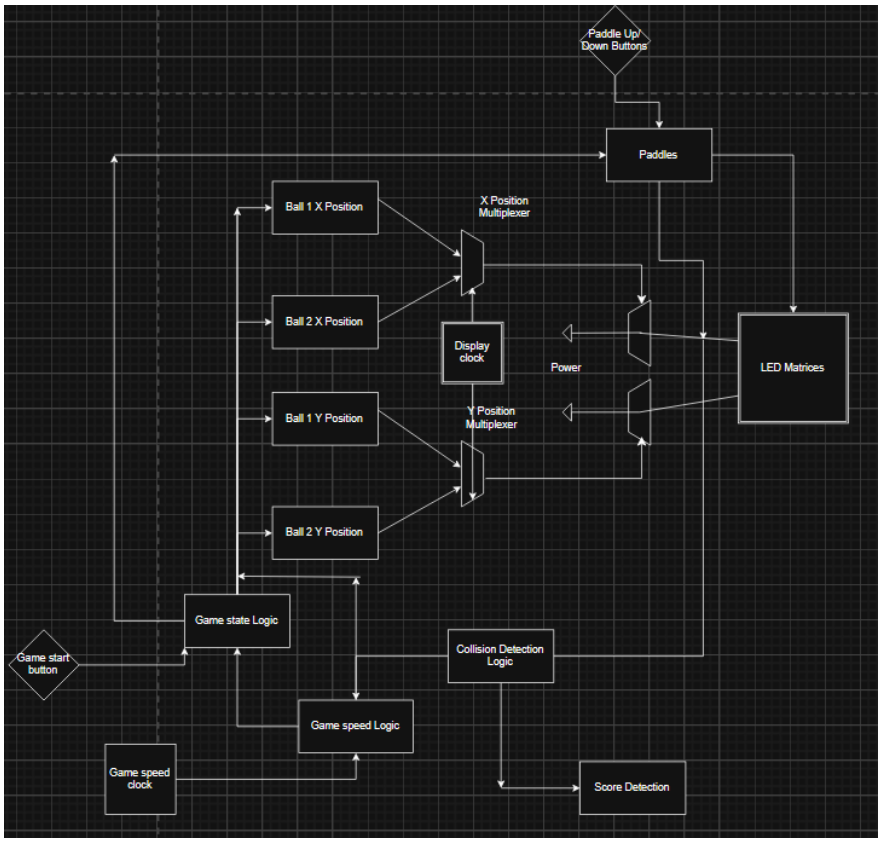

Figure 1: Functional block diagram of the Pong game.

Figure 1 displays the functional block diagram for the overall circuit. Each of the two balls in this pong circuit are represented by two counters, with each holding an X and a Y position for each ball. Each Ball's X position is fed into a multiplexer for display, where each ball is displayed one at a time, each controlled by the display clock. The display clock runs at a high speed so as to create the illusion of persistence of vision, which makes it seem that both balls are being displayed at the same time.

The game state logic is an initialization handler that initializes both the balls and the paddles to their starting positions on the LED matrix, which is triggered every time a point is scored.

The game speed logic is something that manages the speed of the game (balls and paddles); it operates at a much slower speed than the display clock.

Something of note is that each de-multiplexer that feeds into the LED matrix has power running directly to the input, with each position counter feeding into the signal pin inputs of the de-multiplexer. What this accomplishes is that there is a power signal (or a ground due to the technicalities of the LED matrix), sent only to the position of the ball, with the de-multiplexers controlling where the ball is at. The reason a power signal is the input is that a ball is always being displayed, thus, the de-multiplexer should never be disabled.

The collision detection is in sync with the LED matrix so that the visual information communicated to the users corroborates the logic being processed.

2.3 Ball Positions

To reduce the complexity of the ball collision physics, the balls always reverse their x and y directions when they reach the end of the playable ball area, meaning that the balls do not take into account paddle positions to determine if the ball bounces. Instead, the ball is always moving, and the presence of the ball at the end of the playable ball area without the paddle present at the same y position triggers an RST signal to interrupt the rest of the circuit, and to initiate a score. This results in the ball coordinate handler block not requiring any information from the paddle blocks. Additionally, the balls do not interact with each other.

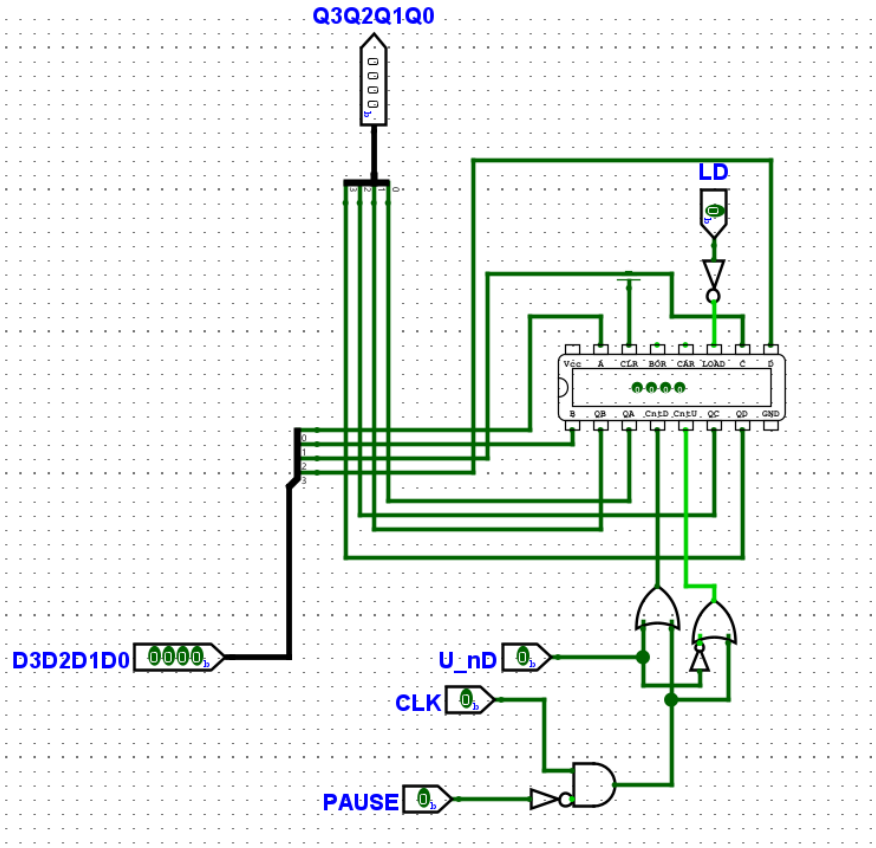

Figure 2: Load counter sub-circuit.

