Digital Etch A Sketch with a Blinking Cursor

Hoseon Yu, Evan Dong
ECE150 — Digital Logic Design

December 21, 2025

Contents

1 Introduction **2**
1.1 Problem . . . 2
1.2 Approach . . . 2

2 Design and Method **3**
2.1 Design Choices . . . 3
2.2 Functional Block Diagram . . . 5
2.3 State Diagrams and Tables . . . 6
2.4 Truth Tables and Boolean Expression . . . 8
2.4.1 Matching Logic . . . 8
2.4.2 Toggle logic . . . 10
2.4.3 Reset logic . . . 11
2.5 Simulated Final Logic . . . 13

3 Implementation **14**
3.1 Interface . . . 14
3.2 Physical Implementation . . . 15
3.2.1 User Guide . . . 16
3.2.2 Labeled Implementation: IC's . . . 17

4 Conclusion **20**
4.1 Summary . . . 20
4.2 Limitations . . . 20
4.3 Possible Applications . . . 21

1 Introduction

1.1 Problem

The task was to create a digital etch-a-sketch that allows the user to draw on multiple canvases and create an animation. The user should be able to choose which canvas they are editing, and the canvases should be displayed for the user to see on an LED matrix of dimensions of at least 4x4. Additional considerations include the following:

- An LED array of 4x4 or greater.
- At least 4 available canvases.
- No more than two 2114 SRAM ICs (allowed 2 chips only if canvas greater than 4x4).
- A method to select and edit specific canvases.
- A method to automatically loop through the available canvases (loop mode).

Extra Credits:

- Larger canvas: (8x4 or 4x8) (10 pts)
- More canvases: $\geq$ 8 canvases (5 pts), $\geq$ 16 canvases (10 pts)
- For design projects: 0-20 pts awarded on a per-project basis

1.2 Approach

The circuit would distribute one bit at a time to the matrix using a system of MUXes at a high frequency. Because of this high frequency, the entire matrix, with all its stored inputs, would light up due to a phenomenon called persistence of vision. This MUX system would work by splitting canvas information into rows of 4 bits at a time, then stacking the rows (ordering them) according to a 3-bit address. Both the address and the input for what LED information on each row will be determined by a 5-bit up counter, where the three most significant bits will influence which row is being displayed, and the least two significant bits will control which LED in which row is displayed. The reason for sectioning each row into 4 bits of information at a time is that the lab's available RAM (2114 SRAM) has 4 words per address.

The movement of the cursor around the canvas will be implemented through four directional push buttons that move left, right, up, and down. An initialize button will also be implemented to set the cursor's location and place it at the bottom left of the canvas.

A finite-state machine will be implemented in order to control the sequence that toggles the specific LED that is selected (writing to the RAM) if and only if the toggle button has been pressed. Otherwise, the system will go through the normal addresses to display the image(reading from the RAM). Control the writing to the RAM and otherwise keep reading. Furthermore, the location of the blinking cursor will be tracked using universal shift registers, as discussed further in section 2.

2 Design and Method

2.1 Design Choices

The major design choice was how the user would interface with our circuit; the user will control the cursor on the canvas using 4-directional buttons: left, right, up, and down. To draw on the canvas, the user will press another button that toggles the bit (if it was high, it turns low, and vice versa). Another design choice was a cursor-on/off button to let the user turn off the cursor while playing the animation or just viewing the canvas.

The main idea that runs through most of the design of the project is that the SRAM that stores the information for all the canvases can only output four bits of data at a time. Therefore, to display a full 8x4 canvas, 8 unique addresses are required to store all the information in a single canvas. Using the three most significant bits to control these eight lines would be most beneficial, so the two least significant bits can be used to determine which bit on each line is activated. Adding these two together, for every full cycle of the counter, every LED in the matrix should have its value in the RAM displayed once.

One issue that was encountered is that mux can not read and write at the same time. In order to change one and only one indices in a row, the RAM needs to be able to take the information of the row and xor only one signal–inverting it. To work around this issue, a register was implemented outside the RAM so that data from the selected row can be stored in it. This way, when the RAM needs to write new data into itself, the register can read the data into the XOR gates while the RAM writes into memory. A series of 4 2:1 MUXes was used to select between the data that needs to be read and the temporary data stored into the register, which enables the ability to switch between reading and writing data since the read output and write input feeding into the RAM share the same bus (Reading and writing logic and system is further discussed in the subsection 2.3.

Controlling the data from the RAM and reflecting it one bit at a time on the LED matrix has its own logic as well. Beyond this, it also needs to be in sync with the read and write logic in the RAM in order to ensure the precise placement of data onto the matrix. To do this, a system of MUXes and deMUXes was implemented around the LED matrix. A 1:8 deMUX was used to feed the signal of either a low or the blinking behavior(a separate clock with frequency 4Hz) to 1 of eight rows on the LED matrix. The select pins for this MUX are taken from the three most significant bits of the clock that was previously mentioned. Then, a 4:1 MUX takes the output from the RAM-register MUX and sends it out as one signal, which goes into a 4:1 deMUX that redistributes one bit of data at a time to one of the 4 vertical inputs of the matrix. The selection pins of both these MUX and deMUX come from the least significant bits of the up counter. The combination of these has the effect of reading each bit in the matrix in sequence once for each complete cycle of the up counter.

To implement the dimmable canvas, the low of the 1:8 deMUX was replaced with a clock of high frequency, attached to a potentiometer that modulates the pulse width. The smaller duty cycle has the effect on persistence of vision that it makes the LED seem less bright, as

it is on for a shorter amount of time, even though it can't be seen by the human eye

Movement logic of the Cursor was implemented with Universal Shift Registers. The shift registers alow for a displayable shift of left or right by one LED of the cursor. By first loading 1 0 0 0(initialization) into the shift register, moving left and right, or up and down, can be represented as just moving the bits left or right. This approach was very intuitive, as it directly corresponds to the cursor's location.

The outputs of the Universal Shift Registers then go into two primary encoders to derive address signals for horizontal location and vertical location. These signals can then be compared with the counter signals using XNOR gates. The comparison for the moving vertical allows the clock to determine whether to pause itself during the writing process, while the AND of both signals configures the Vertical RAM MUX and determines whether to display blinking behavior.

The ability to switch canvases and create animations involves the extra addresses in the SRAM that are not used to signify data for a specific row. If three of the addresses are already used for row location on the matrix, extending the number of addresses used by one bit would create two canvases that exist for 1000 and 0000, which are two completely different numbers. Following this logic, the bits after the three least significant bits imply different canvases, and the number signified after these three least significant bits would address different canvases each. Therefore, adding 4 bits using dip switches after would lead to 16 different canvases. In order to go through the canvases sequentially–creating an animation– would mean controlling these four bits by a counter. Then, MUXing these two signals together gives the ability to create a JK-Flip-Flop button that switches between the animated display and targeted canvas editing.

2.2 Functional Block Diagram

The following functional block diagram was created based on the approach and design choices as mentioned on subsection 2.1.

