Project 2: RAM

Aidan Wong

ECE-150 - Digital Logic Design

November 20, 2025

1 Introduction

This project involves designing and implementing a sequential logic circuit that can write a byte into a single, static, volatile 4 bit-per-word (bpw) SRAM chip and then read the byte using parallel-out registers upon clicking their respective debounced read or write buttons.

1.1 Specifications

- Uses a single 2114 SRAM IC.
- Provides data inputs via DIP switches.
- Uses no more than four breadboards.
- Uses red wire for 5V and black for ground.
- Continuously displays the contents of external parallel-out registers on LEDs.
- All buttons are debounced.
- TTL to CMOS interfaces properly accounted for

1.2 Approach

This implementation uses a finite-state Moore machine to read and write data to the SRAM. It includes an idle state and enters a different "loop cycle" depending on the button pressed (either read or write). This was created with several D flip flops and exclusively TTL chips that are able to "count through" these states.

1.3 Report Structure

This report will cover the RAM design process through three main sections: *Methods*, *Implementation*, and *Conclusions*. The *Methods* section explains the theory behind interactions with RAM and finite-state-machines, state diagrams and transition tables used, relevant boolean expressions and logic diagrams, and functional block diagrams. The *Implementation* section explains how the circuit was constructed physically, the choices made to optimize the number of chips and the wire paths, and additional circuit functionality to improve user experience. It will also display the final breadboard with labeled IC chips and sub-circuits and decisions made to improve the user experience. The final *Discussion and Conclusions* section is a reflection of the final product, detailing limitations and how this project can be used in larger systems/circuits.

2 Methods

This circuit was dependent on solving key issues in controlling the input, RAM chip, and outputting our results. These issues were addressed by using a finite-state machine in combination with registers.

Issues (Summary)

1. **Input:**
   Due to our 4-bit RAM limitation, our implementation needs a way to split the 8-input (byte) into two 4-bit chunks and write the RAM in sequence. This also needs to be done in a way that does not cause conflict with the timing diagram of the RAM (Shown in Figure 1).

2. **RAM Chip:**
   There needs to be a way to retain the input information in the RAM chip and read it back without taking in gibberish or losing parts of it.

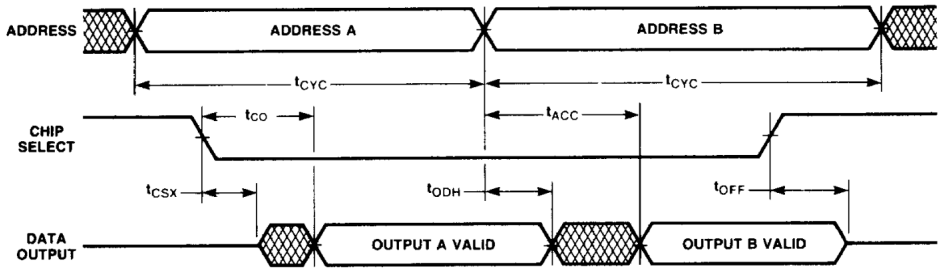
3. **Output:**
   There needs to be a way to output the result on LEDs after the inputs pass through the RAM chip.