The ball positions in the circuit are described using a sub circuit known as "Load Counter", —

"load" denoting that the counter can be set asynchronously to a user defined input— which utilizes a SN74LS193 Synchronous Up / Down counter IC chip. The sub-circuit's logic diagram is shown in Figure 2.

Pause ends up being unutilized in implementation so it will not be discussed. The values $D_3D_2D_1D_0$ and $Q_3Q_2Q_1Q_0$ are the respective inputs and outputs to the load counter. Some unique behavior of note is that the counter works by having either the "CntU" or "CntD" pin being clocked while the conjugate pin is held at a high value (For example, if the "CntU" pin is being held high, and the "CntD" pin is being clocked, the clock will clock down, otherwise the behavior is undefined).

The "LD" pin controls whether or not there are values being loaded into the counter, and the "U_nD" ($U/\overline{D}$) pin makes for a more convenient interface to interact with the IC chip (that being, if the chip is being clocked, and the "U_nD" pin is held high, it will be counting upwards, the opposite holds true for when the chip is being clocked and the pin is held low).

2.3.1 Ball Counter : X Position

The below figure describes the external logic that interacts with the load counter so as to form the X position counter.

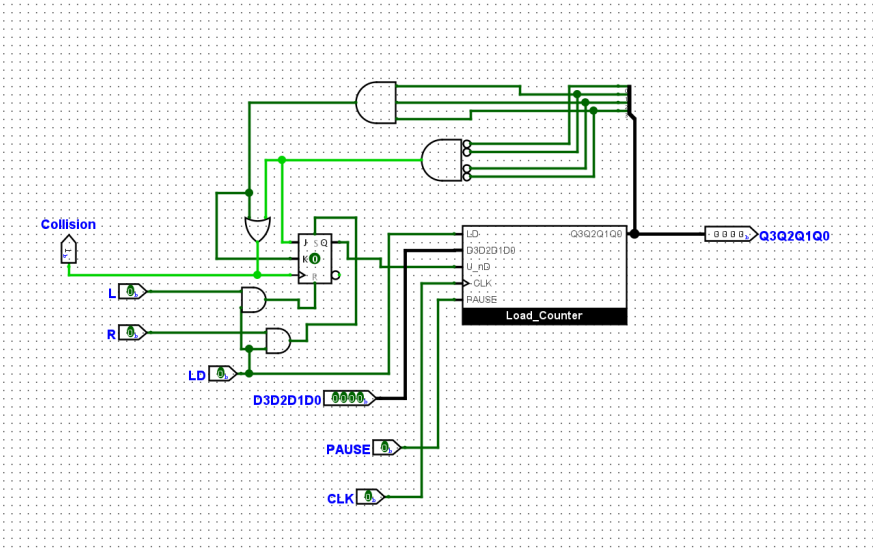

Figure 3: X-position ball counter.

Whenever either of the large AND gates at the top of the figure are triggered (when the counter reaches either the value $Q_3Q_2Q_1Q_0 = 0000$ or $Q_3Q_2Q_1Q_0 = 1110$ (either 0 or 14 in binary respectively), the counter will "bounce" or flip from counting downwards to upwards or upwards to downwards respectively. Both of them feed into an OR gate, which measures whether or not a collision has transpired, which is relevant for external Logic. The load pin allows for asynchronous setting of the initial position and direction of motion of the counter when triggered, with the direction of motion being determined either by the "L" or "R" input pins which are user defined.

2.3.2 Ball Counter : Y Position

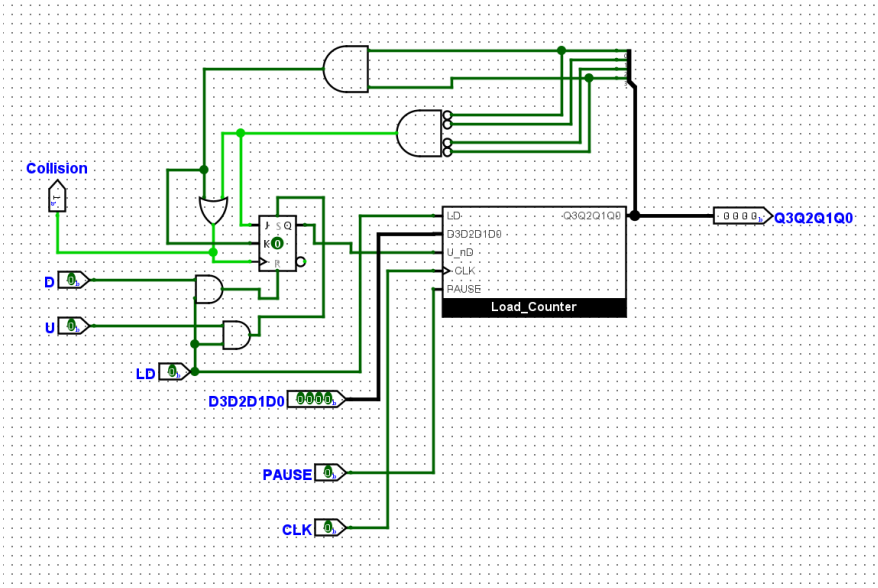

Figure 4: Y-position ball counter.

The Y Position counter uses very similar logic, with only the bounce points and naming conventions being changed. "R" and "L" are changed to "U" and "D" respectively as this counter controls whether or not the ball is traveling upwards or downwards as opposed to right or left, and the counter bounces on the value $Q_3Q_2Q_1Q_0 = 1001$ or 9 in binary instead of 14 as the LED matrix only has 10 rows as opposed to the 15 columns.

2.4 Ball Display Handler Block

To allow for both balls to be shown, the project utilizes the illusion of persistence of vision. The 4-bit x and y coordinates of each ball are input into a MUX, and a discrete 1kHz clock is used to switch between the 4-bit x and y coordinates of the first ball and the second ball, and only one ball's coordinates are ever input into the ball display handling circuit. By switching between the two balls at a high frequency, the eye perceives both balls as being displayed at the same time.