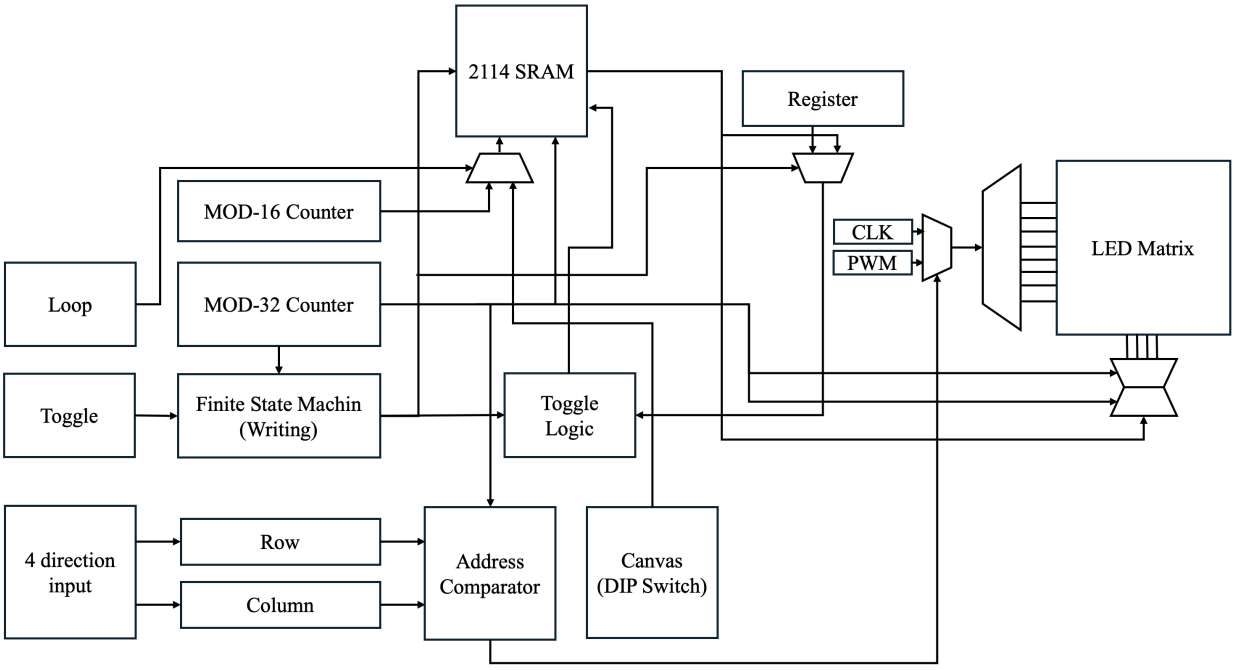


Figure 1: Functional Block Diagram

2.3 State Diagrams and Tables

The FSM (Finite State Machine) was used to interface with the 2114 SRAM. This FSM was used only upon pressing the Toggle Button. Match indicates whether the RAM address matches the cursor's row. In other words, it only escapes S_0 states when the Toggle button is pressed, and the cursor and the RAM address match up. Another Thing to mention is that the counter stops counting when the state changes from S_0 to S_1 and starts again when returning to S_0.

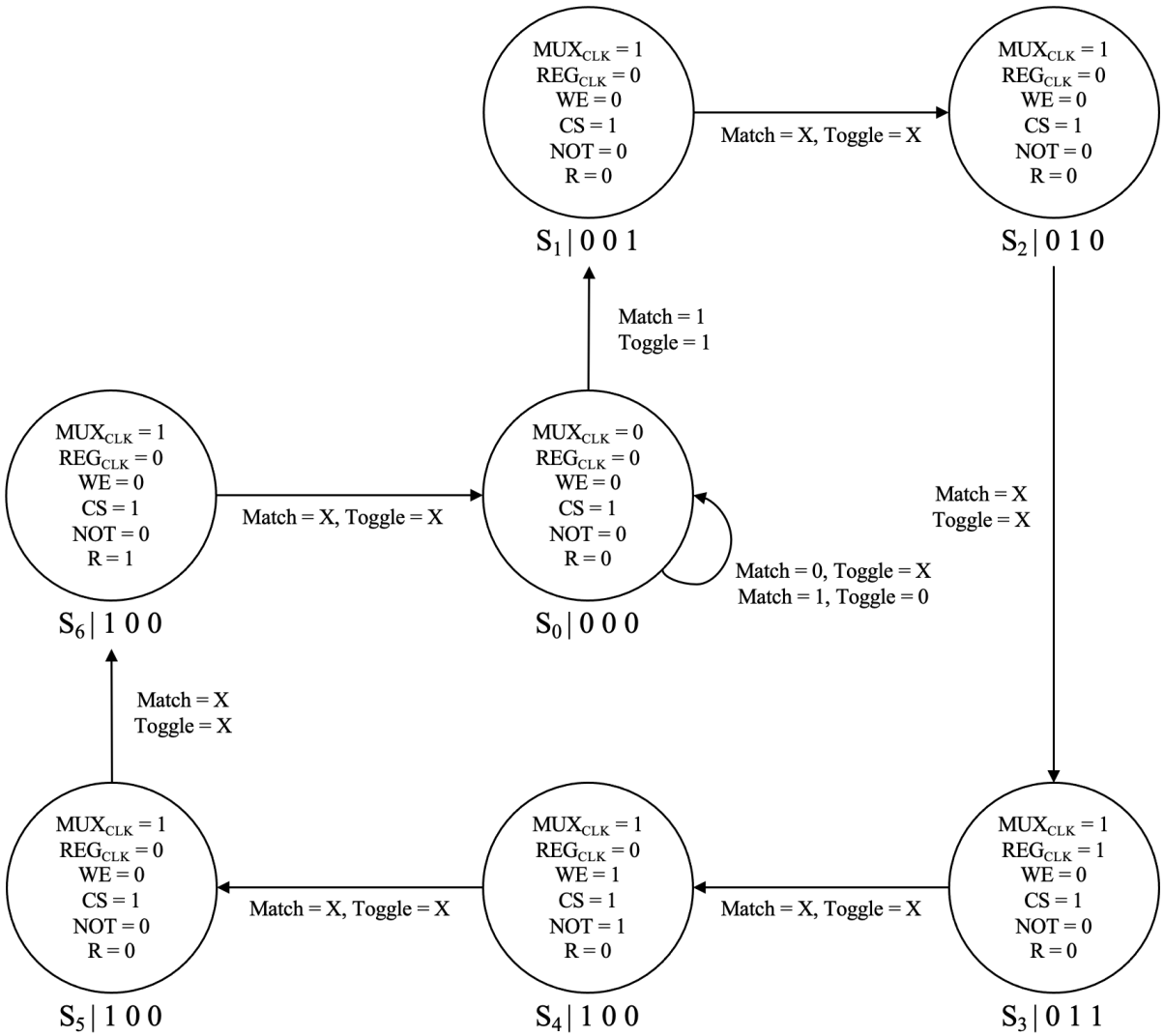


Figure 2: State Transition Diagrams

Symbols	Description
Match	Signal that checks if the address of the RAM is the cursor matches
Toggle	Button that toggles a bit
S_n	Current state
Q_{0-2}	Outputs of a counter
MUX_{CTR}	Selection Pin of a MUX (switches between Register output and RAM output)
REG_{CLK}	Clock of the Register
WE	Write Enable of 2114 SRAM IC
CS	Chip Selection of 2114 SRAM IC
S_{n+1}	Next State
E_{0-4}	Output of Encoder (based on input of Y and X axis Universal Shift Register)
R	Reset signal of FSM
X_{0-3}	Output of Cursor's X-Axis Shift Register
x	No-care
D_{0-3}	Output of MUX controlled by MUX_{CTRL}
I_{0-3}	Input of 2114 SRAM
Init	Initialize Button
U	Up
D	Down
L	Left
R	Right

Table 1: Reference Table for Symbols Used on section 2

The State Transition Table (Table 2) was created based on Figure 2. It was simplified by omitting the case Match = X, Toggle = 0 from the middle of the Finite State Machine to make it more intuitive for the reader, and because the case is not possible in a way that will be implemented in real life (with such a high frequency of a clock, it is humanly impossible to cause such a case).

