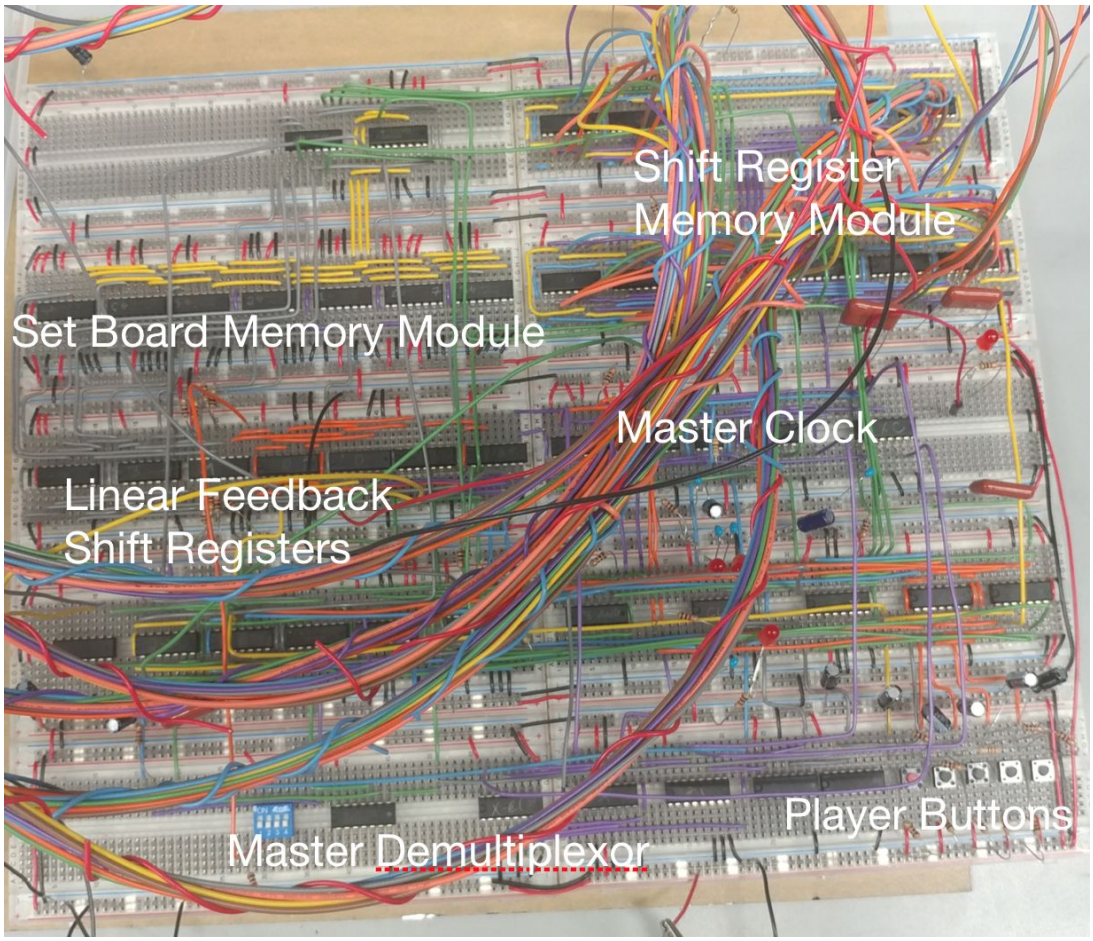
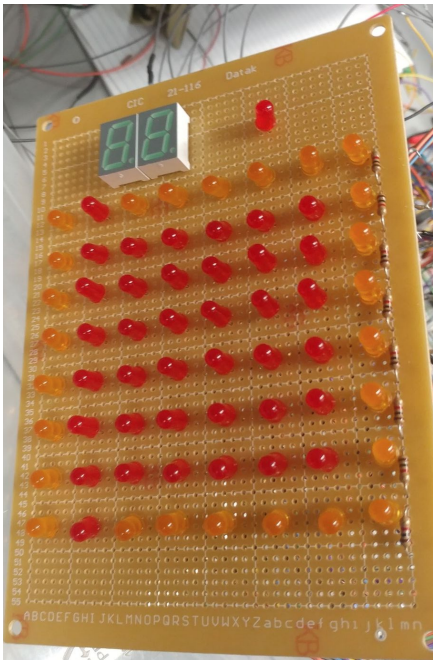
Andrew Lorber and Joshua Yoon
Professor Risbud
DLD Final Project - Don’t Break the Ice!

Abstract

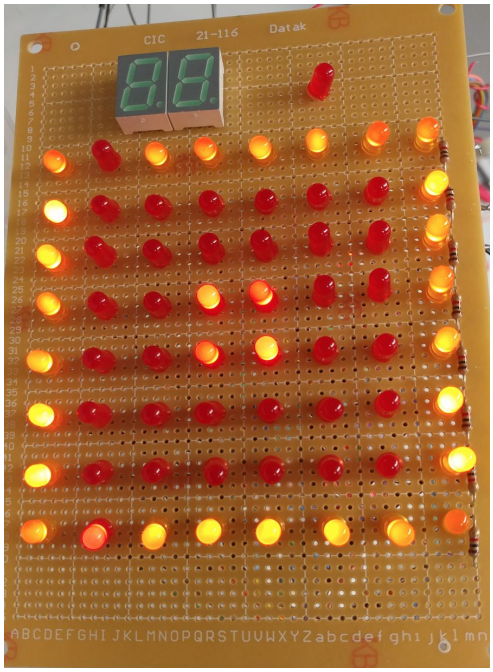
The game plays as follows: each tile on an 8x8 matrix of LEDs must be stepped on once and only once. The goal is to reach a specified ending position from a specified starting position. If the player goes over a tile that has already been stepped on, then the player will lose. The user interface will be five buttons, one for right, left, up and down movement and one to reset the board. The beginning and ending positions will be fixed. In order to add difficulty and a random element, the player can choose the difficulty setting with a DIP switch and a mux will be used to randomly choose a board of set obstacles. There are 4 boards designed for the “easy” setting and 4 boards designed for the “hard” setting. This is necessary as it possible to spawn puzzles that are impossible.

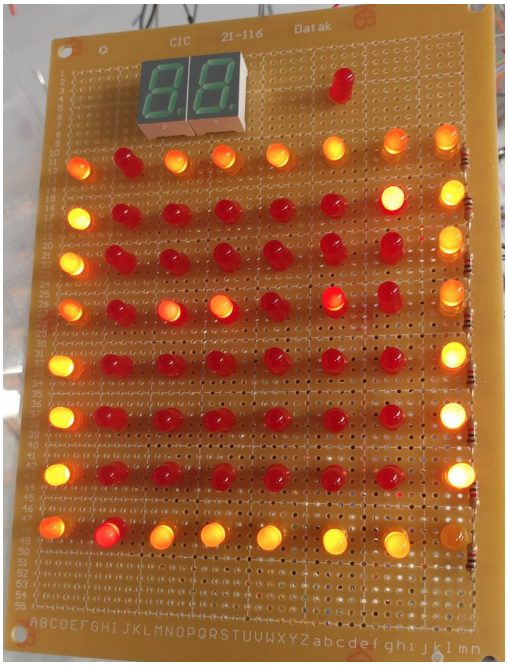

Shift Register
Memory Module
Set Board Memory Module
Master Clock
Linear Feedback
Shift Registers
Player Buttons
Master Demultiplexor