2.5 Paddle Positions

Each paddle was represented as a line of 3 LEDS (out of 10) on the display matrix, for which the simplest method of tracking its position was to use a "line" of bidirectional universal shift registers (BSRs).

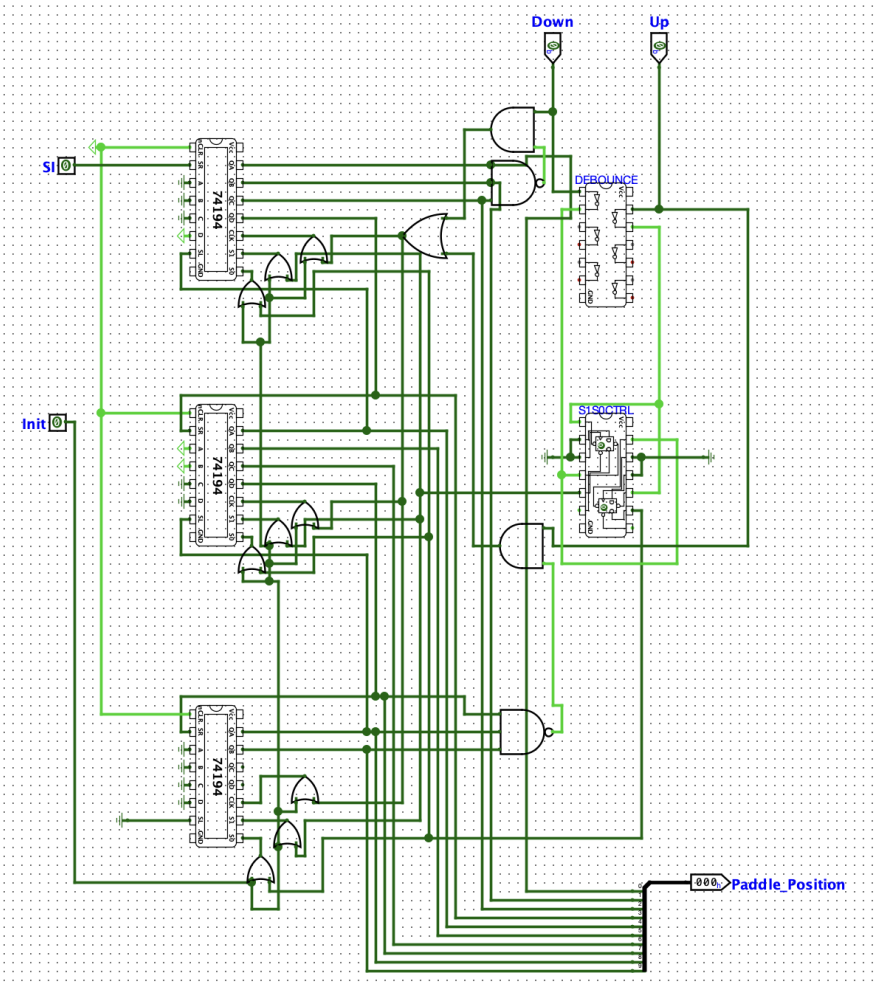

Figure 5: Paddle sub-circuit.

Figure 5 displays the simulated sub-circuit for a single paddle. The labeled "Up" and "Down" inputs represent de-bounced button pulses (not shown on Logisim since we de-bounce them utilizing an RC filter), which are then fed into Schmitt triggers to re-sharpen their signal edges. The "Up" button signal is then fed into $1\overline{CLR}$ and $2\overline{PRE}$ whereas the "Down" button signal is fed into $2\overline{CLR}$ and $1\overline{PRE}$ for a SN7474 dual D flip-flop chip, such that the two outputs are asynchronously set and reset by the button presses, and that each output is set/reset opposite of the other. As a result, the output of the first D flip-flop is 1 when the paddle is moving down (with the other flip-flop held low). and the output of the second D flip-flop is 1 when the paddle moves upwards (with the first flip-flop held low). The clock and input D pins are never utilized and are instead grounded.

The down and up flip-flop outputs are fed into the $S1$ and $S0$ input pins of our BSRs (SN74194) in order to shift them left and right, respectively.

Because the paddles move around in a line of 10 bits, three 4-bit BSRs are connected such that the serial feed-left and feed-right pins between each register are connected to the first and last outputs, respectively. As only positions 0-9 are populated, the final two outputs are simply never read.

The paddle is initialized as a line of three 1s at positions 3, 4, and 5, with the rest of the bits held low. This is done via the parallel feed-in option, with the inputs hard-wired to power or ground as appropriate. This is initialized with an "Init" input, which sets $S1, S0$, and CLK high at

the same time when toggled high, in order to feed in the initial paddle position. This "Init" signal is fed in as a pulse from the game-start JK flip-flop in the main circuit.

To ensure that there is no data-loss at the edges of the shift registers (including at the most-significant register which holds bits 8-11, of which only 8 and 9 are populated), the up/down buttons are only permitted to clock the registers when the paddle is not in the upper-most and the lower-most positions.

The outputs of the shift registers being fed into other shift registers are buffered to ensure that the shift registers feed each other correct data, and the clock signal is double buffered to ensure that any changes to S1 and S0 are made before the clock signal clocks the registers.

2.6 Collision Logic

Ball–wall collisions at the top and bottom edges of the LED screen are always assumed to occur and are directly incorporated into the logic controlling the ball's y-position counters.

At the left and right edges, ball–paddle collisions are similarly assumed in the x-position counters. However, an additional check is included to handle cases where no collision actually occurs (a paddle "miss"). In such a case, the ball is "scored," and the round resets before the ball can bounce and continue moving.

The collision detection functions by first detecting the balls alignment with the paddles.

