DSF: Final Project

By Zeyn Schweyk, Everett Button, Shantanu Misra

Introduction

The Finite State Machine (FSM) we designed acts as a 4 digit digital keypad. If the number 8924 is entered, a green LED turns on, and for any other combination a blue LED stays on.

Methods

Using a Moore machine, we came up with a FSM that initially had 5 states: A, B, C, D, and E. We decided to use one-hot encoding to encode the states as follows and simplify some of our combinational logic: A (00001), B (00010), C (00100), D(01000), E(10000).

However, we had to make a lot of design choices to make sure the project could be smoothly implemented in hardware.

After Shantanu and Zeyn built most of the circuit, we started running into multiple issues that were unclear how to solve. When analyzing it further, Zeyn was able to reduce possible sources of error by simplifying the logic such that only 4 states were used, as well as fewer combinational logic gates, emulating something more representative of a Mealy machine.

Components Used:

- One SN74HC175N - D Flip Flop Chip (4/4 D Flip Flops)
- One SN74HC08N - AND Chip (4/4 AND Gates)
- One SN74HC04N - NOT Chip (1/4 NOT Gates)
- One SN74HC32N - OR Chip (4/4 OR Gates)
- One 555 Timer
- 11 Push Buttons (10 for digit input, and 1 for RESET)
- 5 LEDs (2 for incorrect/correct passcode, 3 to see intermediary states)
- 12 Resistors (4 for push buttons 8, 9, 2, 4; 1 for RESET push button; 2 for incorrect/correct LEDs; 3 for intermediary states; 2 for 555 timer)
- 1 Capacitor (for 555 timer)

The FSM starts out in no state, and returns to this “no state” state if the code entered is incorrect or if the RESET push button is pressed. If 8 is pressed, it advances to State A, otherwise it remains in no state. Once in State A, if 9 is pressed, it advances to state B, otherwise it goes back to being in no state. Once in State B, if 2 is pressed, it advances to State C, otherwise it goes back to being in no state. Once in State C, if 4 is pressed, it advances to State D (turning the green LED on), otherwise it goes back to being in no state. Once in State D, any push button that is pressed will result in the FSM returning back to its “no state” state.

Interestingly, our working clock frequency in Logisim was not behaving flawlessly in the circuit. In our simulation, we found that the best frequency to have the clock at was around 4-6 Hz. Using the 555 timer, 2 resistors, and 1 capacitor, Zeyn and Shantanu found some relevant formulas that would yield necessary resistance and capacitance values that would create a clock signal of around 6 Hz. Anything higher than that would make the flip flops traverse multiple states with the user only pressing a push button once, since it would detect the button being pressed for multiple rising edges on the clock. For these reasons, besides the 2 LEDs that indicated whether or not the passcode was correct, Zeyn added 3 more LEDs to represent the current state, which significantly helped streamline the debugging process. This way, if/when a user presses a digit, they will immediately be able to determine whether or not that input was picked up by the FSM. Of course, password validation logic in the real-world does not reveal whether or not a given digit is correct, but in this context, it helped him troubleshoot issues that would not have been as easy to track otherwise. So, all in all, reading the LEDs from left to right in the video attached below:

LED 1 (blue): represents the “no state” state - passcode is incorrect
LED 2 (white): represents State A - passcode is still incorrect, but user entered the correct first digit
LED 3 (white): represents State B - passcode is still incorrect, but user entered the correct first and second digits
LED 4 (white): represents State C - passcode is still incorrect, but user entered the correct first, second, and third digits
LED 5 (green) : represents State D - passcode is correct

Because the user can choose to manually reset the FSM with the RESET push button, each D-Flip-Flop requires a CLR signal that is directly tied to that push button. However, as the datasheet for the D-Flip-Flop chip shows, the RESET push button signal had to be inverted before being fed into the D-Flip-Flop chip, which explains the purpose of our one NOT gate in our circuit.

To ensure that the clock signal would only reach the flip flops when the user pushed a button, Shantanu had the idea of enabling all the flip flops only when the clock was high and there was an input. We did this by passing the clock signal and the button inputs

ORed together through an AND gate. Due to the fact that we would need another OR chip if we wanted to OR all input signals, we decided to only OR the 4 push buttons of our passcode. This ensured that the clock would only reach the flip flops when the user pushed a button.