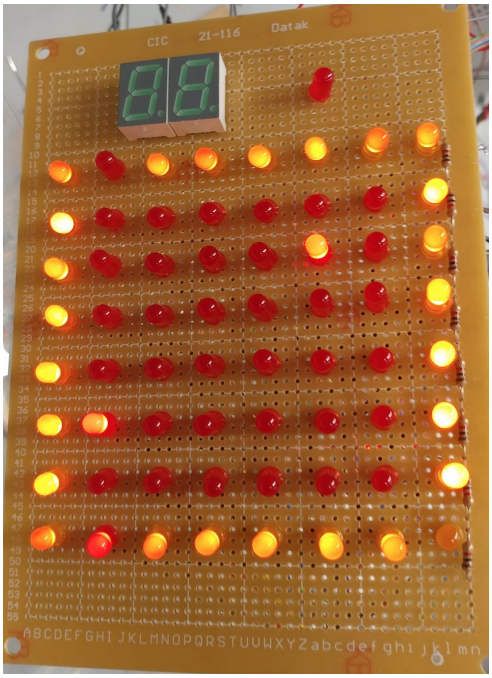
Random Board States

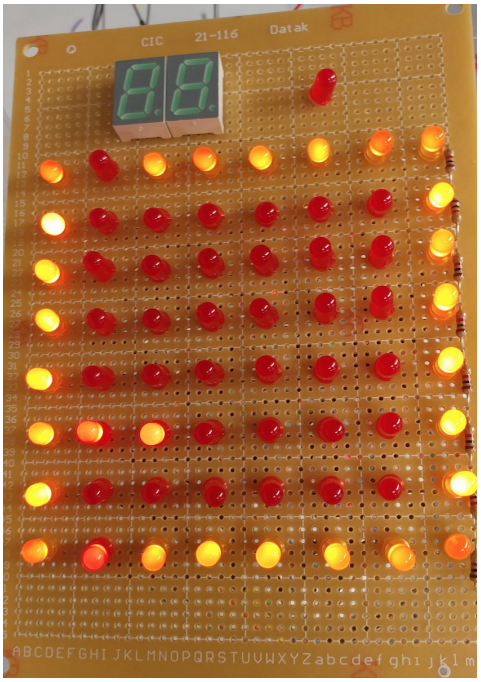
Hard Mode





Easy Mode





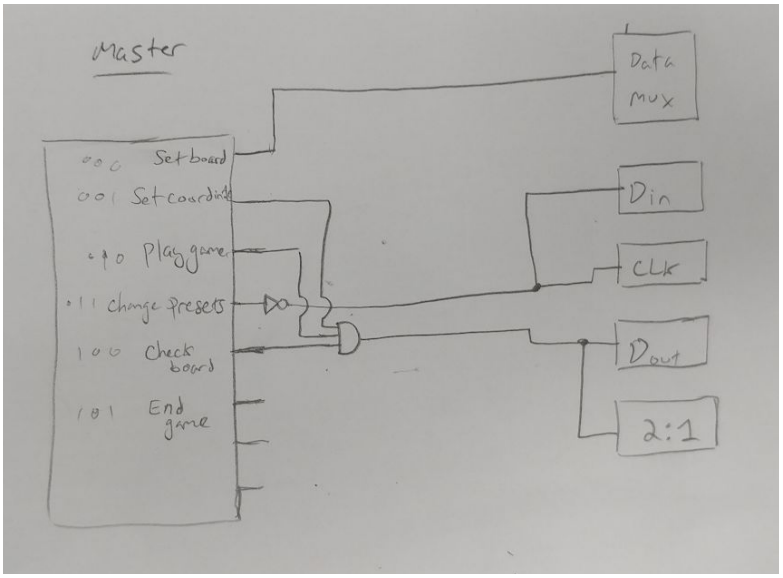
Design

The design is split up into 4 main components: set board, collision detection and memory, user interaction, and combinational logic for the master demultiplexer. Each of these, as well as the master demux, will be explored in further detail. Below are initial designs, and the final schematic diagram is after the module explanations. The timing diagram depicting what happens over a single cycle of the game (including pressing player buttons) is placed at the end of the documentation due to formatting constraints.

There is also a master reset button that will perform the following functions: reset all of the shift registers to 0, preset the initial board setup phase, clock the linear feedback shift register, and reset the master demux back to 0 so the game can restart.

A master counter preset to 8, binary mode and counting down is also used to make sure that the different modules receive exactly 8 pulses. This is accomplished by hooking the carry out of a CMOS 4029 to the reset pint (pin 4) of the master 555 timer. When the counter is preset to 8, the carryout is high, and thus the 555 timer is allowed to operate normally. However, once the counter hits the 0000 state, carry out drops to low and the clock is paused. The clock will not run unless the counter is preset to 8, which will then set the carry out to high and the cycle is repeated.

Master Demux



The game has several important states that it must cycle through in order to be operational. They are as follows: Set up the board and the obstacles; set starting coordinates; play the game; check if the player has won or lost; reset the board. In order to efficiently manage these states, a master demux will be used to choose which part of the circuit to activate at once and inhibit the other parts using the inhibit pin of the (de)multiplexores; this allows us to also short together outputs, and this is useful as different inputs must go to the shift register memory module. The demux input is connected to ground, as the inhibit pins of the CD4051 chips are active high.

The picture to the right depicts in rough details what states will activate what parts of the circuit. The data mux refers to the demultiplexer that controls the set board state, which will be discussed further in the design section. The Data In multiplexor and Data Out multiplexor are used part of the collision detection and shift register memory modules, and thus are activated

during the set coordinate, play game, and check board states. The 2:1 block means a 2:1 demultiplexer, which is used to jam in the correct player coordinate into the memory module.

The master demultiplexer outputs are controlled with a counter that is preset to 0000 at the beginning of the game. This counter is clocked whenever specific events occur within each state, to change the addresses of the demultiplexer to the next state. For example, during the set board state, the carry out of the Y-Counter and Q4 of the data mux counter are NORed together; when both of these states hit zero, the master demultiplexer counter should clock. These states mean that the Y-Counter has reached 1111, meaning the last shift register of the memory module has been reached, and when the last bit is inserted into the shift registers, which is when Q4 switches from 1111 to 0000. The basic schematic used is displayed to the right.

Set Board State and Obstacle Setting

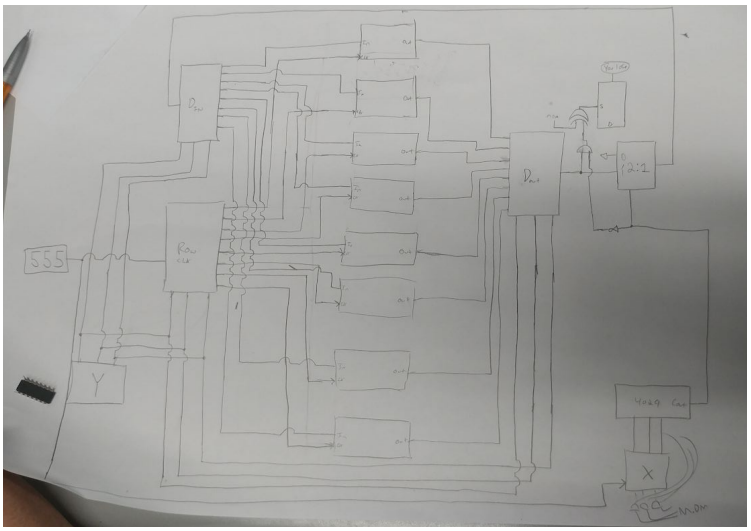
The first state of the game is setting the perimeter border as well as the random obstacles. The perimeter is useful to let the player know that they cannot go off the edge of the board and it also adds a layer of difficulty, since there will only be one possible 'entrance' to the last LED. The starting and ending positions of the user will be fixed, as this will allow us to know if the game is possible and when to check if the game has been won.

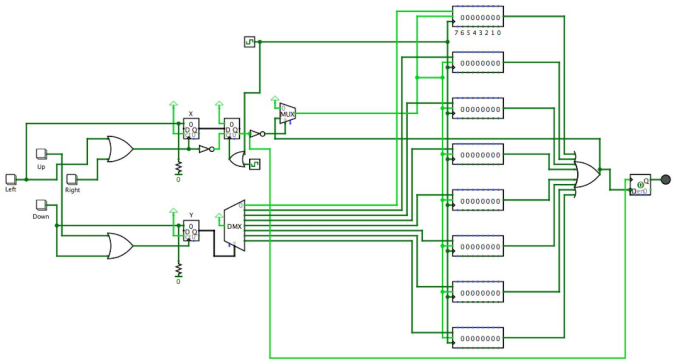
Randomly generating obstacles is problematic as not all random boards are possible to complete in this game. Therefore, we decided to design eight boards, four easy and four hard, that we know are possible to complete. The board will be set using seven muxes, one for each row (the top and bottom rows are identical). The mux outputs will each feed into their respective shift register and be clocked eight times. This will enter the bits on each input of the mux into the shift register in order. Due to this design, we can use a demux to select the board and have combinational logic (depicted below) to decide which bits will be '1' and which will be '0'. The player will control a DIP switch that will be hooked up to the C address pin of a 1:8 demux. This will allow the user to decide which four pins will be randomly chosen from, effectively deciding whether they want an easy or difficult board. A linear feedback shift register will be used to to randomly select the A & B pins of the demux, which will randomly choose a board of the set difficulty to load. The linear feedback shift register will be clocked to another random value every single time the user hits the reset button. Sometimes the same board will be loaded onto the matrix, as the linear feedback shift register might clock to the same state as before.

Collision Detection and Memory

We will use two counters (4029) for position, one for X and one for Y. We will use 8 shift registers (4015), one for each row. Since these shift registers have a master reset, as opposed to a master set, we will need to light on '1'. When the player moves, we will use a demux (4051) to choose the correct shift register based on the Y coordinate and then we will

pre-set the X coordinate into a counter (4029). The counter will be attached to a clock (555) that will also be attached to the shift register. The clock will pulse really fast to cycle through the shift register until we pull out the number we need. The bits pulled out will go into the input of a 2:1 mux (4053). When carry out of the counter is high, the bits will loop back into the shift register. However, when carry out goes low, meaning the counter hits zero, a 1 will be entered into the shift register instead, signifying the player position. A D type flip flop (4013) can be used to detect whether it is already a one and activate a game over sequence. A counter (4029) will be preset to eight on each button press with the carry out sent to the reset pin of the clock. This ensures that exactly eight pulses are received by the shift registers.





A bug was found where the shift registers would register an extra pulse of the clock when moving left or right. This was solved by adding more counters and logic to the schematic. Two X counters were used; one for the true X coordinate and one that was one number less. Logic was also used to change the preset of the 8 pulse counter from eight to seven. This allowed us to have the clock pulse seven times when the right or left button was pressed. A 2:1 mux and a D flip flop were used to select which counter to rely on for the clock. While this fixed the bug, though, it caused the up and down buttons to count an extra clock pulse. This caused many issues and the solution will be discussed below.