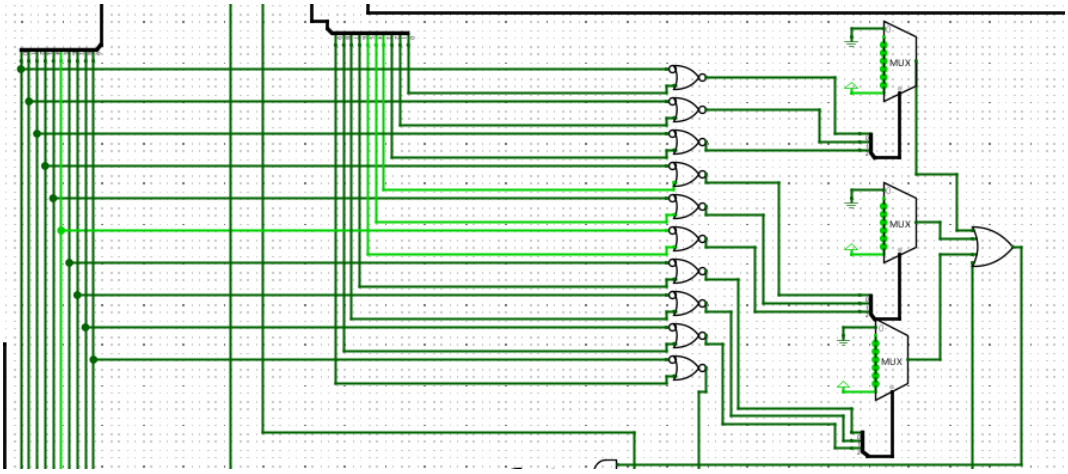

Figure 6: Collision detection sub-circuit

Figure 6 shows the sub-circuit used to detect a paddle miss on the left edge of the LED matrix; an identical circuit is used on the right edge. In the collision sub-circuit, the alignment is found by using a NOR gate on all of the inverted Ball outputs (columns of the LED matrix where the game takes place) with the paddles. Whenever a 1 is outputted on any of NORs, there is misalignment between the ball and the paddles.

What we are really checking with the NOR with the inverted Ball, $\overline{B}$ and the paddle P, is just seeing whether or not the ball is in a position that a paddle isn't.

An equivalent expression to $\overline{\overline{B} + P}$ is $B\overline{P}$, which is achieved by taking $\overline{\overline{P}\overline{B}} = \overline{\overline{B} + P}$ according to the De Morgan law. $B\overline{P}$ is an expression that very intuitively shows that the ball is in a position that the paddle is not, the reason the NOR gate implementation was used was for easier wiring purposes.

The multiplexers effectively act as OR gates, with each signal input that is turned on giving a 1 signal (all of the NOR gate outputs are signal pins to these multiplexers).

Figure 7 shows the same collision sub-circuit as the previous Figure, except now with additional logic gates near the bottom.

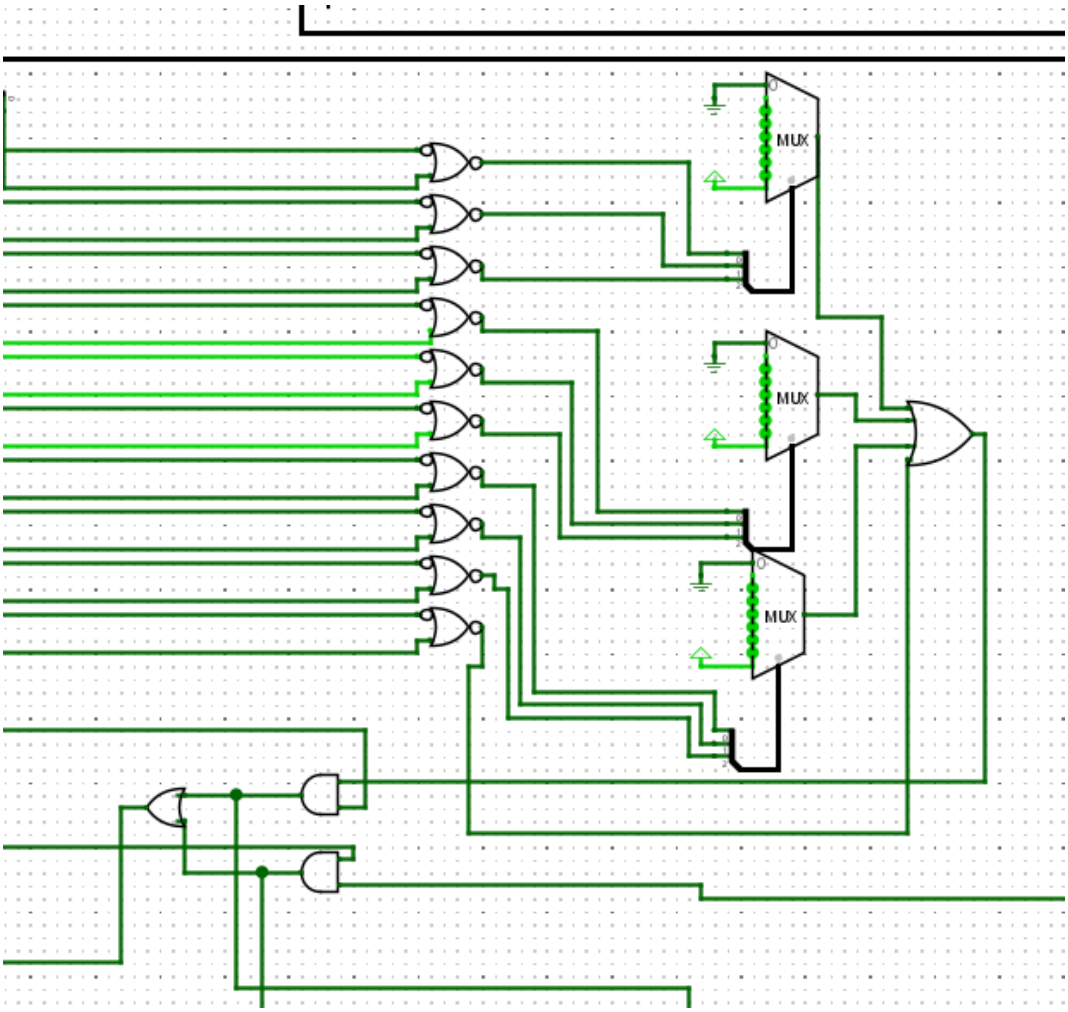

Figure 7: Collision detection sub-circuit including the column checks

So far, this logic does not account for the ball's x-location, even though collision checking should only occur when the ball is at x = 0 (left edge) or x = 14 (right edge). To address this, the paddle-miss signal is ANDed with the ball's x-position signal (high only when the ball is at x = 0 or x = 14), depending on which edge is being evaluated. This side-dependent "score" signal is then sent to the score counters to increment the opposite side's score. These side-dependent "score" signals are also ORed together for a side-independent "score," that goes high whenever any point is scored, which is then fed to the "start game" JK flip-flop.