Match	Toggle	S_n	Q_2	Q_0	Q_0	MUX_{CLK}	REG_{CLK}	WE	CS	NOT	R	S_{n+1}
0	x	S_0	0	0	0	0	0	0	1	0	0	S_0
1	0	S_0	0	0	0	0	0	0	1	0	0	S_0
1	1	S_1	0	0	1	1	0	0	1	0	0	S_2
1	1	S_2	0	1	0	1	0	0	1	0	0	S_3
1	1	S_3	0	1	1	1	1	0	1	0	0	S_4
1	1	S_4	1	0	0	1	0	1	1	1	0	S_5
1	1	S_5	1	0	1	1	0	0	1	0	0	S_6
1	1	S_6	1	1	0	1	0	0	1	0	1	S_0

Table 2: State Transition Table

2.4 Truth Tables and Boolean Expression

A few places that required some Boolean Expressions were checking the correct row and column of the cursor, the address of the AM, the toggle function, and the reset function for directional JK-Flip-Flops.

2.4.1 Matching Logic

E_4	E_3	E_2	Q_4	Q_3	Q_2	Z_1
0	0	0	0	0	0	1
0	0	0	0	0	1	0
0	0	0	0	1	0	0
0	0	0	0	1	1	0
0	0	0	1	0	0	0
0	0	0	1	0	1	0
0	0	0	1	1	0	0
0	0	0	1	1	1	0
0	0	1	0	0	0	0
0	0	1	0	0	1	1
0	0	1	0	1	0	0
0	0	1	0	1	1	0
0	0	1	1	0	0	0
0	0	1	1	0	1	0
0	0	1	1	1	0	0
0	0	1	1	1	1	0
0	1	0	0	0	0	0
0	1	0	0	0	1	0
0	1	0	0	1	0	1
0	1	0	0	1	1	0
0	1	0	1	0	0	0
0	1	0	1	0	1	0
0	1	0	1	1	0	0
0	1	0	1	1	1	0
0	1	1	0	0	0	0
0	1	1	0	0	1	0
0	1	1	0	1	0	0
0	1	1	0	1	1	1
0	1	1	1	0	0	0
0	1	1	1	0	1	0
0	1	1	1	1	0	0
0	1	1	1	1	1	0

E_4	E_3	E_2	Q_4	Q_3	Q_2	Z_1
1	0	0	0	0	0	0
1	0	0	0	0	1	0
1	0	0	0	1	0	0
1	0	0	0	1	1	0
1	0	0	1	0	0	1
1	0	0	1	0	1	0
1	0	0	1	1	0	0
1	0	0	1	1	1	0
1	0	1	0	0	0	0
1	0	1	0	0	1	0
1	0	1	0	1	0	0
1	0	1	0	1	1	0
1	0	1	1	0	0	0
1	0	1	1	0	1	1
1	0	1	1	1	0	0
1	0	1	1	1	1	0
1	1	0	0	0	0	0
1	1	0	0	0	1	0
1	1	0	0	1	0	0
1	1	0	0	1	1	0
1	1	0	1	0	0	0
1	1	0	1	0	1	0
1	1	0	1	1	0	1
1	1	0	1	1	1	0
1	1	1	0	0	0	0
1	1	1	0	0	1	0
1	1	1	0	1	0	0
1	1	1	0	1	1	0
1	1	1	1	0	0	0
1	1	1	1	0	1	0
1	1	1	1	1	0	0
1	1	1	1	1	1	1

Table 3: Truth Table for Y-Axis Comparison Logic

E_1	E_0	Q_1	Q_0	Z_0
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Table 4: Truth Table for X-Axis Comparison Logic

However, the comparison logic can be viewed as a list of XNOR gates, then passing them through an n-input AND gate. This is possible due to how XNOR is designed, as shown on Table 5. Thus, by doing a bit-wise XNOR operation (*e.g.* $\overline{E_0 \oplus Q_0}$), it was possible to achieve the same logic without a significant amount of work

A	B	Z
0	0	1
0	1	0
1	0	0
1	1	1

Table 5: Truth Table of XNOR

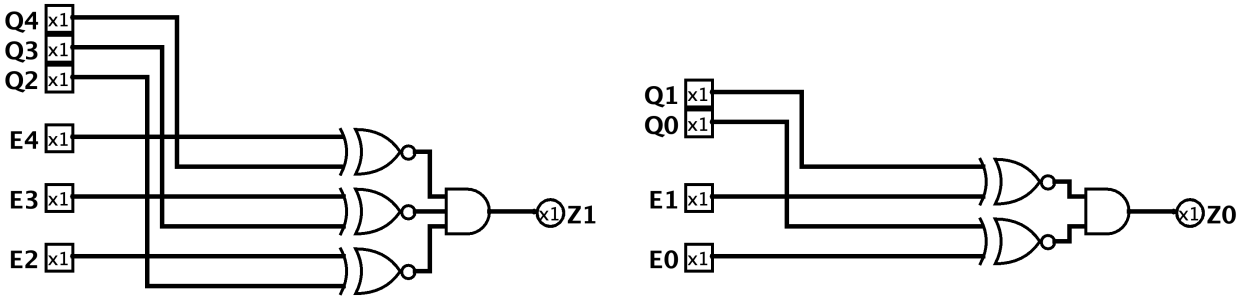


Figure 3: Simulated Matching Logic

2.4.2 Toggle logic

The basic Idea behind the Toggle Logic is that whenever the FSM (Table 2) outputs high on the *NOT* signal, it inverts the bit at the cursor. Since the FSM runs only when the row matches and the toggle button is enabled, the circuit only needs the cursor's X-coordinate to determine which column of that row to toggle the bit.

NOT	X_3	X_2	X_1	X_0	D_3	D_2	D_1	D_0	I_3	I_2	I_1	I_0
0	x	x	x	x	D_3	D_2	D_1	D_0	D_3	D_2	D_1	D_0
1	1	0	0	0	D_3	D_2	D_1	D_0	$\overline{D_3}$	D_2	D_1	D_0
1	0	1	0	0	D_3	D_2	D_1	D_0	D_3	$\overline{D_2}$	D_1	D_0
1	0	0	1	0	D_3	D_2	D_1	D_0	D_3	D_2	$\overline{D_1}$	D_0
1	0	0	0	1	D_3	D_2	D_1	D_0	D_3	D_2	D_1	$\overline{D_0}$

Table 6: Truth Table of Toggle Logic
(cases that can never happen are excluded)