Another design choice that we made involved how many push buttons we wired. For simplicity, we decided to include all 10 digit inputs in our circuit for aesthetics, although we only wired up the 4 push buttons that mattered for our code. If we hadn't done so, we would have needed to add 2 more OR chips, as explained above, as well as 6 more resistors.

In general, when working on the circuit, we decided to color code our logic as follows: Yellow (input signal), White (Q), Blue and Green (some modification on an input signal or flip flop output), Red (power), Black (ground).

Self-Assessment

The first stage of the project involved all three of us meeting and designing the FSM on paper. We decided the states, encoding and the circuit together. After that Shantanu made the simulation using Logisim.

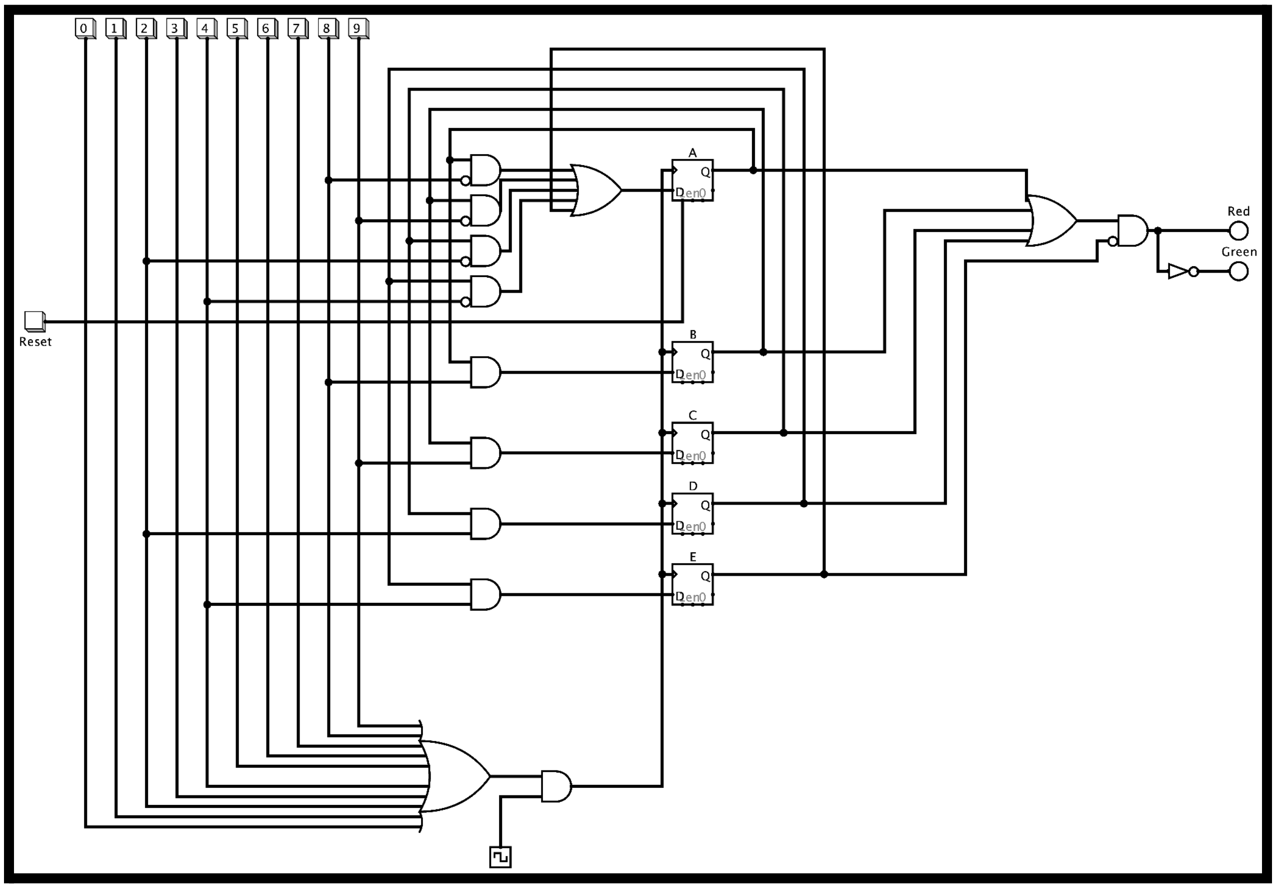
Using the simulation as a template, Shantanu and Zeyn collected the parts required to assemble the circuit and started making it. After finishing most of the circuit though, we ran into a few issues regarding the push buttons and the reset/preset on the flip flops.