2.7 Speed control

The speed of the game changes over time. To manage the current speed value selected, a "Non-load Counter" is used —naturally named "Non-Load" as its a logical block without programmable inputs— which is described below:

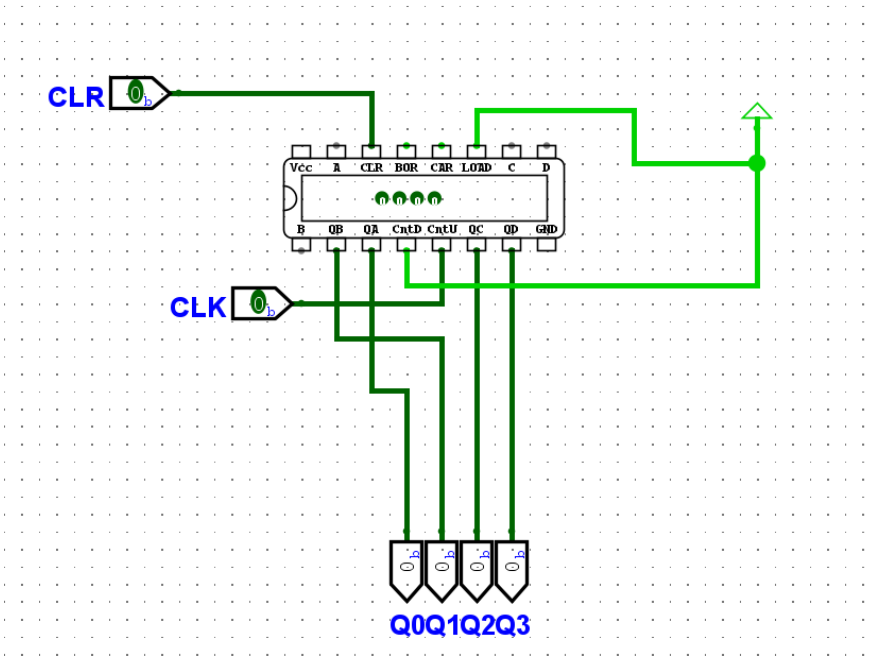

Figure 8: Non-load counter sub-circuit.

This circuit is fairly intuitive, there doesn't exist any inputs $D_3D_2D_1D_0$ as in the load counter, and the input pins, "CLR" (Clear), "CLK" (Clock), as well as the outputs $Q_3Q_2Q_1Q_0$, are quite self explanatory.

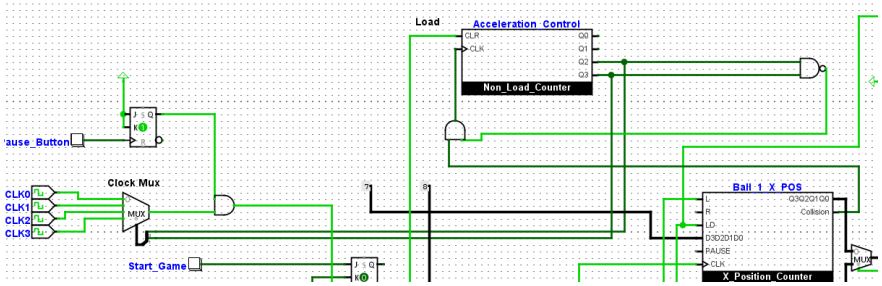

Figure 9: Speed Controller

The speed of the game is primarily managed by the collisions of only one of the balls, every 4 collision with either side of the board increments the speed one, which is done by feeding into the clock multiplexer on the left of the figure (A multiplexer with different clock speeds as its inputs). The "Acceleration Control" Counter is reset by the "Load" signal which is responsible for initializing the game, and it ceases to increment whenever both Q_3Q_2 (of the "Acceleration control" counter) = 11 (as that is the maximum speed of the logic).

*Within the simulation software used for this diagram, it is not possible to create separate clocks that have different clocking speeds without using any memory logic, so all the clocks feeding into the multiplexer are the same. This is not true in the real life implementation.

2.8 Score Counters

The sub-circuit that keeps track of each player's score makes use of two "Non-load Counters" (see Figure 8). The two simulated score counters are displayed in Figure 10. As seen, each counter is clocked by the output of a D flip-flop, which itself is clocked by the main game clock. The input to

each D flip-flop is fed by the lack-of-collision signal from the other side of the board (side-dependent "score"), thus clocking the non-load counter whenever one side scores a point against the other. The "Clear" (CLR) input of each counter is simply fed by a score-reset button, for the players to utilize at-will.

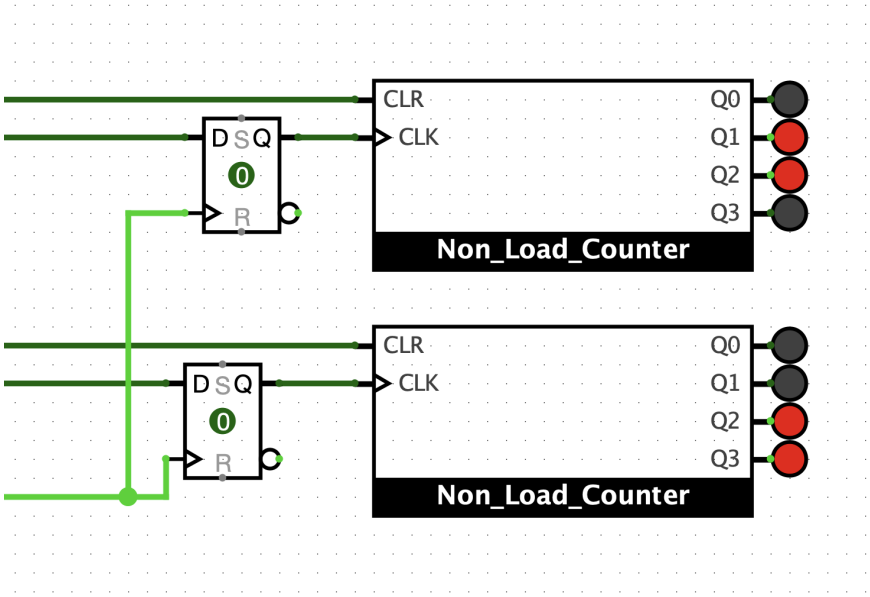

Figure 10: Score counter sub-circuit.