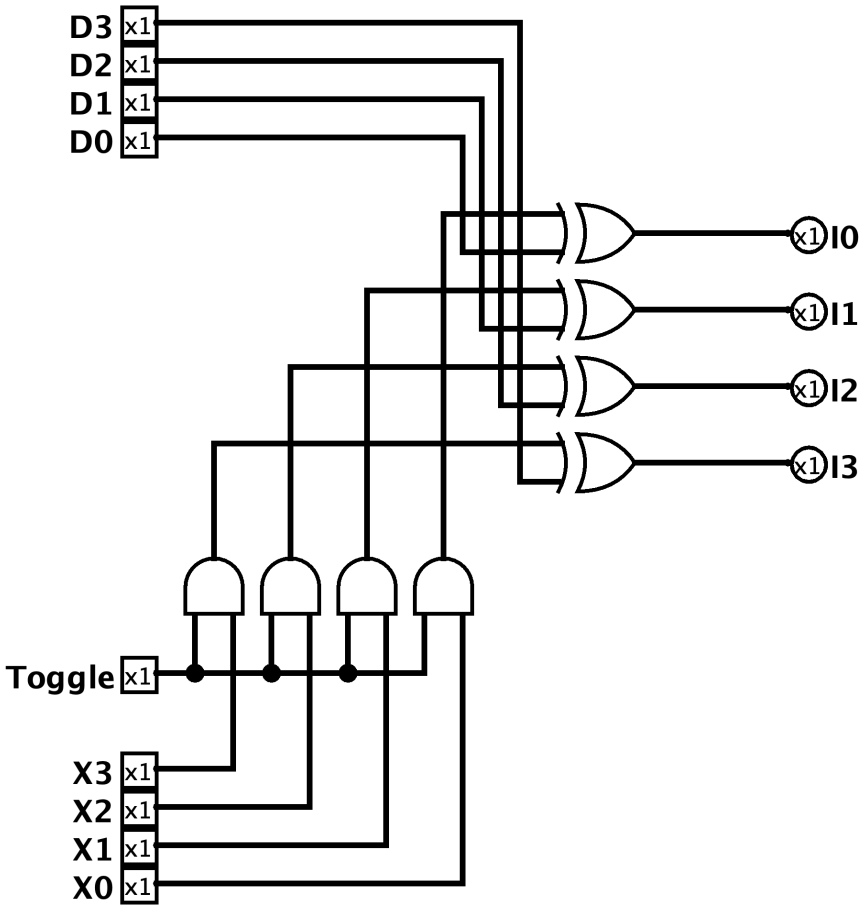


Figure 4: Simulated Toggle Logic

2.4.3 Reset logic

Before discussing the reset Logic, the reasoning behind it must be addressed first. While JK-Flip-Flops might be unnecessary, the goal was to make sure no inputs fail to register. This was done with JK-Flip-Flops for each button. Furthermore, the Initialize button was necessary to ensure that one bit is moved per axis, preventing circuit failures and providing additional functionality for the user.

Init	U	D	L	R	R_{n+1}
1	x	x	x	x	1
0	0	0	0	0	1
0	0	0	0	1	1
0	0	0	1	0	1
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	0	1	1
0	0	1	1	0	1
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	0	1	1
0	1	0	1	0	1
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	0	1	1
0	1	1	1	0	1
0	1	1	1	1	0

Table 7: Truth Table of Reset Logic

This Truth Table, however, can also be simplified in a similar way as subsubsection 2.4.1. By inspecting the truth table, the pattern is quite clear.

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	0

Table 8: Truth Table of XOOR

When both buttons that are on the same axis are high, it returns 0. But if only either of them is high, it returns high. Also, there is a special case where if the input is high, it returns high. This can be represented as

$$\text{Init} \; + \; \Big[(U \oplus D) + (L \oplus R) + (UDLR)\Big]. \tag{1}$$

Since this has to happen after the next clock cycle to give room for the Universal Shift Register to correctly capture the input, an additional D Flip-Flop was necessary after Equation 1.