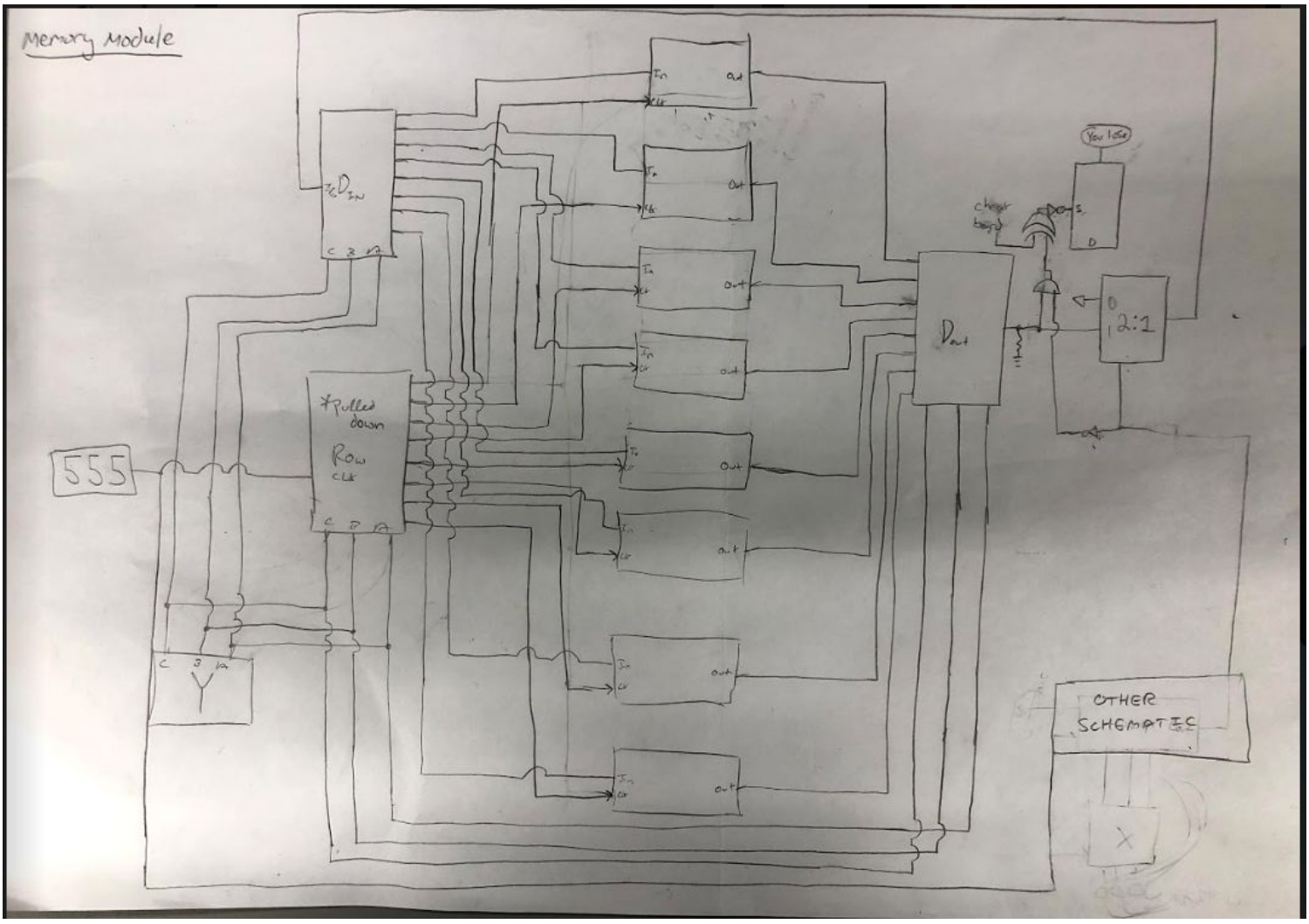
Combinational Logic for the Master Demux and Player Interaction

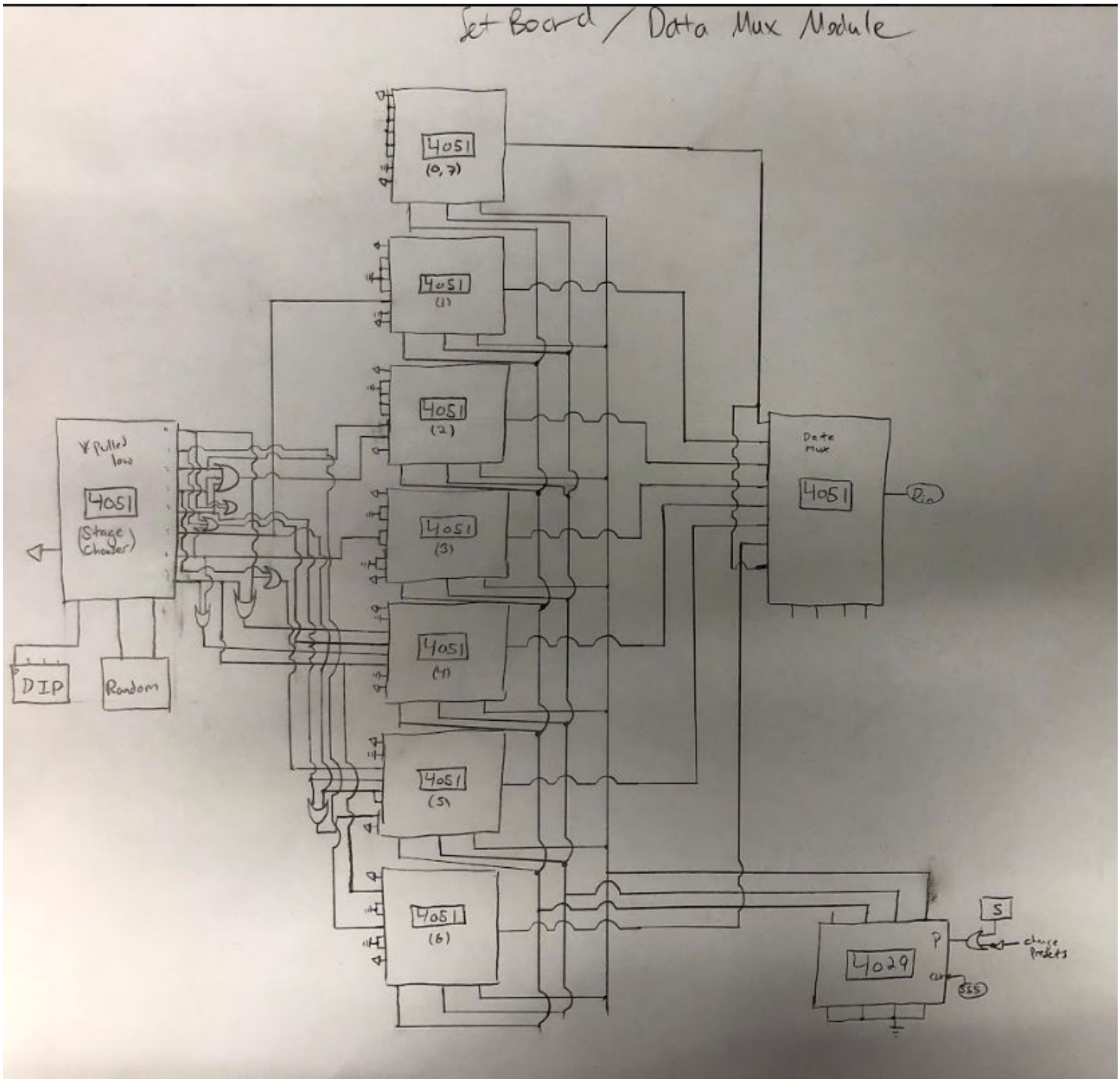
There is a considerable amount of combinational logic necessary to make sure that the numerous clocks, counters, and demultiplexers work as planned. For example, the counter that keeps track of the number of pulses the 555 timer outputs needs to run continuously for the checking board and set board state, and thus combination logic with the mux outputs is used to ensure that that this is the case. For the set coordinate and play game states, 8 pulses are necessary only during specific cases; in the play game state, 8 pulses should clock the shift

register memory module only when a player presses a button. For the set coordinate state, the counter requires only one 8 pulse cycle to preset the starting coordinate, and thus the counter is preset. A rough schematic used while building is depicted below:

Player interaction consists of 4 buttons that, from left to right, correspond to moving the player LED Right, Left, Up and Down. The buttons were debounced using RC circuits to smooth out the inevitable mechanical noise. The buttons are also heavily involved in the combination logic, as the counters are either clocked up or down depending on which button is pressed, and thus the UP/DOWN pin of the CMOS 4029 chips are manipulated, as seen in the schematic on the previous page. These buttons are also ORed together and fed into the master counter that outputs 8 pulses, as mentioned previously in the design.