2.1 Finite-state Machines

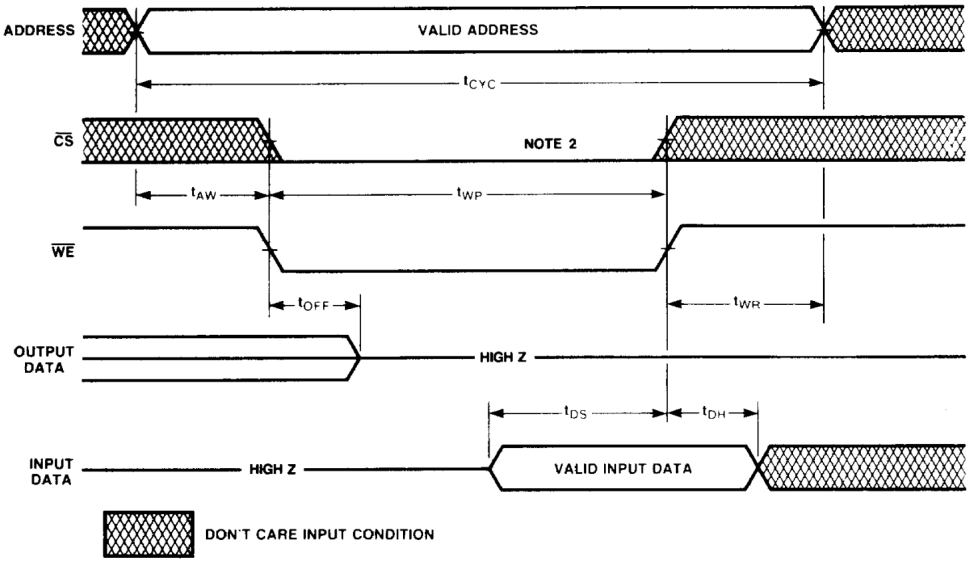
A finite-state machine is a tool that defines a circuit as a series of states that are dependent on clock pulses. It is generally useful when a set of instructions must be completed in sequence. This tool is vital for interfacing with a SRAM chip due to the propagation delay that is inherent to the chip when enabling and disabling chip select (CS) and write enable (WE). By creating individual states for enabling and disabling these pins, propagation delay can be circumvented.

Although this implementation employs a finite-state machine that considers the changes in these states, the timing intervals in the 2114 TTL RAM chip are measured in nanoseconds and would not be impacted by any clock built in the lab (as it is not possible to build a clock fast enough to run into these issues). In short, the timing diagram could have been ignored entirely and there would have likely been no issues.

Read Mode Timing Diagram, Note 1



Write Mode Timing Diagram



Notes

1. $\overline{WE}$ must remain HIGH during READ cycles.
2. t_{WP} is measured from the falling edge of either $\overline{CS}$ or $\overline{WE}$ (whichever is the last to go LOW) to the rising edge of either $\overline{CS}$ or $\overline{WE}$ (whichever is the first to go HIGH).

Figure 1: Timing Diagram of the TTL 2114 SRAM Chip

2.2 Registers

It is not possible to display 8 values using a 4 bit-per-word RAM chip. To allow the display of 8 values without changing the RAM chip, we use two 4-bit parallel out registers. These external registers are able to hold data and display what they are holding, which makes it ideal for this project. In addition, these registers are isolated from what is being written into the RAM, which allows a constant LED array even if new things are being written into the RAM (meaning the values won't be changed until the read button is pressed).

2.3 State Diagram

This finite-state machine is defined by three modes of operation: read, write, and reset or idle.

The reset or idle state (S_0) is the state that the circuit remains in when no inputs are being read from the read or write buttons. This is also the state that the circuit will return to after completing the read or write operations.

The writing mode is entered whenever the write button is pressed and the clock value is on the rising edge. If the clock is not on the rising edge when the button is clicked, the JK-flip flops used will hold the value and "enter" it at the next rising edge. This "mode" spans 3 states.

The reading mdoe is entered whenever the read button is pressed and follows the same logic as the writing mode to enter the "cycle of states." It is relevant to note that the counter doesn't start at the next "value" of the count (S_3 is 011, while S_4 is 110) because of the logic that allows the states to go back to the reset or idle state.

The state diagram and key corresponding each of the states to the inputs and outputs are shown in Figure 2 and Table 1 respectively.