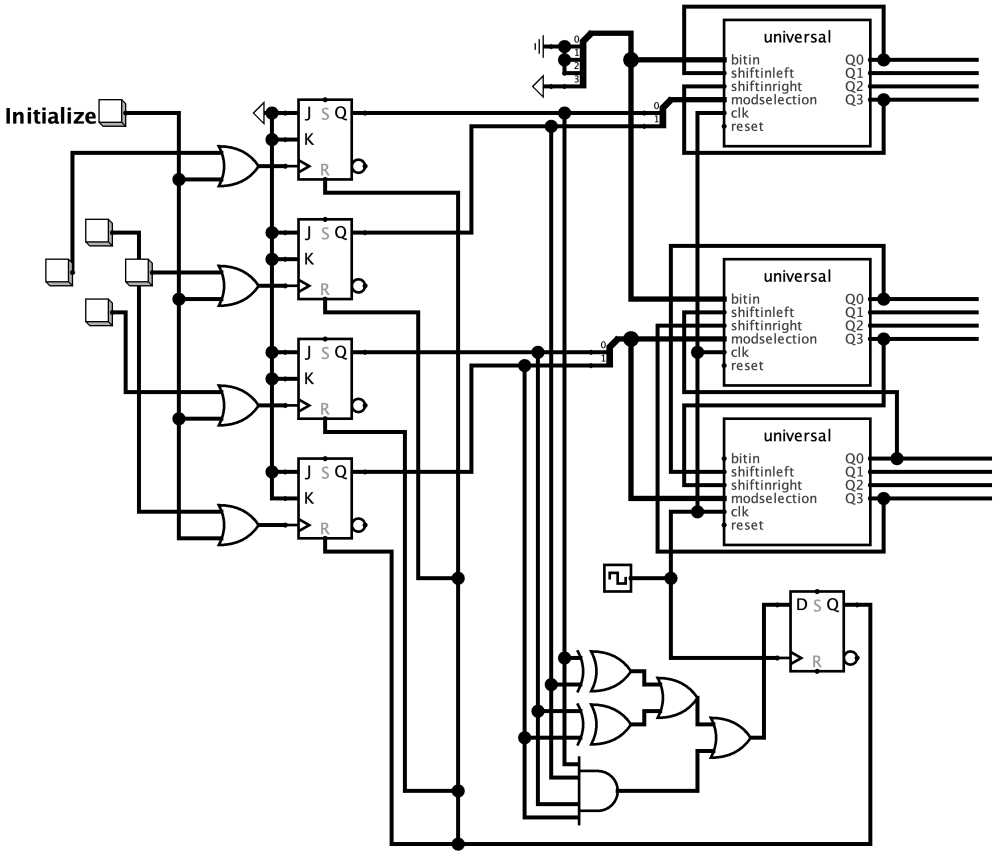


Figure 5: Simulated Reset Logic with Buttons and Universal Shift Register

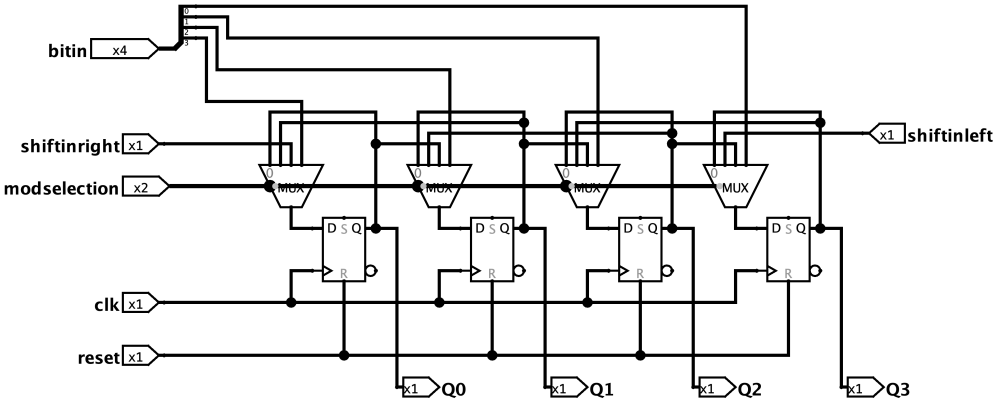


Figure 6: Internal of Universal on Figure 5

2.5 Simulated Final Logic

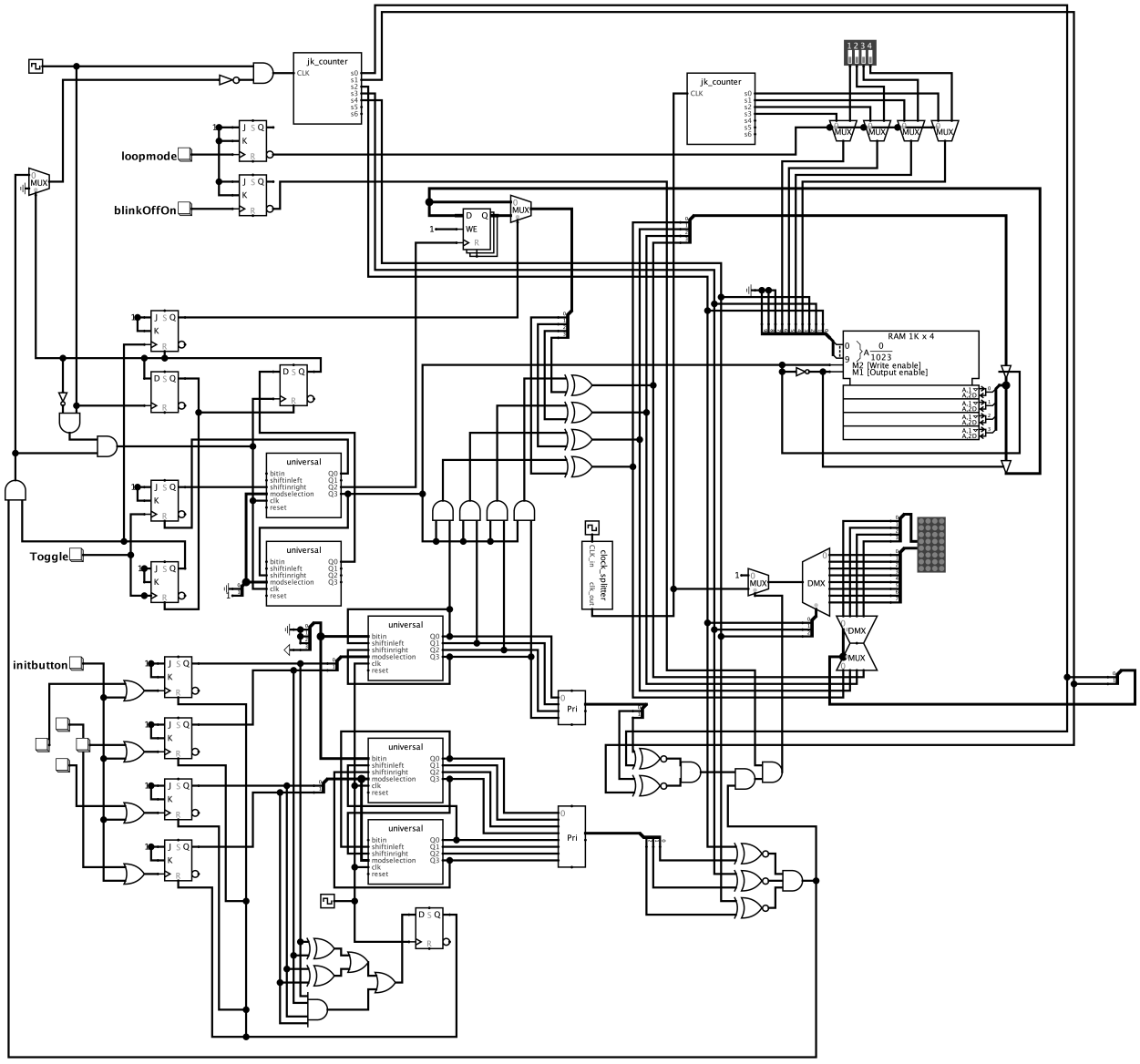


Figure 7: Simulated Logic with Logisim Evolution

3 Implementation

3.1 Interface

The Physical circuit was designed in such a way that it keeps the controls of the canvas intuitive. Buttons that are closely related to each other are put near each other for convenience, such as the moments up, down, left, right, and toggle button at the bottom. Buttons that are less related are isolated, such as the toggle-on/off buttons and the dim canvas knob.