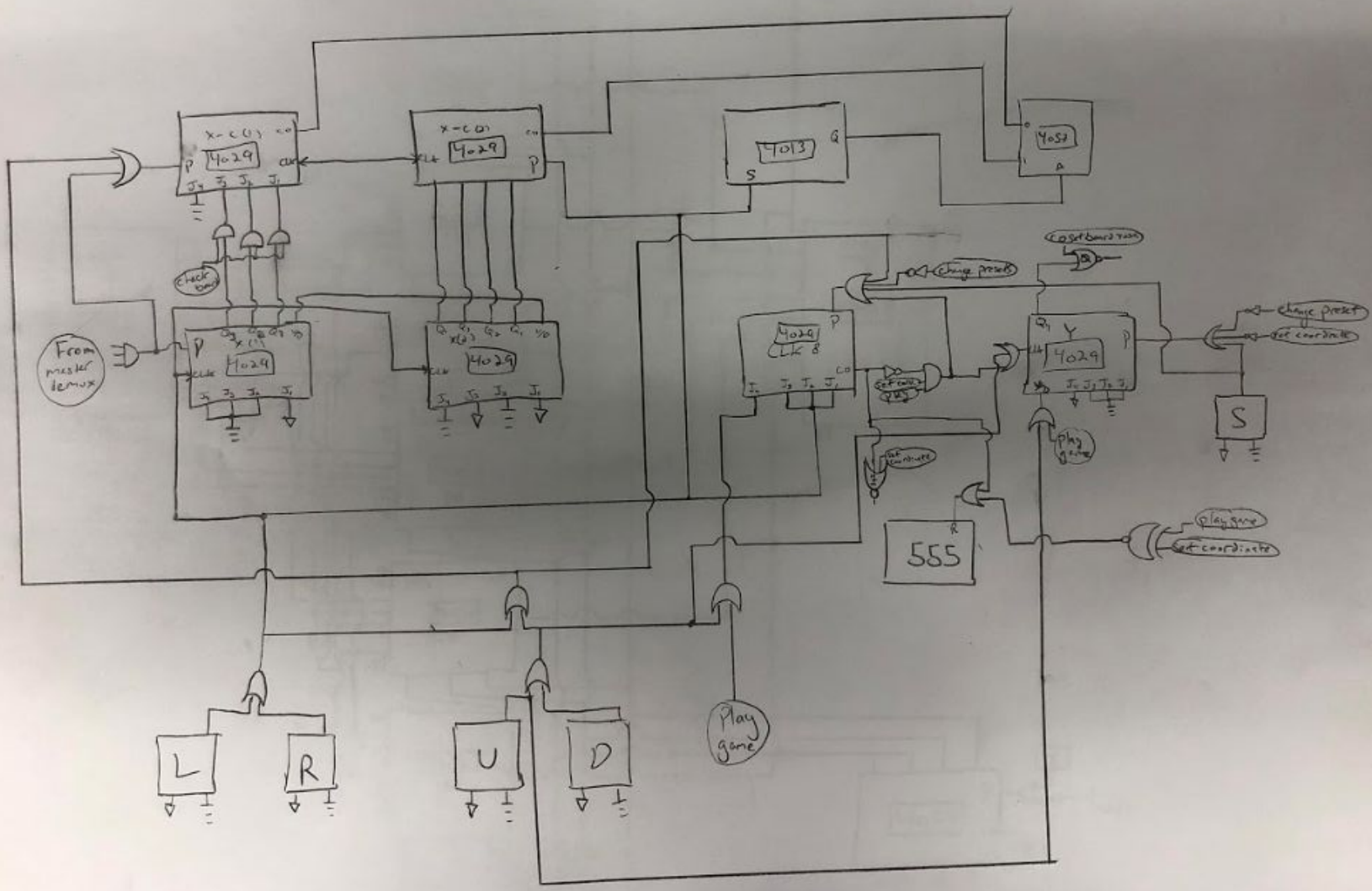
Schematic Diagram



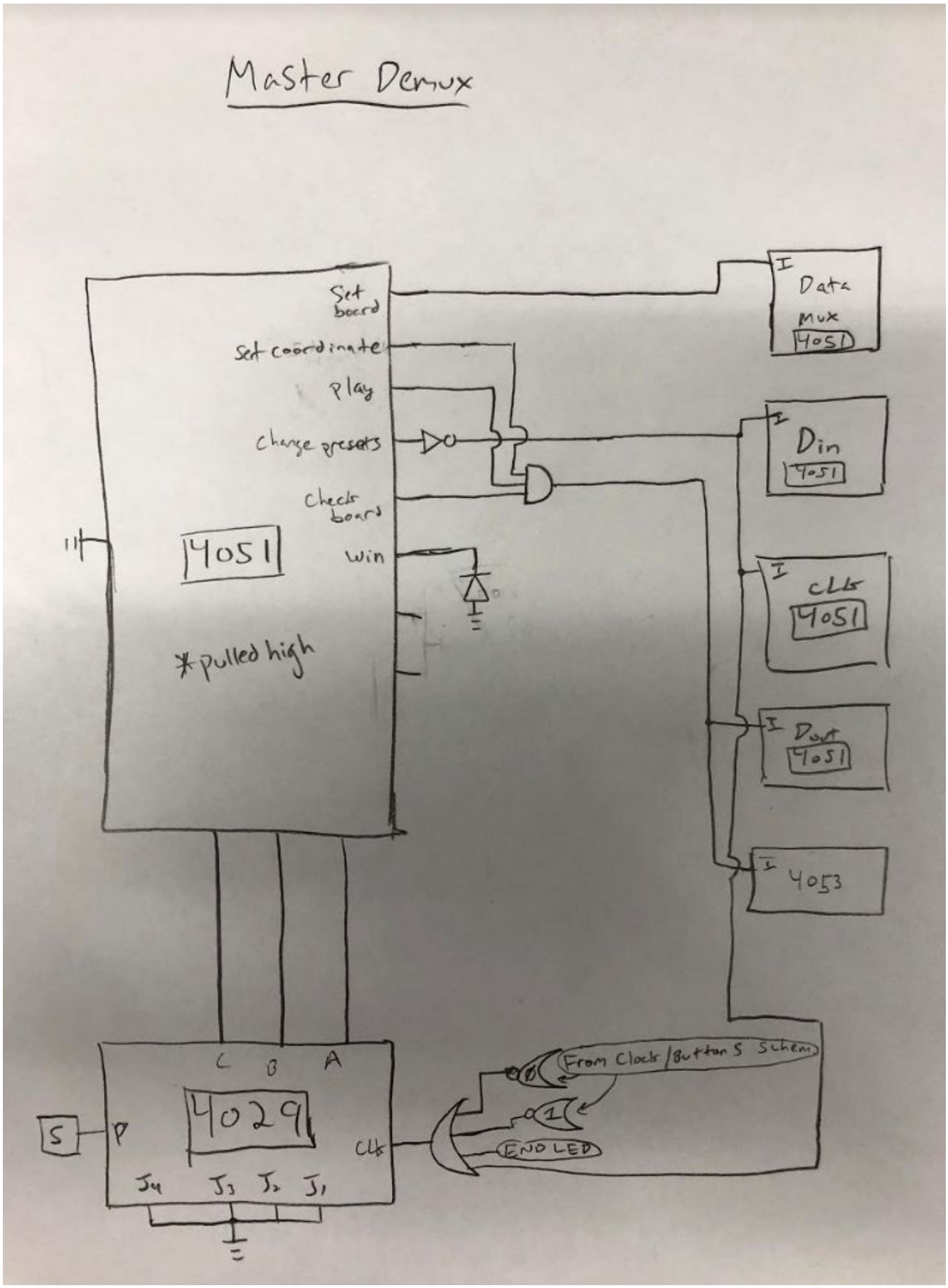

Set Board / Data Mux Module
4051
(0,7)
4051
(1)
4051
(2)
4051
(3)
4051
(4)
4051
(5)
4051
(6)
Data mux
4051
D.o
* Pulled low
4051
(Stage Chaser)
DIP
Random
4029
P
S
clock Presets
clock
555

*words are Stages on master demux

CLOCK/BUTTONS/COUNTERS



Master Demux



Parts List:

- 555 - Clock: 2 clocks
- 4013 - D Type Flip-Flop ~4-5 chips
- 4015 - 4 Bit Shift Register ~8 chips
- 4029 - Up/Down Counter ~6-8 chips
- 4051 - 8:1 (De)Mux ~14-15 chips
- 4053 - 2:1 Mux ~1 chip
- 4069 - NOT ~1 chip
- 4070 - XOR ~1 chip
- 4071 - OR (2:1) ~3 chips
- 4075 - OR (3:1) ~2 chips
- 4081 - AND ~3-4 chips
- 8x8 Perf Board
- 5 push buttons
- Resistors (for pull down and timers)
- Capacitors (for timers, debouncing, decoupling)
- 10 breadboards maximum

Work Distribution

Andrew was responsible for designing and building the memory module circuit. He did this by taking advantage of the two 4 bit shift registers available on the CMOS 4015 circuit in order to save board space and efficiently loop in information back into the same circuit. He used a flip flop for collision detection and 2:1 mux to properly loop the information back into the memory module. He was also responsible for designing and building some of the player interaction and the clock and counter mechanisms.

Josh was responsible for designing and building the set board module as well as setting up the master demux and the 8 bit clock counter as well as the 555 timer. He was also responsible for building out the LED matrix and interfacing it with the board.

Both team members were responsible for wiring and debugging the circuit.

Error Discussion and Troubleshooting

The following is a timeline of our design and building process and the bugs that occured.

- Built memory module and tested using LEDs
- Built set board data module and tested using LEDs
    - BUG: Linear feedback shift register not outputting random values, locking into one value when clocked.
        - SOLUTION: Tester LEDs were sinking too much voltage, output of the shift registers were checked with oscilloscope and found to be working properly.
    - https://drive.google.com/file/d/1iD7MwtdJQfQQ1n4-5gj5x2pzGs3TFobb/view?usp=sharing
- Soldered LED matrix with heat shrink, connected leads to anode leads of the 64 LEDs.
    - Tested LEDs by connected resistors and power supply to anode and cathode leads.
- Built Master 8 Counter and Master clock.
- Built Player Counter and Interaction Module.
    - Buttons debounced with RC circuit consisting of 1 uF capacitor and 1 KΩ resistor, output confirmed using single mode of oscilloscope.
- Built Master demultiplexer counter and hooked up states to clock the master demultiplexer.
- Main modules now completed, debugging begins in earnest.
- BUG: First power up of all modules fails. Test LEDs are not lighting up.
    - SOLUTION: A short was present in the board. After checking all of the connections to the chips, Andrew noticed that a ground wire was connected to the power line, resulting in a massive voltage drop.
- First row of LED matrix attached to the shift registers.
- BUG: Master demultiplexer not clocking properly. Not inhibiting the correct states.
    - SOLUTION: Outputs of the demultiplexer required pull up resistors so that they are default high and set to low when the corresponding address is reached.
- BUG: Counter of the data mux was counting in a very erratic fashion, skipping numerous states and thus not setting the correct values to the shift registers.
    - SOLUTION: It was determined that messing around with the 555 timer reset pin was sending weird values to the clock of the Data Mux Counter. This problem was resolved with an XNOR gate to set the reset pin of the 555 timer high during states where it needs to be continuously run.
- BUG: During the set coordinate phase of the master demultiplexer, the bottom row was clocked 8 times to jam in the value of the player's starting position, which works as

intended. However, when the demultiplexer went to the next stage, the play game state, the 8 cycle set coordinate state would occur again, which is problematic as this triggers collision detection.
    - SOLUTION: It was determined that the demultiplexer was being clocked one pulse too early, while the carry out of the 8 clock counter was still low, and thus the counter was preset once more and resulted in the extra state. This was resolved by slowing down the play game state of the demultiplexer that was clocking the preset enable of the 8 clock counter with an RC circuit.
- BUG: Top row of LED matrix was resulting in buggy states.
    - SOLUTION: The shift register values of the top row were jumped to a breadboard with LEDs, and was found to be working correctly. Thus, the problem was with the matrix itself. It seems that some of the orange LEDs sink more current than other orange LEDs, resulting in the LEDs not receiving enough current to turn on. Also, 5.1 KΩ resistors were connected to the LEDs, which was too much resistance and resulted in the LEDs being too dim. The issue was resolved by resoldering a 1 KΩ resistor and replacing the faulty LEDs.
- BUG: Far right LED of each row not turning on.
    - SOLUTION: As mentioned previously, 5.1 KΩ resistors was too much resistance for the LEDs to be bright, and thus 1 KΩ replaced the existing components.
- BUG: The buttons were being latched to high, resulting in the preset enable being held high and the counters not properly clocking down as they should have.
    - SOLUTION: The buttons were pulled down with 1 KΩ resistors, and decoupling capacitors were added to try and get rid of the spurious state.
- MAJOR BUG: Off by one error.
    - SOLUTION: Discussed on the following page.