2.9 Game State

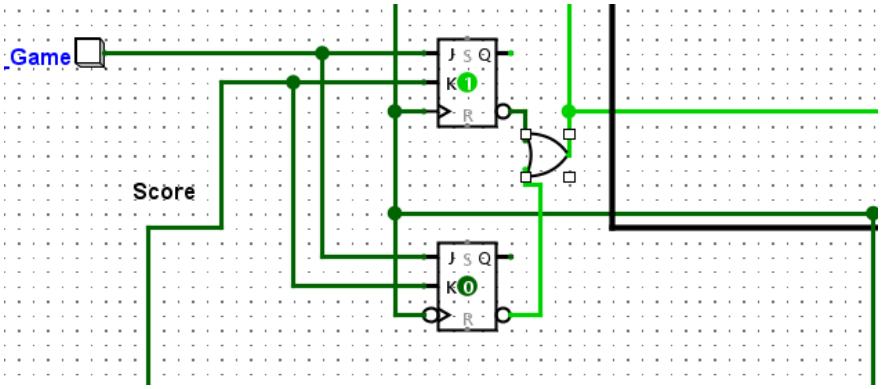

Figure 11: Overarching game state machine.

In the figure above, two JK Flip Flops are shown, although they have the same functionality, the only differing quality between the two is that they have opposite edge triggers which is required for the correct functionality of the scoring mechanism (which really makes them function more like a dual edge triggered, singular JK flip-flop).

Game state is toggled between ready-to-run ($Q = 0$) and running ($Q = 1$). We utilize a JK flip-flop, with the input to J being a "Start Game" button, the input to K being the direction-independent "Score" signal (that is high whenever a point is scored), and the clock input being our clock output (and whatever speed of clock happens to be chosen at the time). This sub-circuit is displayed in Figure 11. It should be noted that the clocks are ANDed with a "Pause" signal such that they only run when the game is unpaused, though this did not end up being implemented in

real life. A state transition table is displayed in 1, noting that we utilize the inverse of the state variable, writing all outputs in terms of $\overline{Q}$.

$Start$	$Score$	$\overline{Q_n}$	$\overline{Q_{n+1}}$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Table 1: State transition table.

When a score is made, $\overline{Q} = 1$,. This value gets fed directly into our "Init" or "Load" signals for the balls and paddles, such that our game is reset as soon as we score. We stay in this "ready-to-run" state until the "Start" button is pressed, which turns $\overline{Q} = 0$. This stops the initialization of the ball and paddles, and allows them to run as normal for the game to continue.

For this state machine to work, we must initialize it in the $\overline{Q} = 1$ or "ready-to-run" state such that our paddles and balls are initialized in their starting positions. This does not work in Logisim, so we implemented an additional button which would score a point to start the game. In real life, however, this can be achieved using an RC circuit to quickly load the IC with our desired value whenever the circuit is first powered.

2.10 LED Matrix Handling

Although an array of 2 x 5 of 5 x 5 LED matrices are used providing a resolution of 10 x 25, the used portions of the LED matrix actually measures 10 x 17. The reason for this is that the circuit for handling the paddles is heavily simplified by using a discrete paddle LED matrix rather than using any persistence of vision effect for displaying the paddles and other game elements on the same LED matrix. As such, the middle 10 x 15 matrix (consisting of 6 total 5 x 5 sub-matrices) is where the two balls actually move, and is therefore switched at high frequency to display each ball. The outer 4 sub-matrices are utilized to statically display the paddles only on the inner column closest to the balls' matrix.

2.11 Controllers

The controller sub-circuits were made with the intention of portability (such that each player could hold one in their hands during gameplay). As such, these sub-circuits are quite small, consisting only of two RC de-bounced buttons, which get fed into the paddle sub-circuit, as well as a four-LED display to display each player's score (for which the input comes from the score counters).

3 Implementation

The described circuit was implemented on breadboards and LED matrices using jumper cables and bent solid stranded wire. An image is attached below. Figure 12 displays the finished circuit, with sub-circuits annotated. Figure 13 displays the LED matrix screen, which is located on the right-hand-side of the previous Figure, though oriented in a manner that makes it poorly visible to