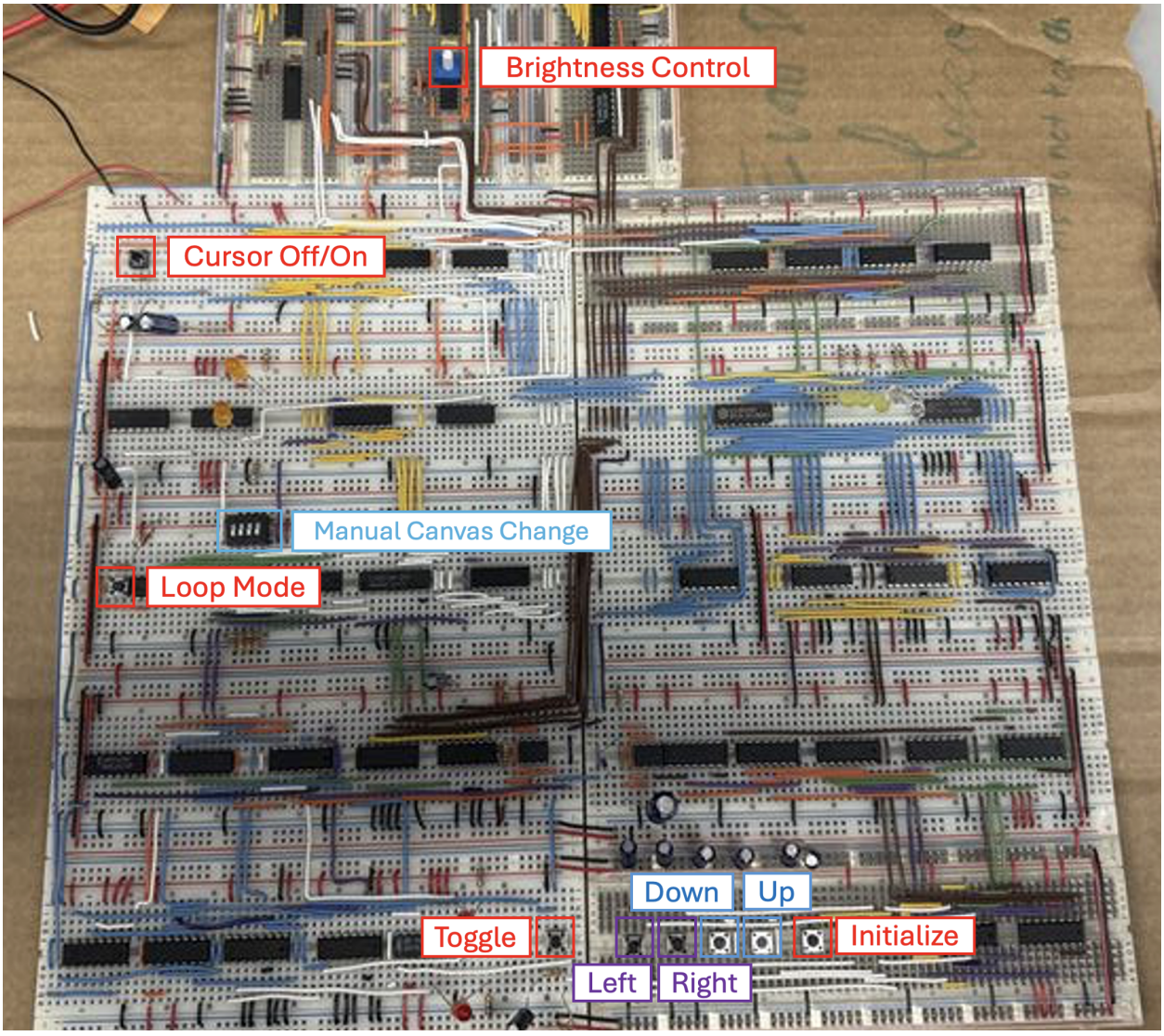


Figure 8: Circuit with Labeled User Interface

3.2 Physical Implementation

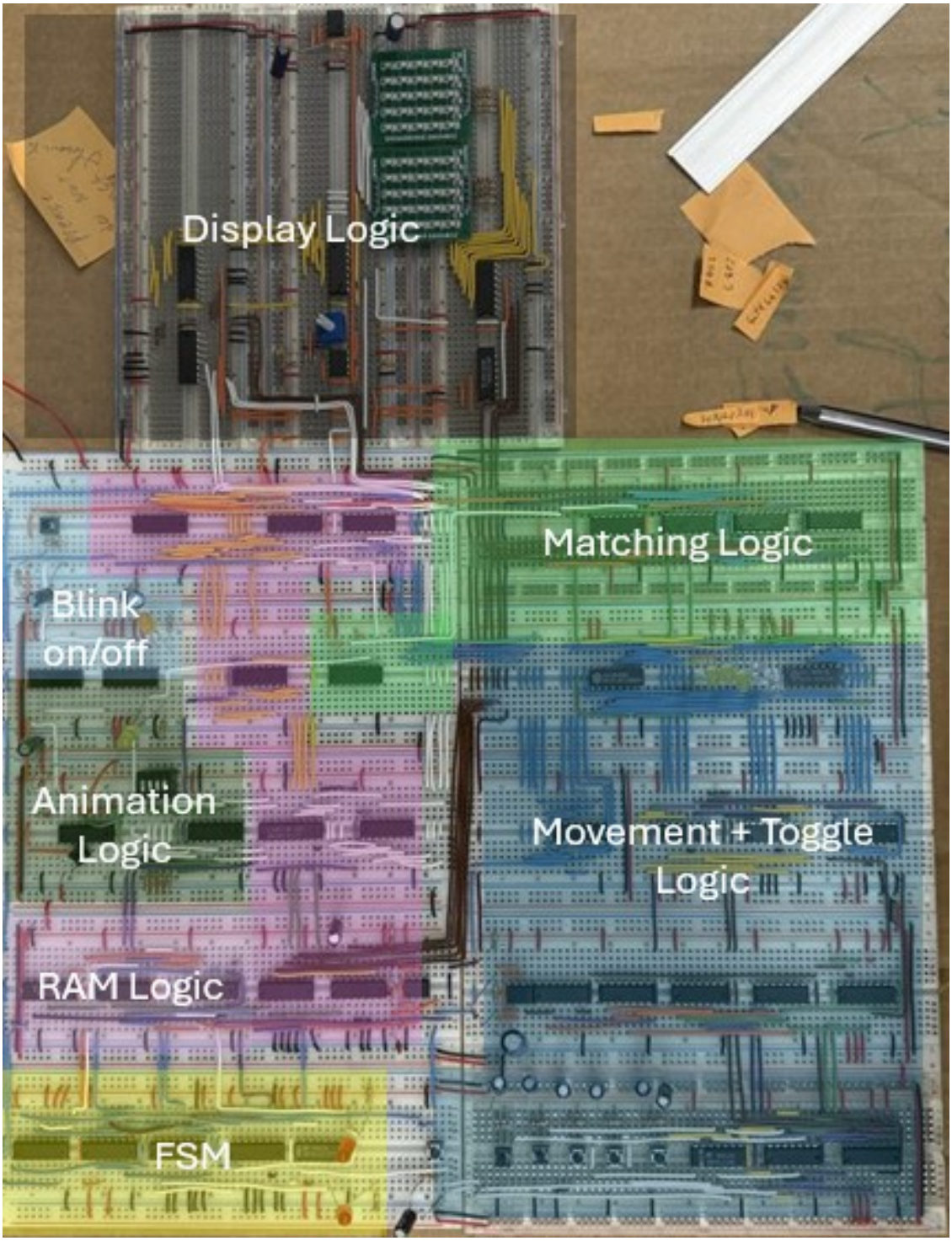


Figure 9: Labeled Sub-Circuits

The physical implementation closely resembles the Logism Evolution diagram (Figure 7), with a few minor differences. LEDs were added around the circuit to make the circuit more easily debugged and iterable. Two red LEDs were used next to the toggle button attached to the corresponding JK-Flip-Flop to indicate when the user can write. In this case, the user can take another in the circuit when both the LEDs have turned off. LEDs were also used to indicate where the cursor should be on the matrix–the yellow LEDs signify which row in the matrix is selected, while the blue LEDs indicate the column position. Also, indicator orange indicator LEDs were implemented near the pause blinking and cycle reading buttons in order to be able to tell when they turn on.

Many of the NOT gates in the circuit diagram were able to be skipped in implementation by scouring the NOT signals on the ICs. Additionally, the JK-Flip-Flop implemented in the FSM is positive-triggered, but that issue was resolved by sourcing the signal into a NOT gate in a nearby inverter with an open input.

Rather than only three clocks implemented in the circuit diagram in Logism Evolution, four were implemented in the circuit. One was used to implement the blinking behavior, another was used to clock the RAM logic and FSM, another was used to clock the universal shift registers, and the final one was implemented to enable dimming the canvas.