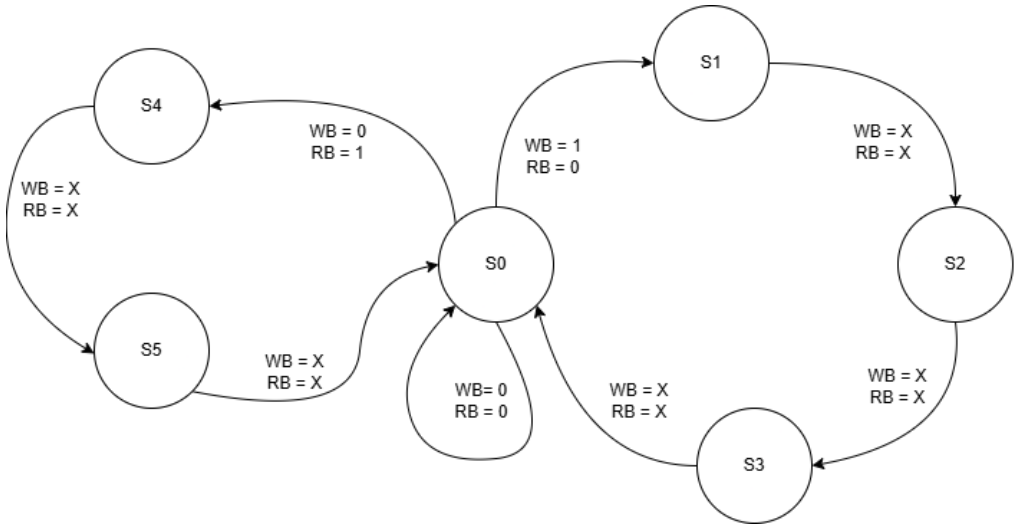


Figure 2: State Transition Diagram of this Implementation

State	$\overline{CS}$	$\overline{WE}$	$\overline{OE}$	S (Switch)	CLK 0	CLK 1	A (Address)	Purpose (Read/Write/Idle)
S0	1	1	0	X	0	0	0	Idle
S1	1	0	0	X	0	0	0	Write
S2	0	0	0	0	0	0	0	Write
S3	0	0	0	1	0	0	1	Write
S4	1	1	1	X	0	0	0	Read
S5	0	1	1	X	0	1	1	Read

Table 1: State Diagram Key

2.4 State Transition Table

State	Q_2	Q_1	Q_0	CS	WE	OE	S	CLK0	CLK1	A	Q_{2+1}	Q_{1+1}	Q_{0+1}
S0	0	0	0	1	1	0	X	0	0	0	0	0	1
S1	0	0	1	1	0	0	X	0	0	0	0	1	0
S2	0	1	0	0	0	0	0	0	0	0	0	1	1
S3	0	1	1	0	0	0	1	0	1		0	0	0
S4	1	1	0	0	1	1	X	1	0	0	1	1	1
S5	1	1	1	0	1	1	X	0	1	1	0	0	0

Table 2: State Transition Table

2.5 Boolean Algebra and Logic Tables

A second truth table (shown in Table 3) was created in addition to the state transition table (as shown in Table 2) in order to better keep track of the states and find relationships to create logic to move between states. This allowed the generation of expressions through inspection, and Karnaugh maps were not needed. This was done by creating a custom counter using three D-flip flops (Logisim implementation shown in Figure 3).

WB	RB	Q2	Q1	Q0	D2	D1	D0
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
X	X	0	0	1	0	1	0
X	X	0	1	0	0	1	1
X	0	1	1	0	0	0	0
0	1	0	0	0	1	1	0
X	X	1	1	0	1	1	1
X	X	1	1	1	0	0	0

Table 3: State Transition Truth Table

From this state transition/truth table, we derived the expressions:
$D_2 = (\overline{Q_2 Q_1 Q_0})RB + Q_2 Q_1 \overline{Q_0} = (\overline{Q_2 + Q_1 + Q_0})RB + (\overline{\overline{Q_2} + \overline{Q_1} + Q_0})$

$D_1 = (Q_1 \oplus Q_0) + (\overline{Q_2 + Q_1 + Q_0})RB$

$D_0 = Q_1 \overline{Q_0} + (\overline{Q_2 + Q_1 + Q_0})WB$

Note: The expression for D2 expands $Q_2 Q_1 \overline{Q_0}$ to $(\overline{\overline{Q_2} + \overline{Q_1} + Q_0})$ because it allows for the reuse of the NOR chip when building the physical circuit.