Zeyn alone worked to fix these issues, and Everett helped retrieve necessary parts. In addition to connecting all the push buttons correctly, he was able to get rid of many of the initial circuit problems we faced by simplifying the combinational and sequential logic even further. He realized that another D-Flip-Flop chip (74LS74) would be required to get our circuit working such that it matched our initial simulation but, in the process, came up with another schematic that would eliminate the need of that new chip and only require a total of 4 D-Flip-Flops on the chip Professor Buccafusca provided the class with, as well as only 1 AND, NOT, and OR chips.

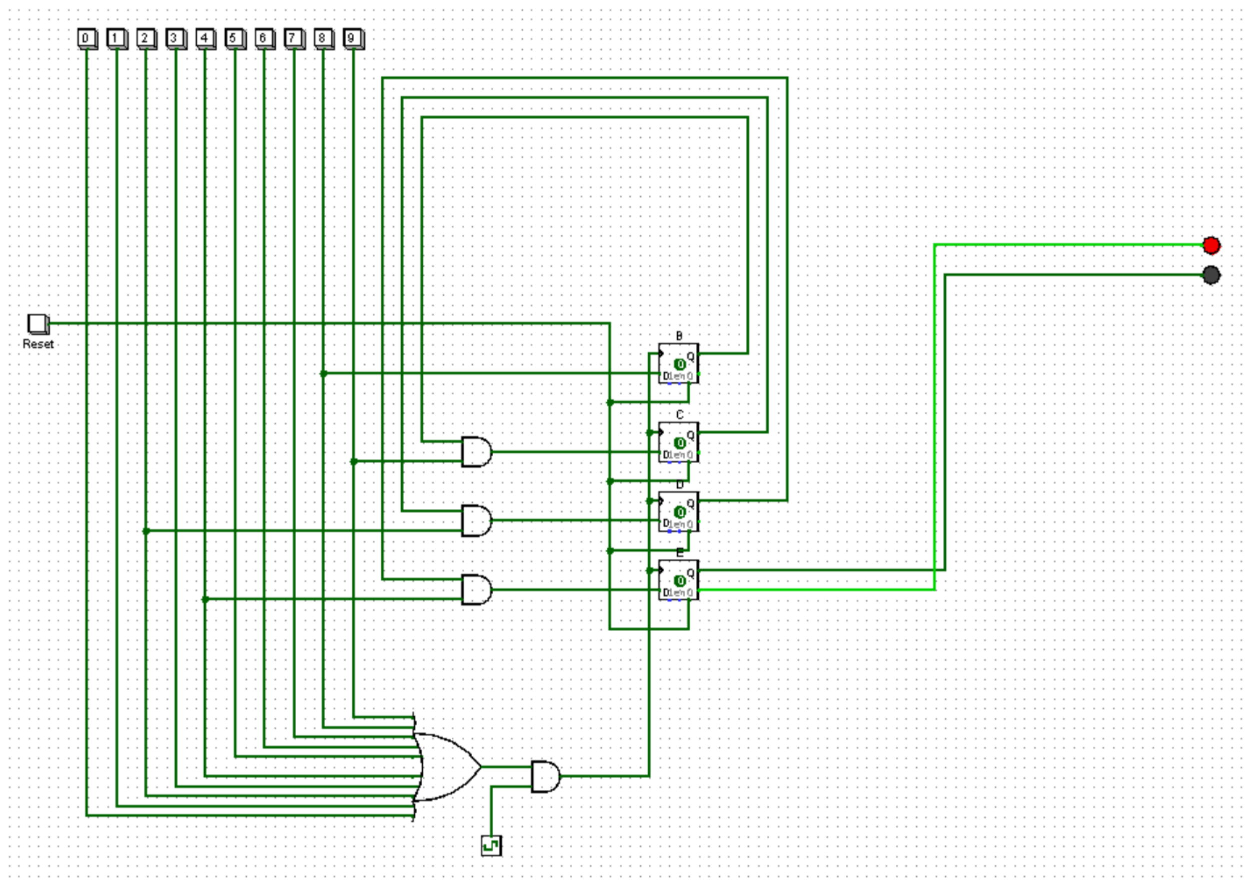
Everett laser cut an acrylic blue board that says "Digital Keypad" on it to make the circuit more appealing.