3.2.1 User Guide

1. Press the Initialize Button to initialize the cursor position.
2. Choose the Canvas through the DIP switch.
3. Use directional movement buttons to move the cursor around the canvas.
4. Push the toggle button to change the LED at that location to the opposite state.
5. Turn off the cursor by pressing the labeled button.
6. Play the animation by pressing the Loop Mode button.
7. Control the Brightness with a Potentiometer

3.2.2 Labeled Implementation: IC's

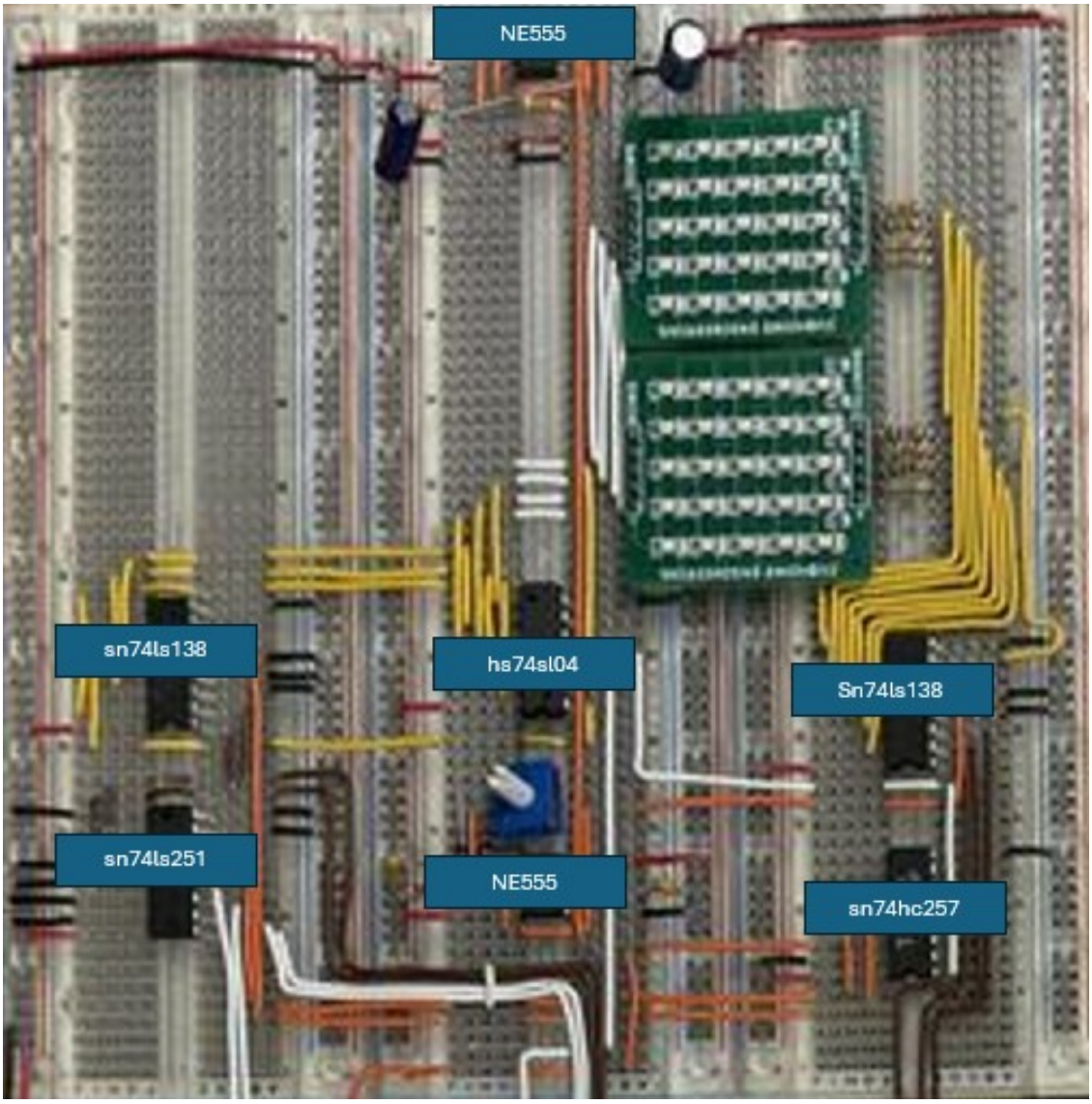


Figure 10: Labeled ICs Top

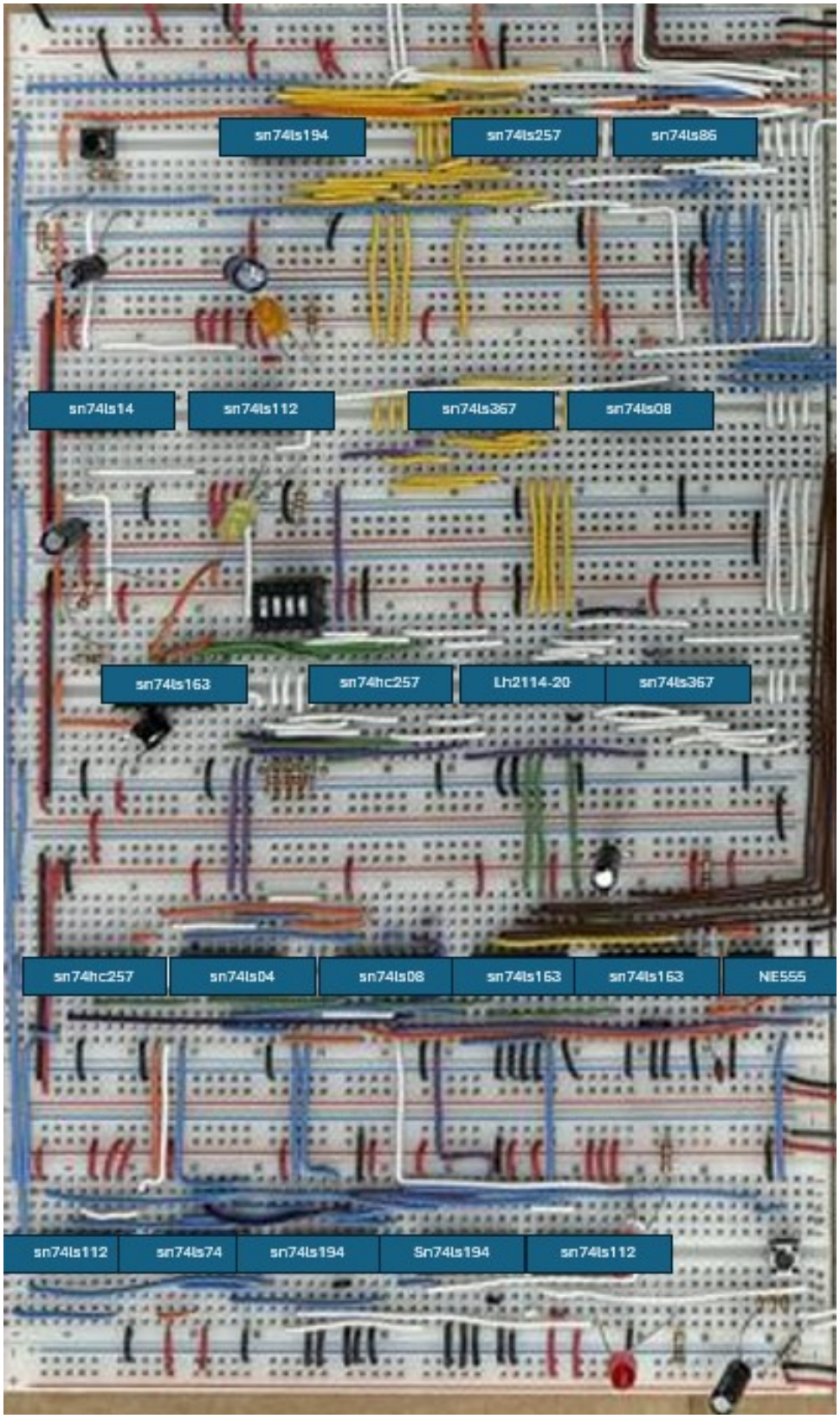


Figure 11: Labeled ICs Left

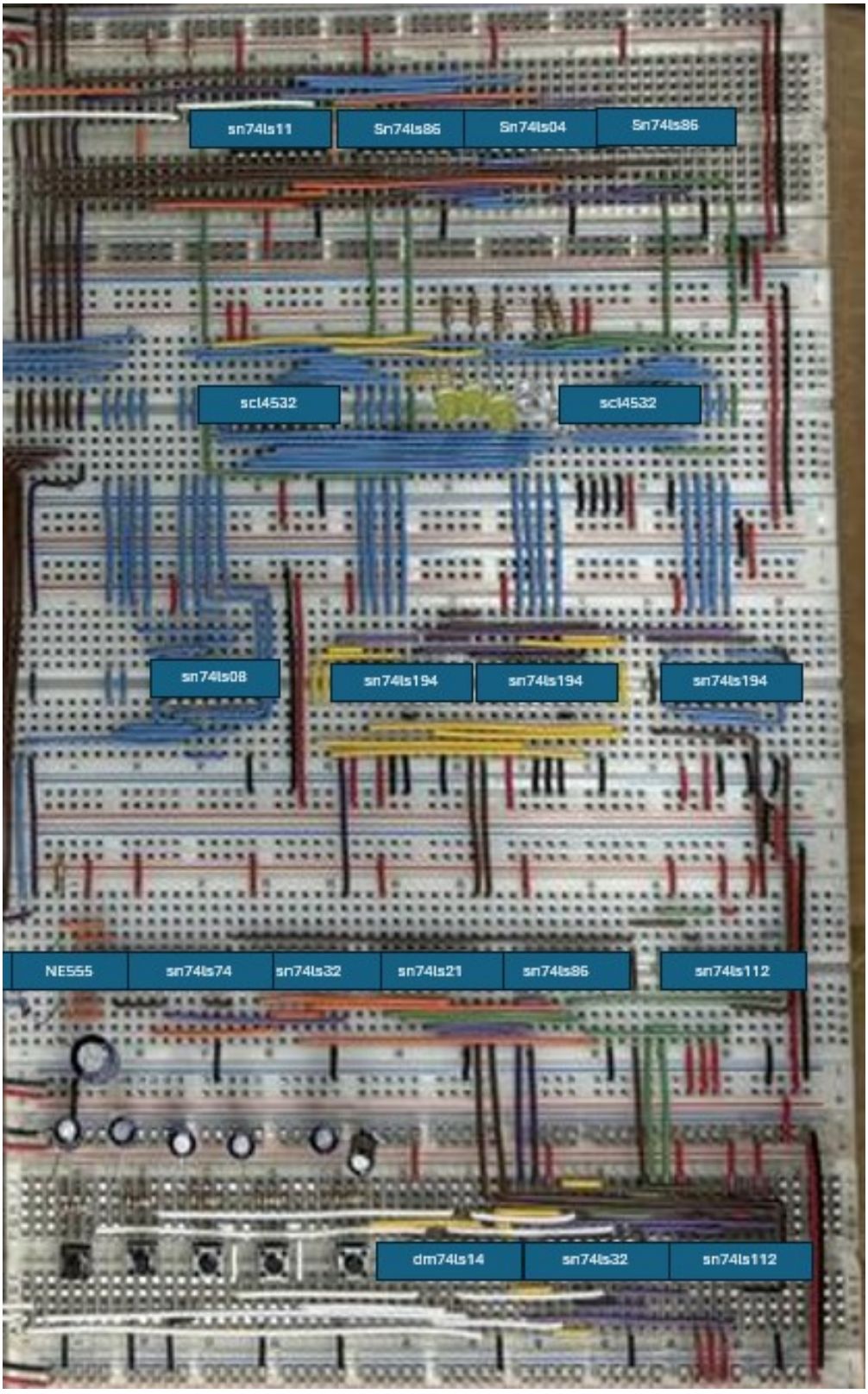


Figure 12: Labeled ICs Right

4 Conclusion

4.1 Summary

The objective of this project was to design and implement a digital drawing canvas system that supports multiple canvases and animation on an LED matrix using a maximum of two 2114 SRAM ICs. To address this problem, a digital Etch A Sketch system was implemented.

Each canvas's row was stored in the first three bit addresses of 2114 SRAM and displayed at high frequency to produce a canvas to achieve persistence of vision while still reading 4-bit words at a time. Cursor movement and buttons were implemented using JK flip-flops, universal shift registers, and priority encoders. Toggling a bit function was implemented using a comparison-based XNOR logic and a Finite State Machine to correctly interface with the SRAM. Multiple canvases were made by using more significant bits of the SRAM. The Loop Mode was implemented using a MUX, with the inputs being a DIP switch or the output of a counter.

4.2 Limitations

The way FSM (subsection 2.3) was implemented allows the user to click Toggle between cycles, which can break the entire circuit. As previously mentioned, this is very unlikely to happen, but with this many canvases and the time that the user has to spend, it is not guaranteed that such behavior will never occur during the operation.

Furthermore, the artifact of this is the fact that the circuit needs to be turned on and off multiple times to be in the correct state before operating correctly. This requires either an additional button or re-implementing the FSM to put the system in the proper state or to avoid such behavior, which is extremely important. While LED makes it very easy to tell, this is still a problem that can impair users' quality of life.