Major Bug Debugging - Post Presentation Discussion

In order to determine what the bug is, we systematically tested each of our functional blocks individually and together to eliminate potential sources of error. The bug was occuring in the play game state, and thus we did not have to worry about the set board or set coordinate states of the master demux, as that occured in states 1 (set board) and 2 (set coordinate). Also, the set board module was already confirmed to be working properly when we connected it to the LED matrix and observed the board setting in properly. Thus, we were able to determine that this part of the circuit was not causing the problem.

We then moved onto testing the memory module of the circuit. We first observed with LEDs and then subsequently with an oscilloscope the inputs that were being fed into the data in multiplexor, and confirmed that the LEDs were cycling properly.

We first took an oscilloscope to the 8 cycle clock counter to see if there were any spurious pulses. There appeared to be no spurious pulses, as we saw 8 clear timing pulses. Thus, we isolated our problem to either the player input or the combinational logic surrounding the counters and master 555 timer. We experimented with capacitors and resistors that made up the RC circuits that debounced the buttons, but these didn't seem to make much of a difference.

Post Presentation Debugging Attempts

As mentioned during the presentation, we were not sure if the error resulted from spurious design or from the player inputs. We decided to continue down the route of debugging the contact points with the player inputs to the rest of the circuit. The touch points works as follows: when the player pushes a button, it presets the 8 clock counter that will allow 8 clock pulse cycles to clock the rest of the circuit. This is done with the carry out of the counter, as when the counter reaches 0000, the carryout goes low. The carryout is thus connected to the reset pin of the 555 timer, and holds the clock until the counter is preset asynchronously back to 1000.

Andrew speculated that while the button was being held, the 555 timer immediately starts clocking because the counter is not at 0000 anymore, but the player is still briefly holding the preset enable high, which means the counter is not clocking at that first pulse. He attempted to solve this problem by slowing down the input to the reset pin of the 555 timer. This solved the problem, and now player movement works as intended; this is displayed in the videos also in this folder. Now the game is fully functional; the player can move around properly, and hit detection is properly displayed on the upper right land LED. The videos of the player moving around both easy and hard mode, as well as hit collision, are all included in the overall shared folder.

Lessons Learned and Conclusion

We have gained a lot of knowledge and experience in electronics and circuit design through this project. Outside of learning how to better interact with chips and gaining fluency with building neat circuits and matrixes, we learned more methods to troubleshoot, how to systematically debug our circuits by identifying problems and applying corresponding solutions. We learned the importance of time management and starting our designing early to ensure that we leave plenty of time to debug and build our circuits. We also learned the incredible importance of checking our components to make sure that they are what they are marked to be, as well testing our circuit as we build it.

In the future, to improve and create more robust and less glitchy circuits, we will fully vet our designs and test them using simulations and also while building. For this project, our attitude was very cavalier and reckless, as we only properly tested the memory module and the set board data array before deciding to build out the surrounding combinational logic as fast as possible. We will use the experience with debugging and troubleshooting gained in this project to better debug circuits in the future.

Acknowledgements

We would like the acknowledge the following people for their help with this project:

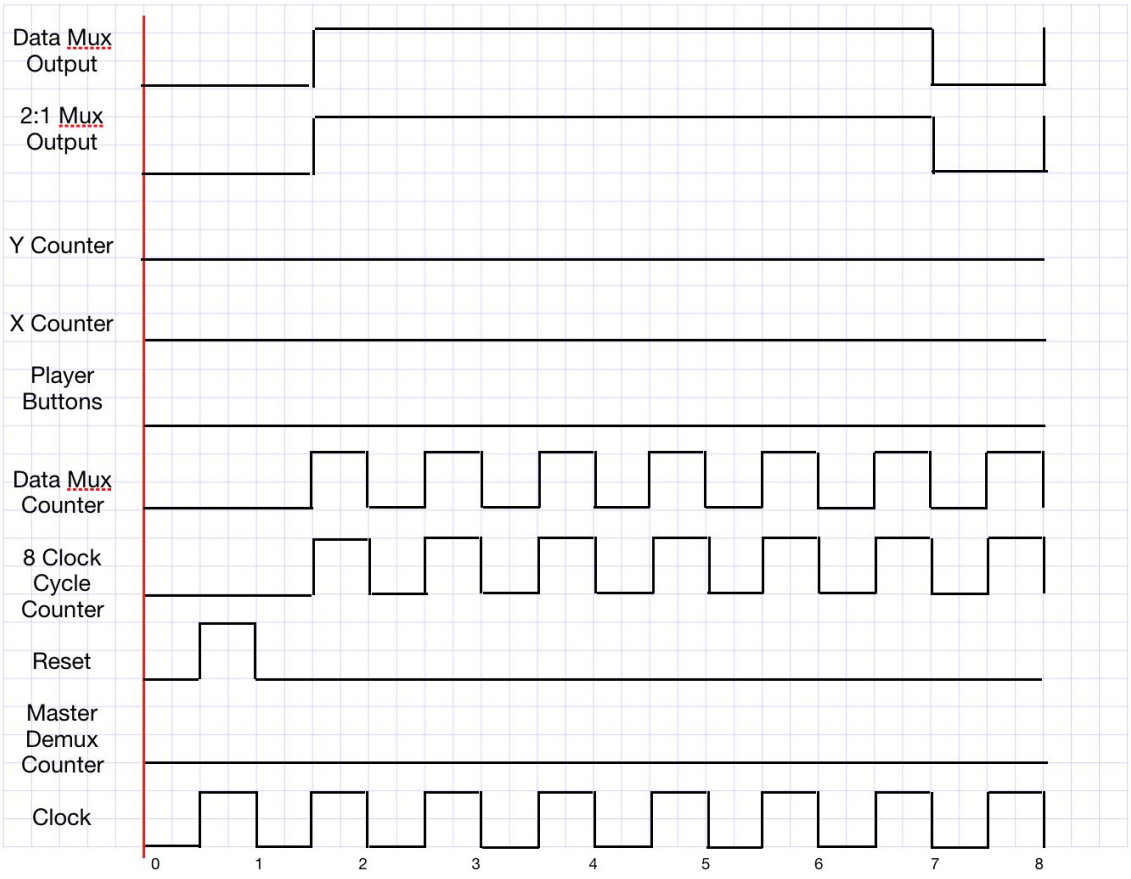
We would like to thank Senior EE, Brian, for his assistance with getting us components and helping us debug the off by one error of our circuit.

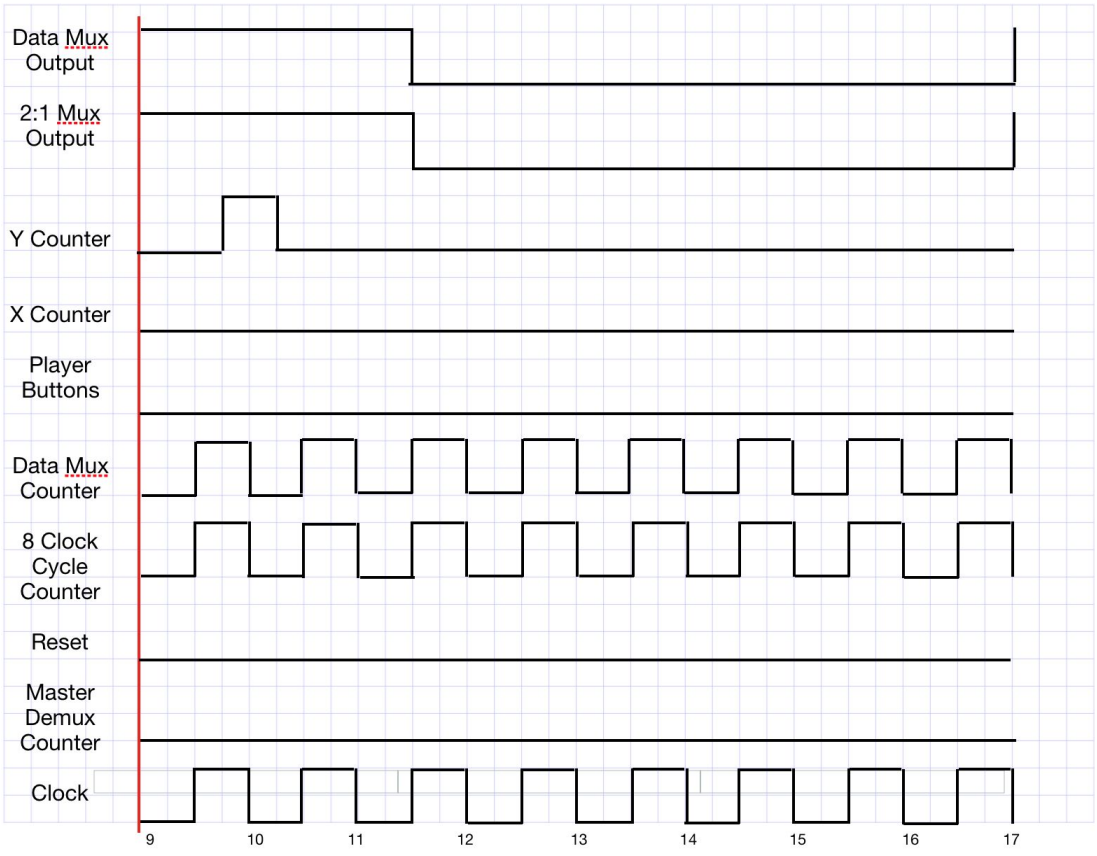
We would like the thank Dino Melendez for supplying us with the necessary ICs, protoboards, breadboards, and heat shrink to create this project.