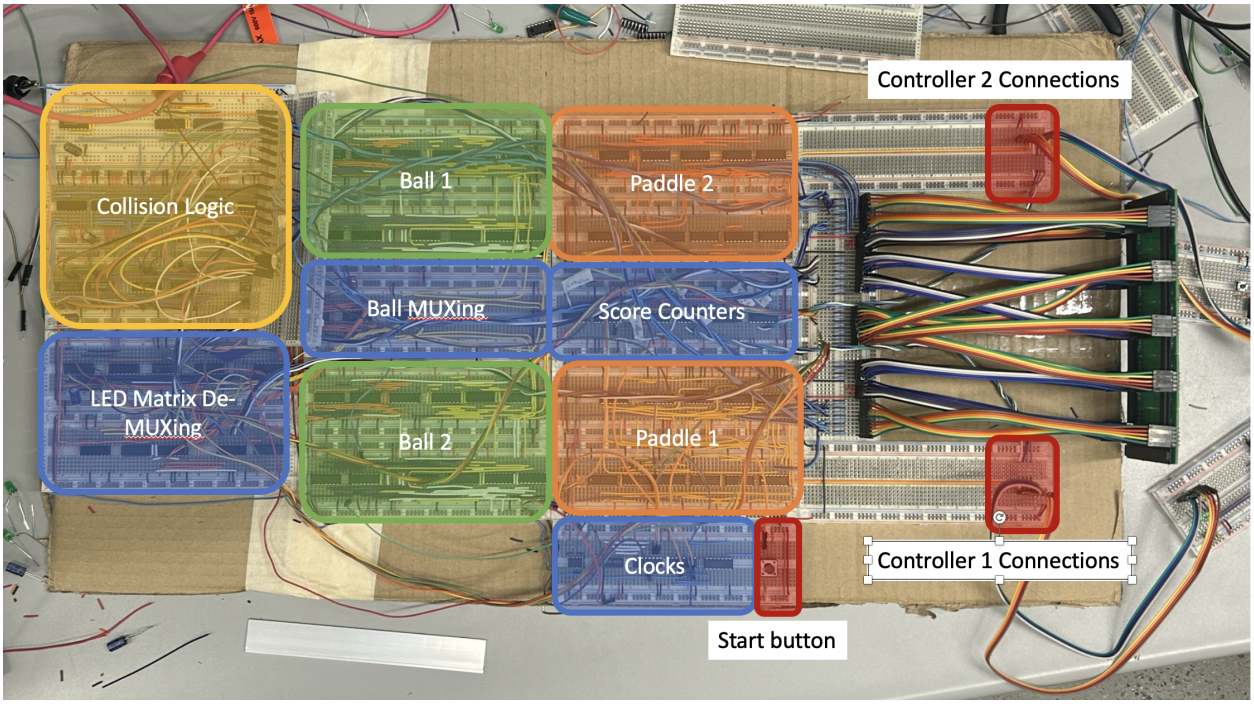

Figure 12: Annotated image of completed circuit.

the camera. A breadboard was used for distribution and connection of the jumper cables attached to the male headers of the LED matrices according to the rows and columns that need to be shared across.

Figure 14 displays one (out of two) controller sub-circuits, which connects to the main circuit via jumper cables and the breadboards labeled "Controller 1/2 Connections" in Figure 12. The controllers were created with mobility during gameplay in mind, hence their very long jumper cable connections.

4 Conclusion

The implementation of the circuit ultimately did not exhibit desired behavior when built on breadboards. When testing, the paddles' control and display modules worked, but the ball outputs that did not behave according to the defined specification above. One ball was observed to be traveling close to desired behavior, while the other was jumping all over the LED matrix in seemingly random patterns. The rest of this report will detail troubleshooting and possible errors in the implementation.

4.1 Troubleshooting

To isolate the problems associated with each ball, the display multiplexer signal pin was held low or high to display only one ball at a time. Because the collision logic is dependent upon the output of the display multiplexer's outputs, the non-displayed ball would not cause any erroneous behavior while testing one ball only, allowing for testing of one ball's logic at a time.

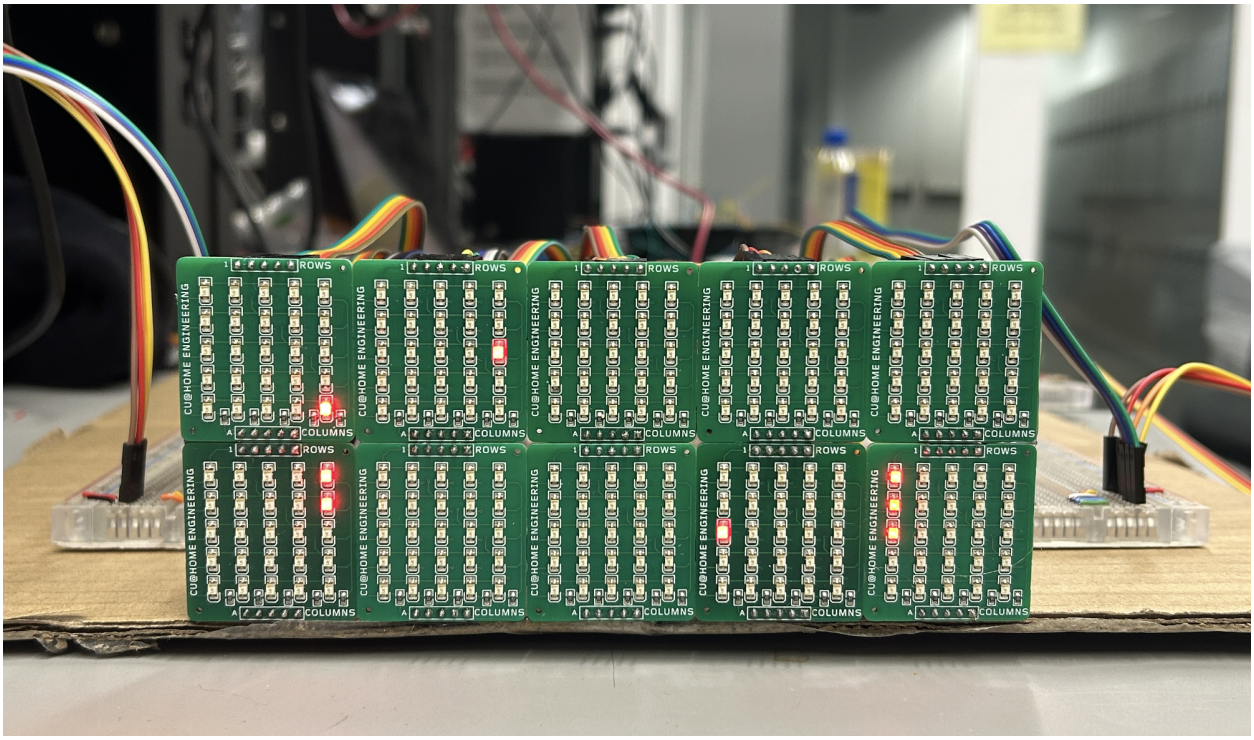

Figure 13: Image of LED matrix screen, showing two paddles on the left- and right-hand sides, as well as two balls in the middle.

4.1.1 Ball 1

In the ball when the display multiplexer's signal pin was held at 0 (henceforth referred to as Ball 1), the LED matrix displayed proper x-axis movement from the ball, traveling from x=0 to x=15 in 1 step increments and back, but reset logic (when the ball would return to its default position) was strange, with the ball at times bouncing and not resetting when there was no paddle to bounce off of, and at times resetting even when the paddle was present. This issue was determined to be due to the y-axis outputs being wired to the LED matrix incorrectly causing collision logic to be improperly handled. After fixing this, Ball 1's y-axis movement was still erroneous, bouncing between y=3 and y=6 and seemingly switching directions every clock cycle. While the proper wiring of the y-axis outputs partially corrected this, we were ultimately unable to resolve this issue.