As shown in Figure 7, three of the address pins are grounded. This means the maximum of 111 canvases are unused, even though our design was intended to conserve every bit of the 2114 SRAM IC. Furthermore, the Loop Mode only traverses a set number of canvases, not the number the user selects. This means the user must manually make the canvases blank or live with it. Making the counter programmable would be a nice touch. This causes another problem: the canvas cannot be cleared. Especially with the digital Etch A Sketch method of canvases, erasing the entire canvas is very time-consuming, which is a nice feature this system lacks.

Talking about animation, another limitation that the system has is not being able to choose how the loop will happen. Features like the ability to select count up, count down, or ping-pong counters are not possible on our circuit, but can still be easily implemented. Furthermore, the animation speed is set to a constant frequency of 3-4Hz, as it shares the same clock as the cursor. Being able to control how fast the animation progresses through another potentiometer is another essential feature missing from an animated drawing canvas.

4.3 Possible Applications

The project can be separated into a few parts: a Driver that interfaces with the LED matrix, cursor logic, blinking logic, and an FSM that keeps reading until an action occurs. The LED matrix driver can be used as a general-purpose display controller with possible scalability by extending the counter and adding more rows and columns. The same design can be used for a higher-resolution system, such as larger displays for other embedded systems.

The system can be used to create a larger animating system. More RAM memory addresses can be implemented followed by corresponding dip switches. In doing this, the system would be able to support different colors and higher resolutions which can yeild more complex animations. subsection 4.2.

Also, the finite state machine that interfaces with the 2114 SRAM IC allows continuous reading of stored data. It also allows modifying a specific bit in RAM based on the user's input. This behavior closely resembles the configuration system in larger systems, where data is held and reused until it is modified. Thus, the FSM can be reused as a generic control logic for updating a system that uses RAM as data storage.

Furthermore, the cursor logic is a very intuitive system that can be and is widely used when human and machine interaction is necessary. It gives an intuitive interface, especially when the GUI is grid-based. The independent blinking behavior can serve as a visual indicator and be used alongside the cursor system to provide the user with more information without adding too much complexity. Examples include warnings, system status, and a visual indication of selection.