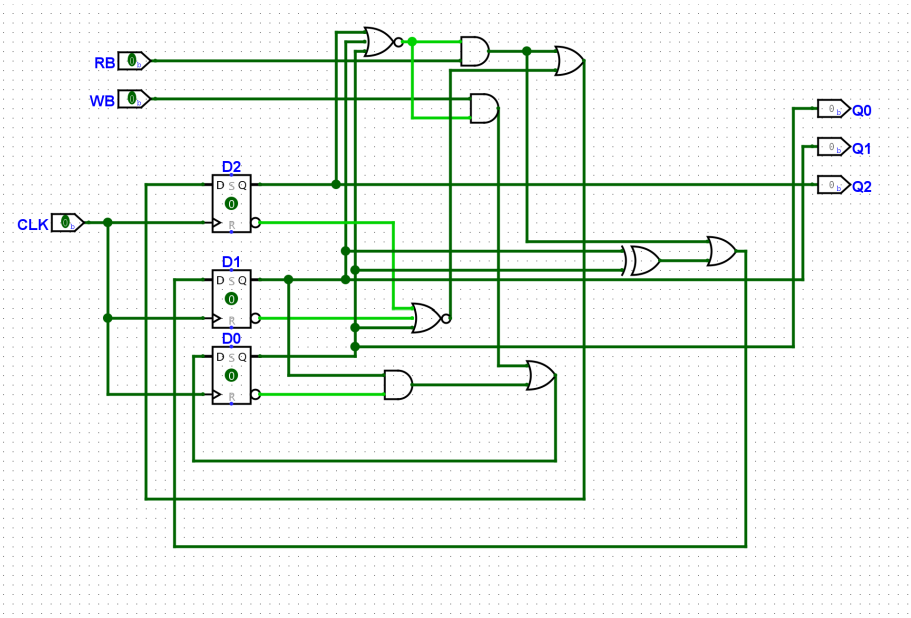


Figure 3: Logisim Diagram for custom counter

After creating the counter, we used the state transition table to create logic to control the inputs to the RAM chips and logical shift registers through inspection:

RAM:
$A = Q_1 Q_0$
$\overline{WE} = \overline{(Q_0 + Q_1)} \oplus Q_2$
$\overline{CS} = \overline{Q_0 + Q_1}$ $S = Q_1 Q_0$

MUX:
$S = Q_1 Q_0$
$\overline{OE} = \overline{(Q_0 + Q_1)} \oplus Q_2$

Registers:
$S_0 = S_1 = Q_2$ (for both registers)
$CLK_1 = Q_0 \oplus Q_1$
$CLK_2 = \overline{Q_0}$

This logic was then tied together with the custom counter logic, as shown in Figure 4.