The most grounded theories of what may have caused the unintended behavior lies with two specific chips: the JK Flip Flop used to handle direction control, and the 3 line to 8 output decoder chip used in our implementation of Ball 1. To begin with, during construction and testing of our two discrete ball circuits, we discovered that there were two different part numbers of the JK Flip Flop in use that exhibited different behaviors in our circuits. While one JK Flip Flop worked as intended, a JK Flip Flop of the same series but of a different model failed to properly switch directions when the logic to change the direction of the JK Flip Flop was true. We had previously remedied this by substituting in a JK Flip Flop with the part number of the chip that was confirmed to have been working previously, but we suspect that due to the nature of the erroneous behavior (being related to the direction of the ball's movement), that the JK Flip Flops used may have played a part.

We also suspect that there may have been an issue with the 3 line, 8 output decoders that were used for the handling of the LED matrix. The implementation of the 3 line to 8 output decoders used necessitated the inversion of all of the y-axis outputs due to the active-low output of the decoder chip, which because the x-axis was working properly but the y-axis was working strangely,

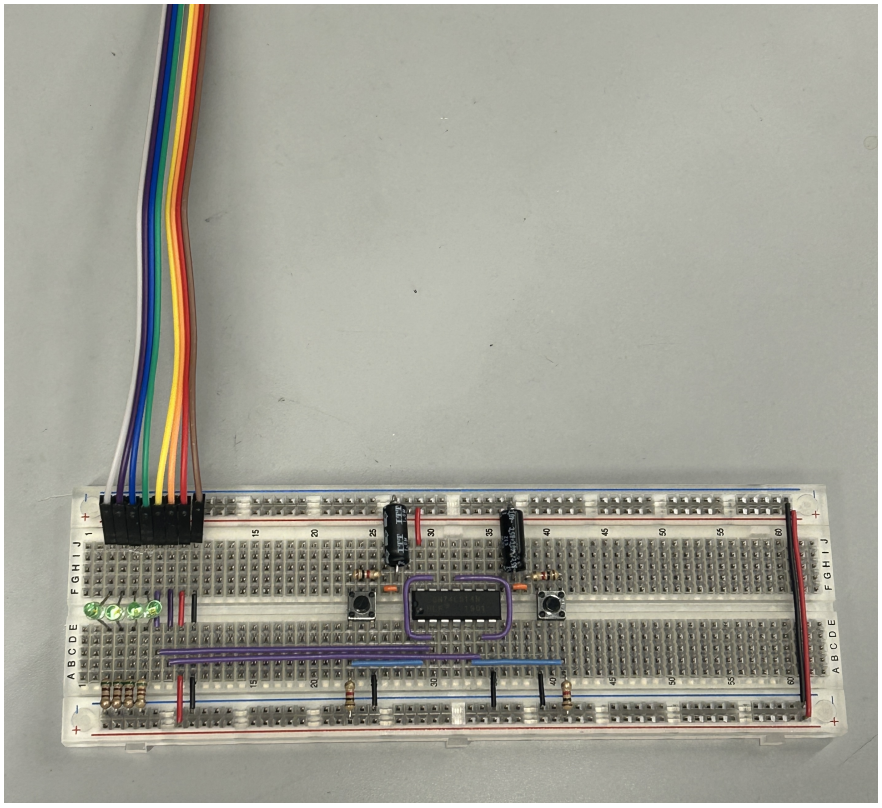

Figure 14: Image of controller sub-circuit, with two de-bounced buttons in the middle for paddle controls, and a line of four LEDs on the left for score display.

suggests that the inversion of all the y-axis outputs may have played a role in the issues with the implementation. One way that we may have been able to debug this was to utilize dip switches to manually cycle through the y-axis positions of the ball and identify possible errors in wiring, but we ultimately ran out of time to troubleshoot this.

4.1.2 Ball 2

Ball 2 exhibited similar issues as Ball 1, but also exhibited erroneous movement across the entire LED matrix that was eventually fixed by re-wiring all the connections from the x and y outputs to the LED matrix.

4.1.3 Both Balls

After fixing the movement of the balls, another issue presented itself on the LED matrix in the form of the balls disappearing for several clock cycles at a time, and reappearing in positions that did not make sense given the context of the state at which they disappeared. We suspect that the disappearance of the balls is linked to the usage of two 3-line to 8-output decoders used in conjunction with a 2:1 multiplexer to effectively create a 4-line 16-output decoder, and either the improper wiring of the 4-line 16-output decoder which resulted in the ball going to unused outputs, or improper wiring of which outputs from the counter went to which control pins for the decoder, with an end result of the ball sometimes going off-screen. This does not, however, explain the disappearing of the balls completely and for what seemed to be a random number of clock cycles. Further troubleshooting could have been done by probing the control pins of the decoders and confirming that the output channels corresponded correctly to the decoder's control pins, and that

the decoder was properly wired to achieve a 4-line 16-output decoder, but unfortunately given the time constraints we were working under, we were only able to confirm that the counters responsible for handling the ball's position coordinates were properly working.

4.2 Final Implementation

In the end, the final implementation was not functioning to the defined specification and while ball movement was mostly fixed, the ball would disappear and reappear seemingly at random times, and would not trigger any resets while not displayed on the LED matrix. Our final assumption is that there was an issue with the wiring of the 4-line 16-output decoders that led to the ball not always being displayed.

4.3 Limitations and Potential Improvements

Unfortunately, our design requires discrete hardware for each additional set of balls due to the combinatorial logic used to handle the balls, and increasingly large decoders and multiplexers to handle displaying each of those balls for additional balls to be used. It is likely that n-ball pong can be made by instead of using discrete counters, loading each ball's position and momentum into SRAM and using one set of counters to increment all the balls stored in RAM by 1, displaying all the balls statuses, and then repeating until a score is achieved. This would utilize sequential logic and RAM rather than purely combinatorial logic like we used in this project. While this is likely more efficient for n-ball pong, it is not certain whether the implementation of this n-ball pong for n=1 or n=2 would be simpler than implementing 1 or 2 ball pong using combinatorial logic and discrete sets of counters for 1 and 2 balls.

Additionally, our physical design does not allow for the inclusion of a score reset, instead relying on a power cycle to reset the score of the game. The implementation of a score reset is simple: pure combinatorial logic ANDing the outputs of the score to determine at what point both scores will be reset (and when one side wins). A button can also be OR'd with the AND gate output to allow for a manual score reset. We were able to simulate this in Logisim, but did not end up incorporating this in the physical circuit due to prioritizing testing and fixing more important parts of the circuit.