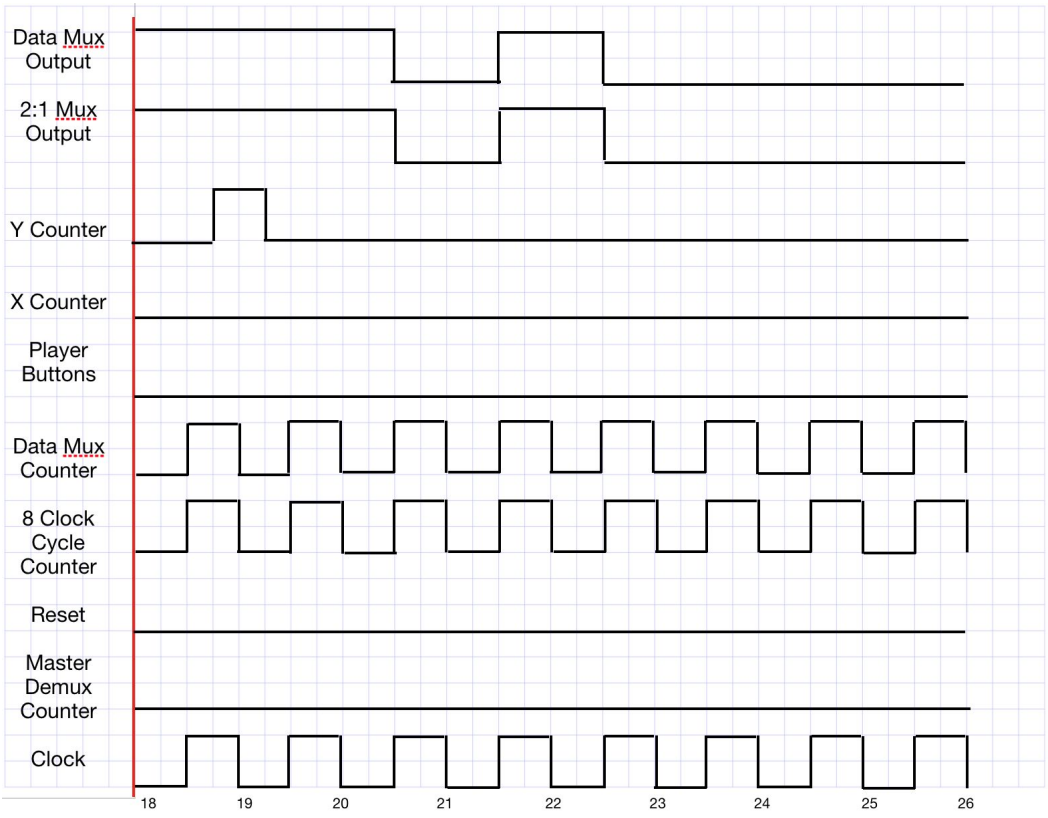
We would like the thank Nathaniel Kingsbury for his assistance with debugging the off by one error and for helping us to properly design the memory module.

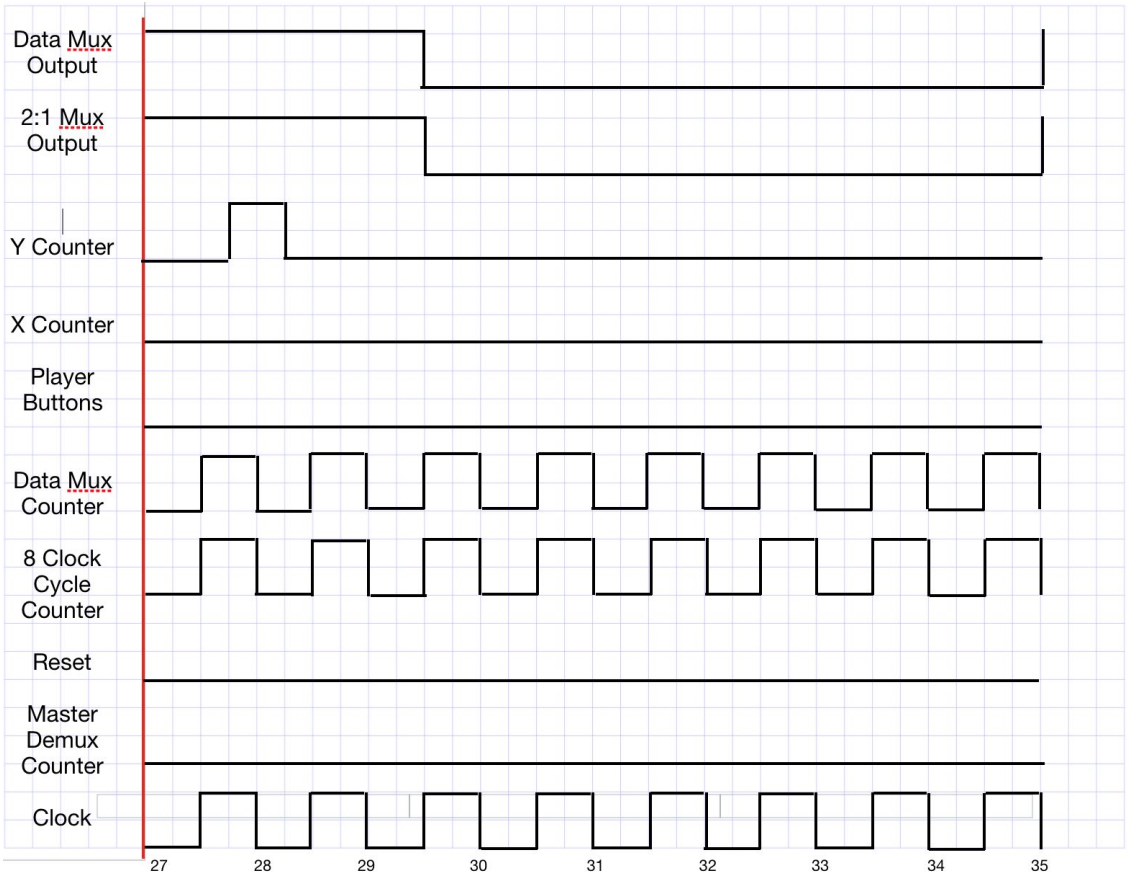
Finally we would like to thank Professor Risbud for supplying us with the knowledge to create a final project like this.