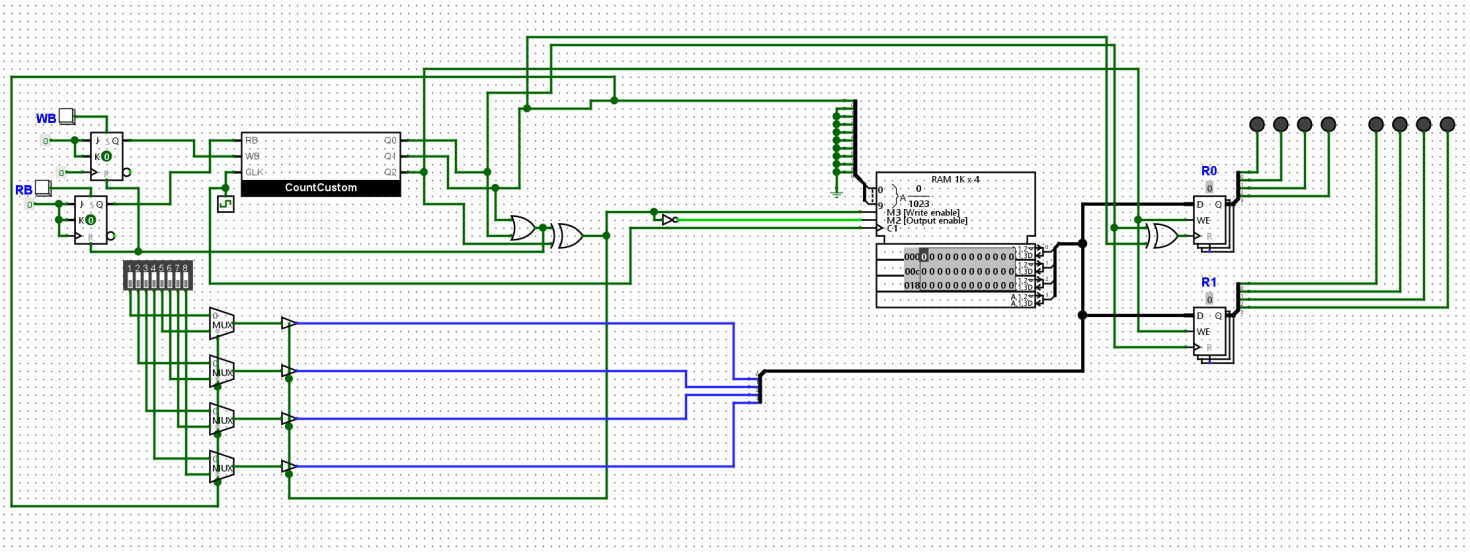


Figure 4: Logisim Diagram for all the logic

Note: The logic for the reset pin of the JK flip flop was pulled from $Q_1 + Q_2$, which prevents the button inputs from impacting the circuit when it is in the read or write mode (as the reset pin asynchronously sets the JK Q to 0).

2.6 Functional Block Diagram

This functional block diagram in Figure 5 shows how the finite-state machine interacts with different parts of the circuit like the RAM, MUX, buffer, and registers.

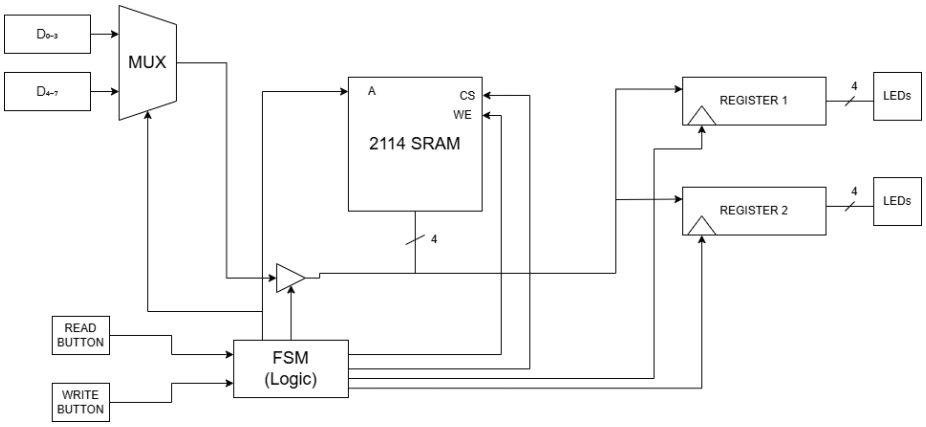


Figure 5: Functional Block Diagram

3 Implementation

This section will cover the physical implementation of the circuit and an explanation of the color coding scheme.

3.1 Circuit Design

To plan out the circuit and organize the debugging process, an Excel spreadsheet was used to keep track of everything (as shown in Figure 6 and 7, broken up due to the size of it).

Afterwards, the Excel was mapped out to the breadboard and expanded on when wiring to include the speaker and LED indicators to show when reading and writing are complete, allowing the user to know when each process was completed (Shown in Figure 8). Additionally, exclusively TTL chips were used in this implementation to prevent the need for TTL-CMOS interfacing.

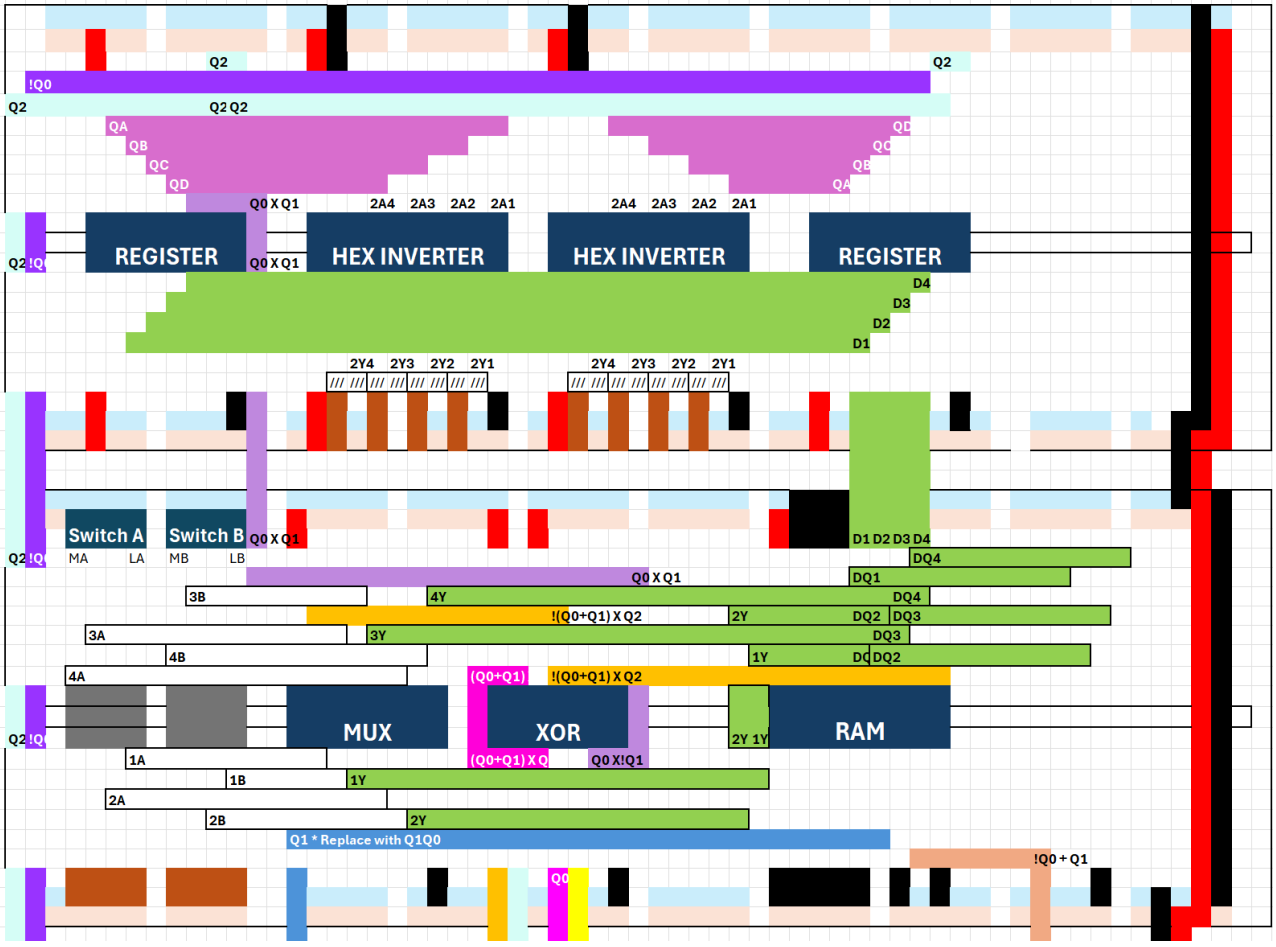


Figure 6: First two breadboards of implementation on Excel

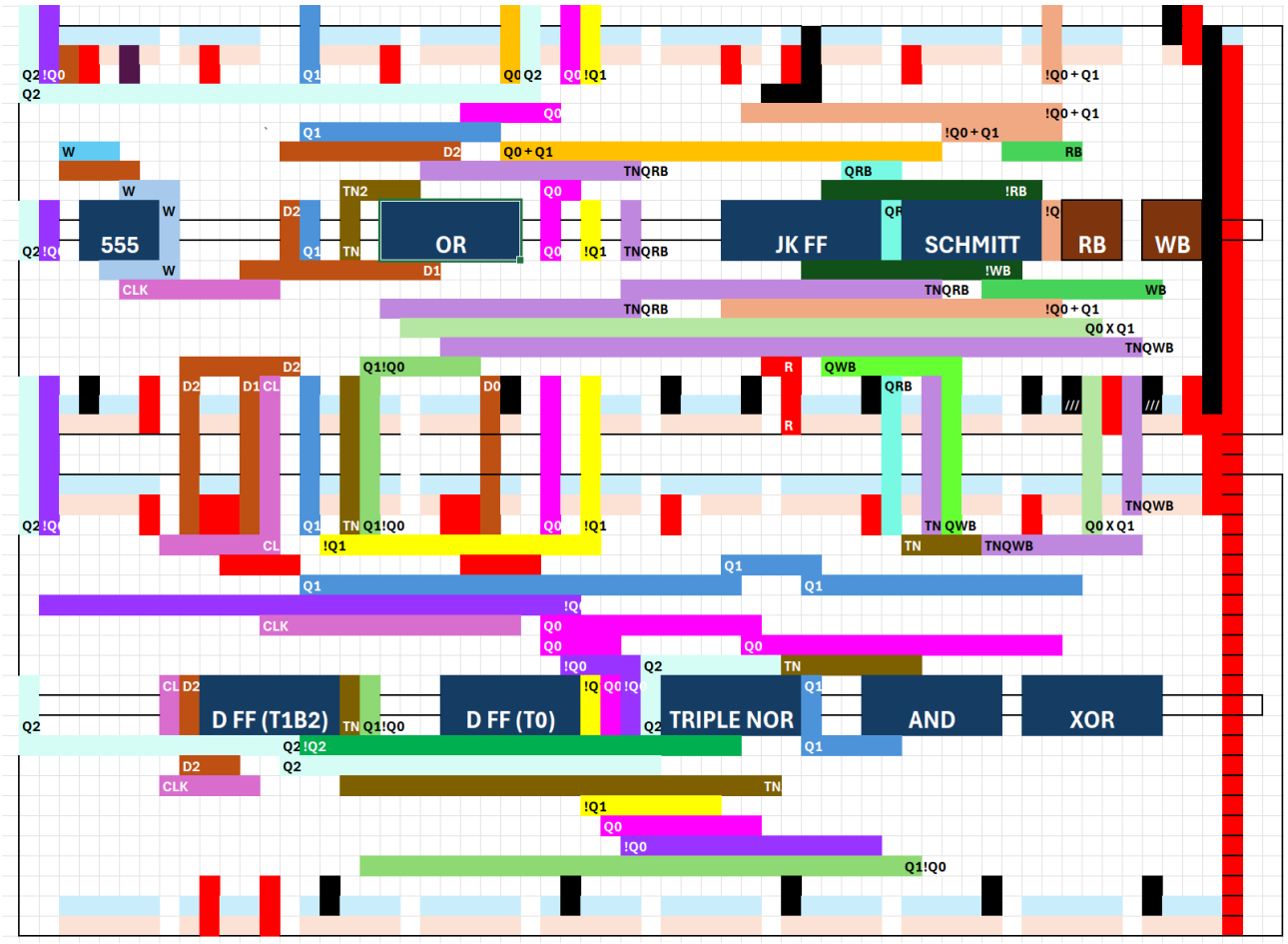


Figure 7: Second two breadboards of implementation on Excel

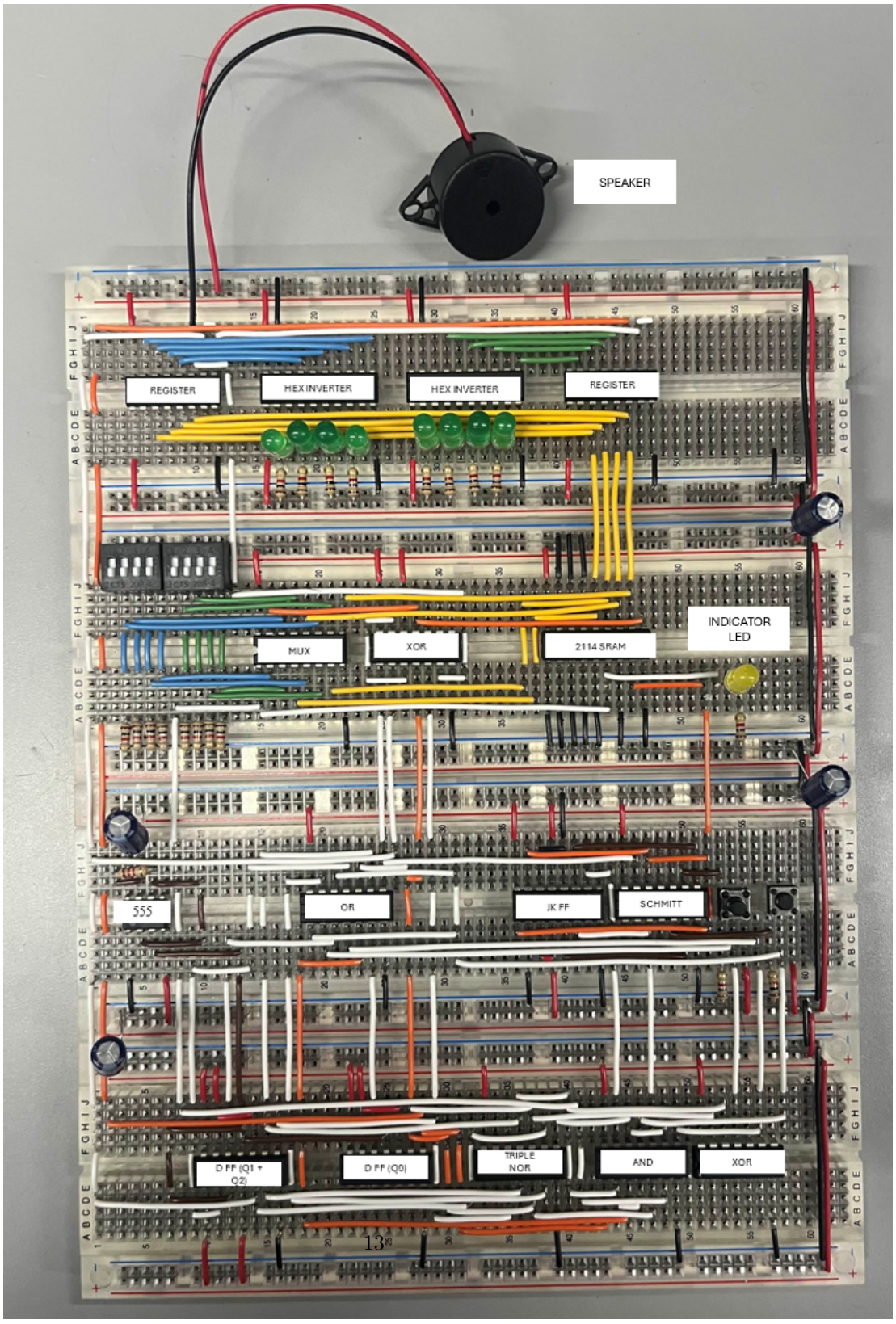


Figure 8: Implemented Circuit on Breadboard

4 Conclusion

4.1 Summary

This project uses a finite-state machine to write a 8-bit pattern from a DIP switch into a 2114 SRAM, and the most recently read pattern will be constantly outputted on LEDs using logic shift registers. In addition, the read and write buttons are debounced using a capacitor and Schmitt trigger and fed into JK flip flops to ensure that read and write cycles are not interrupted by inputs once they are started. This is done by feeding logic into the reset pin of the JK flip flop, which keeps it outputting a 0 or low signal.

To ensure that the circuit was operational, the following operation was done:

1. Set a desired 8-bit pattern on the DIP switch.
2. Press Write Button..
3. Wait until the speaker makes a sound.
4. Change pattern on the DIP switch if desired.
5. Press Read Button
6. Wait until the speaker makes a sound.

4.2 Limitation and Additional Considerations

One limitation of the circuit is that it doesn't use the full capabilities of the SRAM chip and only uses one of the address pins. If properly leveraged with more logic and additional parallel shift registers, more data could be stored and continuously output on LEDs using parallel shift registers. Another limitation would be expandability, as there is no way to record and display more data without completely changing the logic. Finally, the RAM chip had a tendency to heat up, which suggests that long term usage could be dangerous and not ideal.

Although it is not expandable, the current implementation can be integrated with larger systems to serve a variety of purposes. In modern computing systems, this implementation can be used in parallel with a CPU to hold cache data while it does other operations, calculations, and tasks. This temporary storage could also be used as a buffer for various sensors connected to a microntroller through a single BUS to prevent clashing signals. In short, anything that requires data to be temporarily held could benefit from this SRAM implementation.