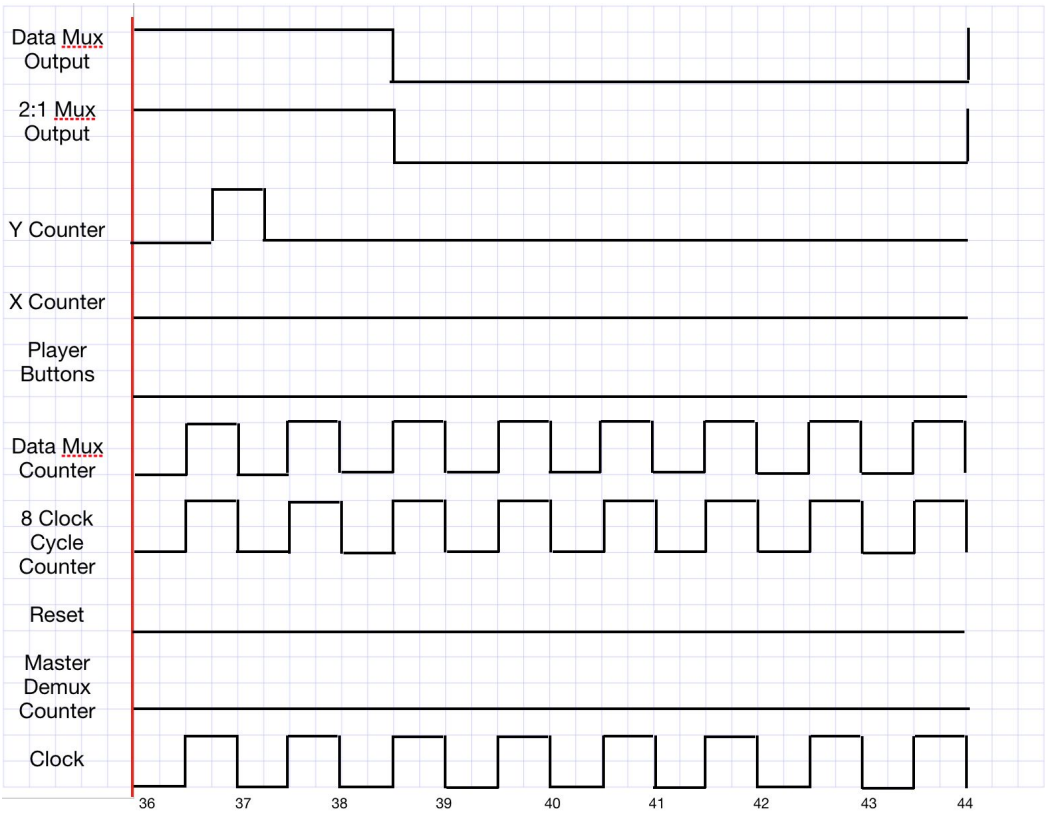
Timing Diagram

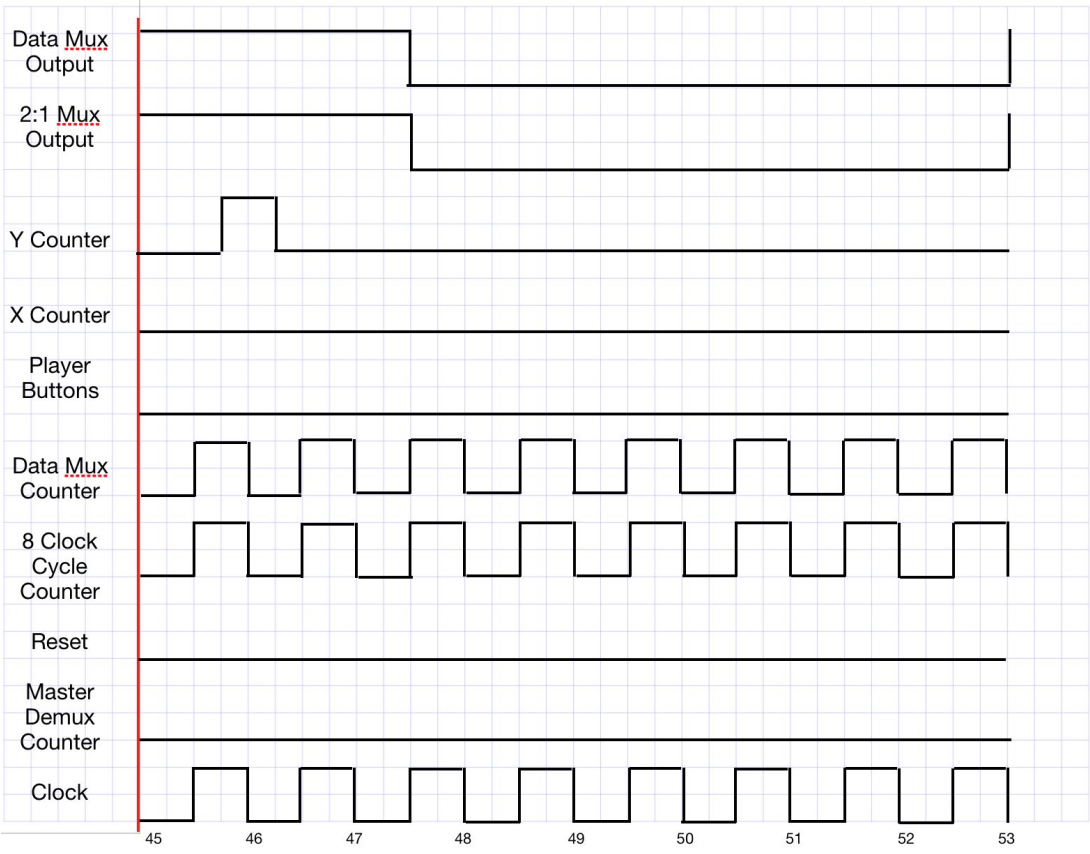


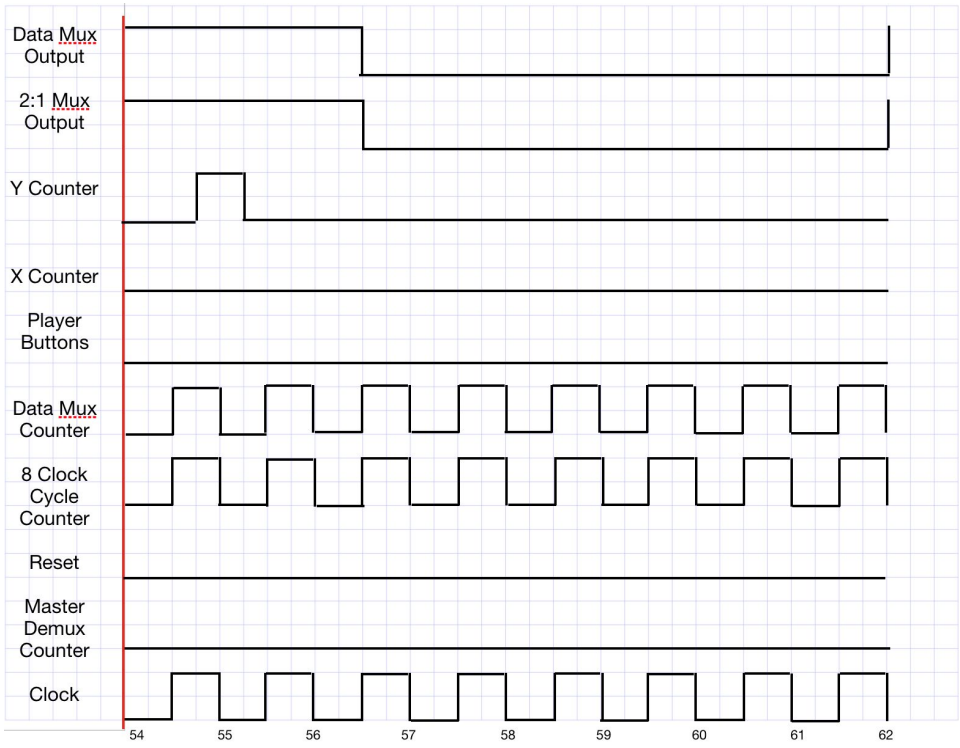


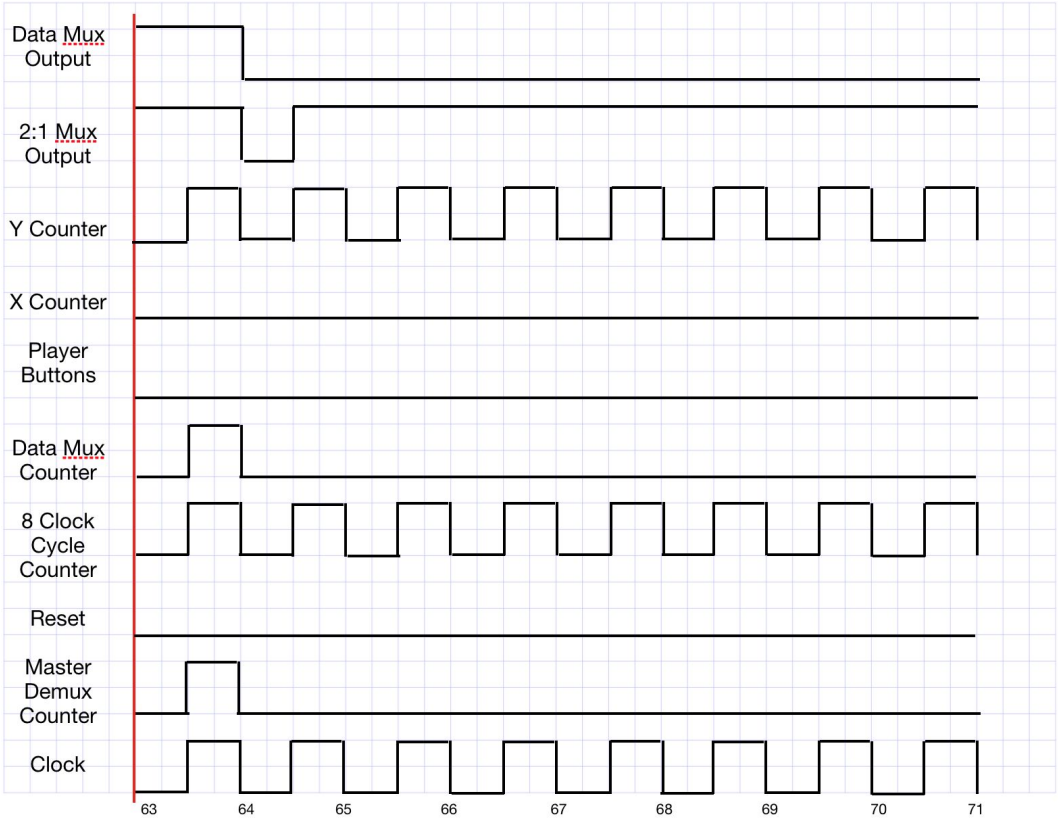

Data Mux Output
2:1 Mux Output
Y Counter
X Counter
Player Buttons
Data Mux Counter
8 Clock Cycle Counter
Reset
Master Demux Counter
Clock
18
19
20
21
22
23
24
25
26

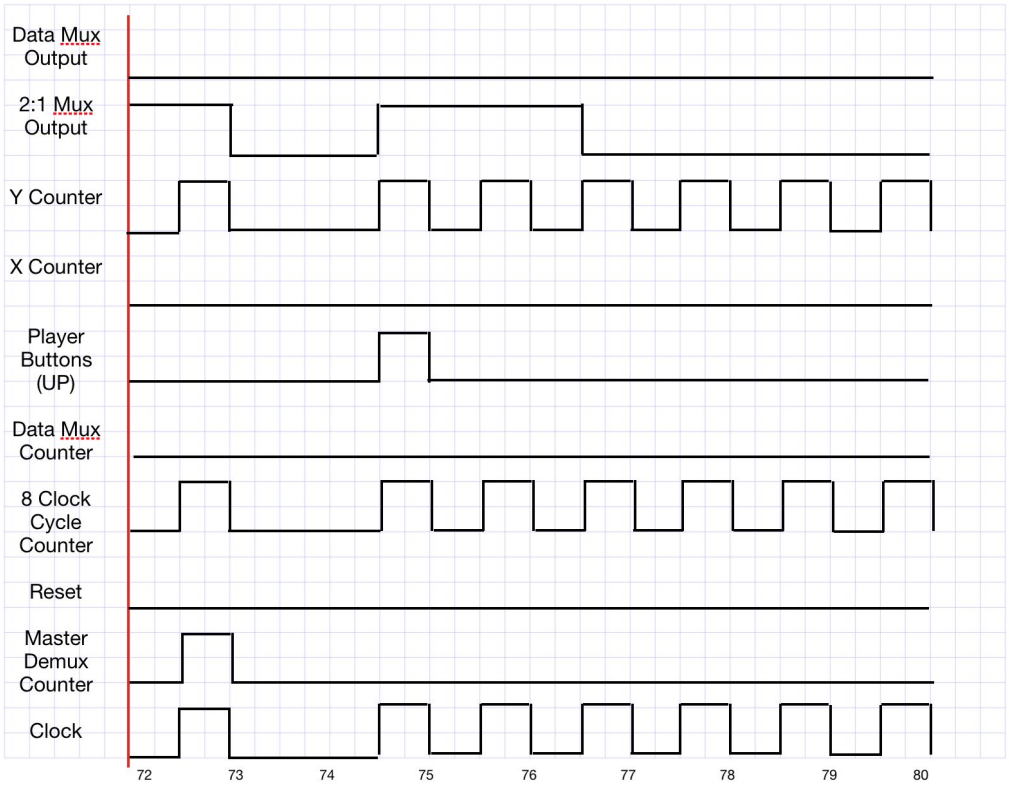

Data Mux Output
2:1 Mux Output
Y Counter
X Counter
Player Buttons
Data Mux Counter
8 Clock Cycle Counter
Reset
Master Demux Counter
Clock
27
28
29
30
31
32
33
34
35

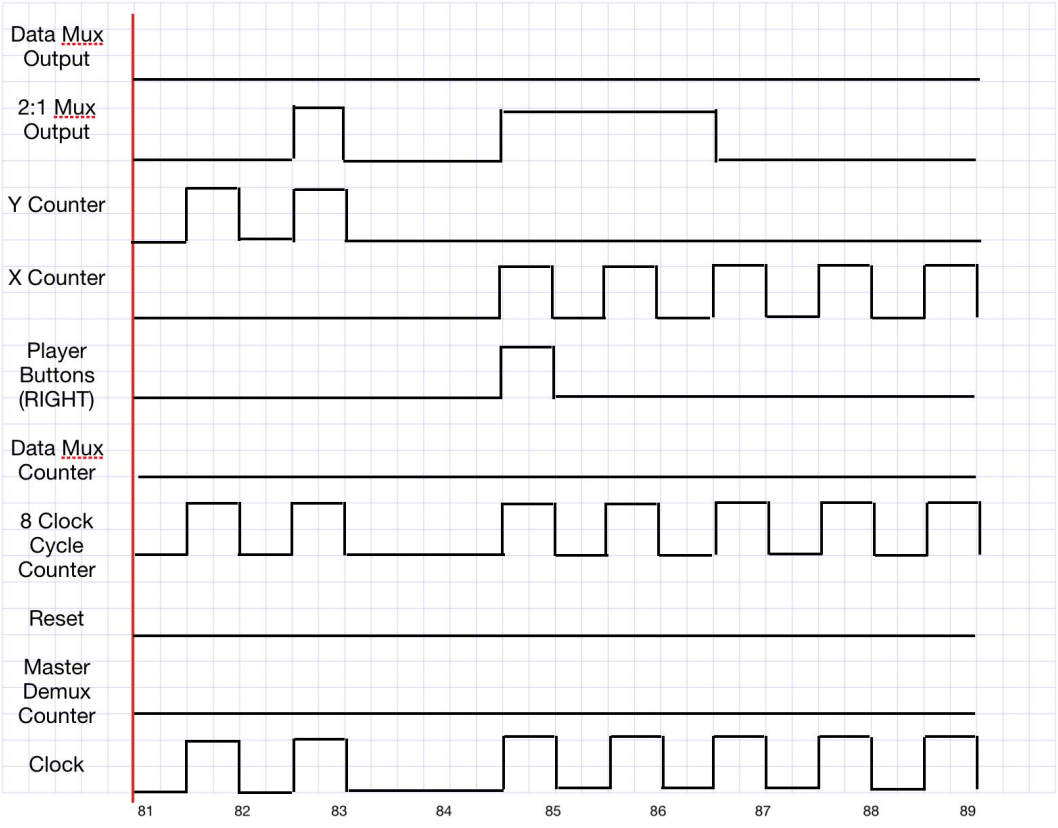

Data Mux Output
2:1 Mux Output
Y Counter
X Counter
Player Buttons
Data Mux Counter
8 Clock Cycle Counter
Reset
Master Demux Counter
Clock
36
37
38
39
40
41
42
43
44

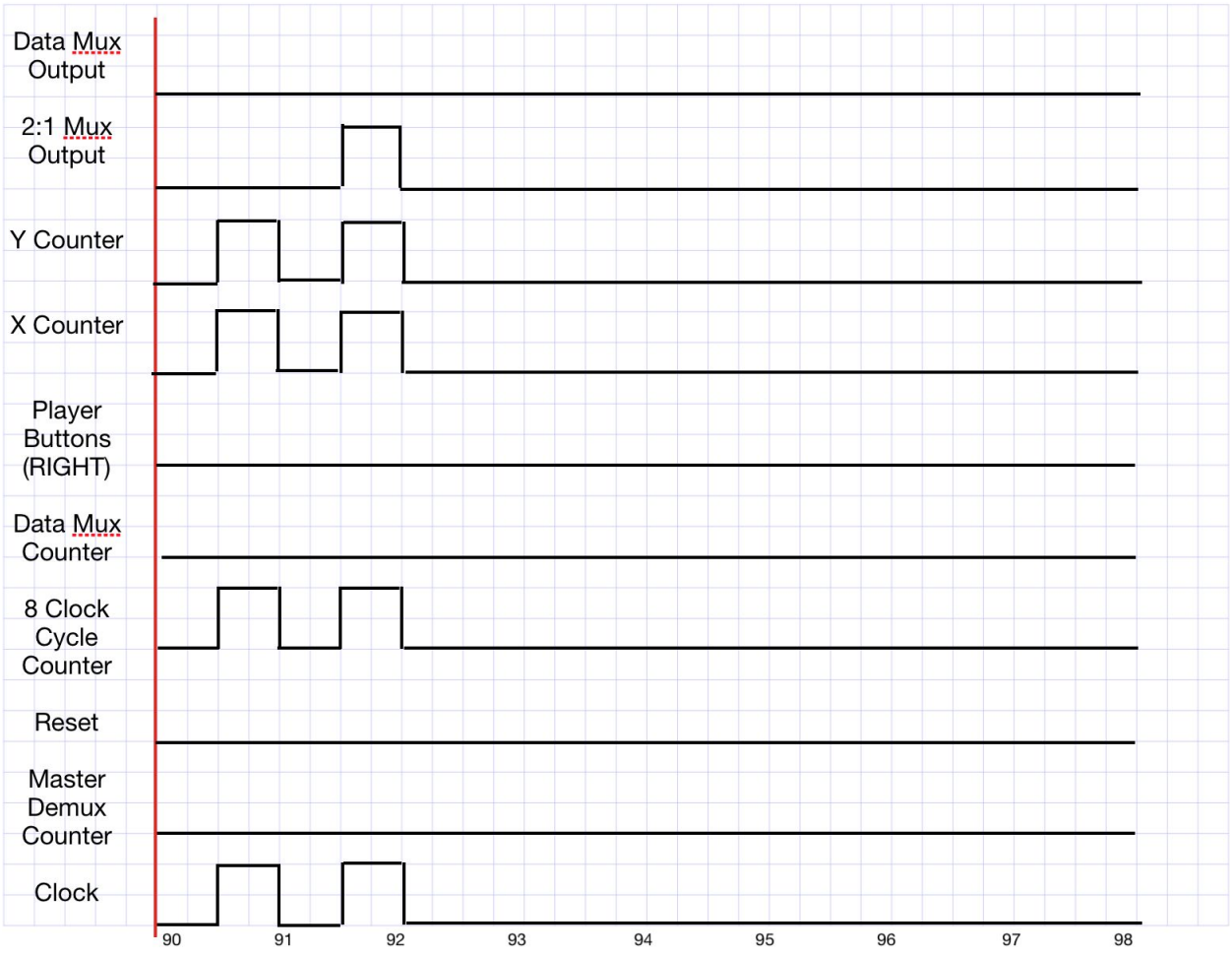

Data Mux Output
2:1 Mux Output
Y Counter
X Counter
Player Buttons
Data Mux Counter
8 Clock Cycle Counter
Reset
Master Demux Counter
Clock
45
46
47
48
49
50
51
52
53


Data Mux Output
2:1 Mux Output
Y Counter
X Counter
Player Buttons
Data Mux Counter
8 Clock Cycle Counter
Reset
Master Demux Counter
Clock
54
55
56
57
58
59
60
61
62


Data Mux Output
2:1 Mux Output
Y Counter
X Counter
Player Buttons
Data Mux Counter
8 Clock Cycle Counter
Reset
Master Demux Counter
Clock
63
64
65
66
67
68
69
70
71


Data Mux Output
2:1 Mux Output
Y Counter
X Counter
Player Buttons (UP)
Data Mux Counter
8 Clock Cycle Counter
Reset
Master Demux Counter
Clock
72
73
74
75
76
77
78
79
80


Data Mux Output
2:1 Mux Output
Y Counter
X Counter
Player Buttons (RIGHT)
Data Mux Counter
8 Clock Cycle Counter
Reset
Master Demux Counter
Clock
81
82
83
84
85
86
87
88
89


Data Mux Output
2:1 Mux Output
Y Counter
X Counter
Player Buttons (RIGHT)
Data Mux Counter
8 Clock Cycle Counter
Reset
Master Demux Counter
Clock
90
91
92
93
94
95
96
97
98