Hardware and Software Pictures

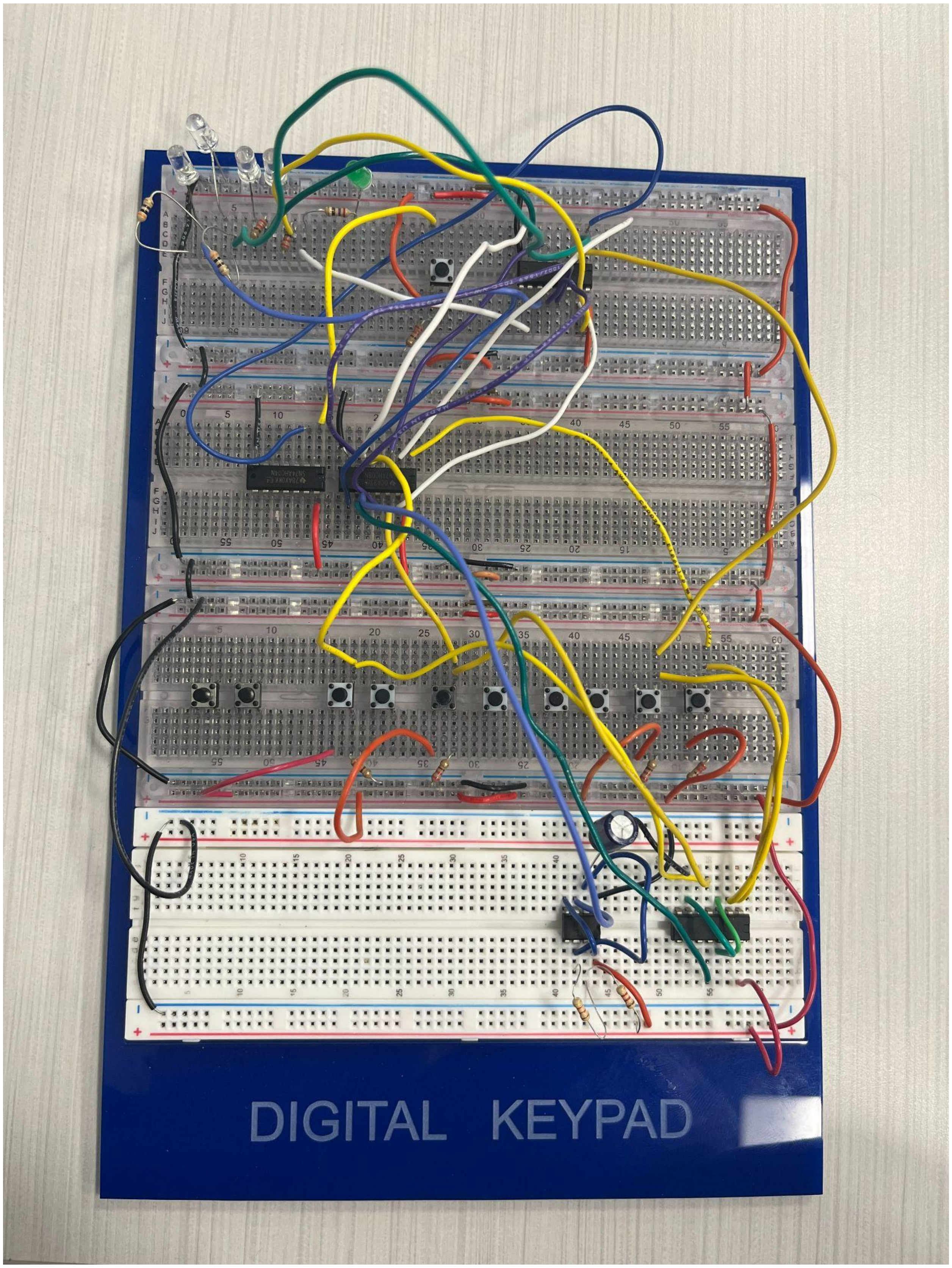
Initial schematic that Shantanu made in Logisim based on the Moore Machine we came up with:

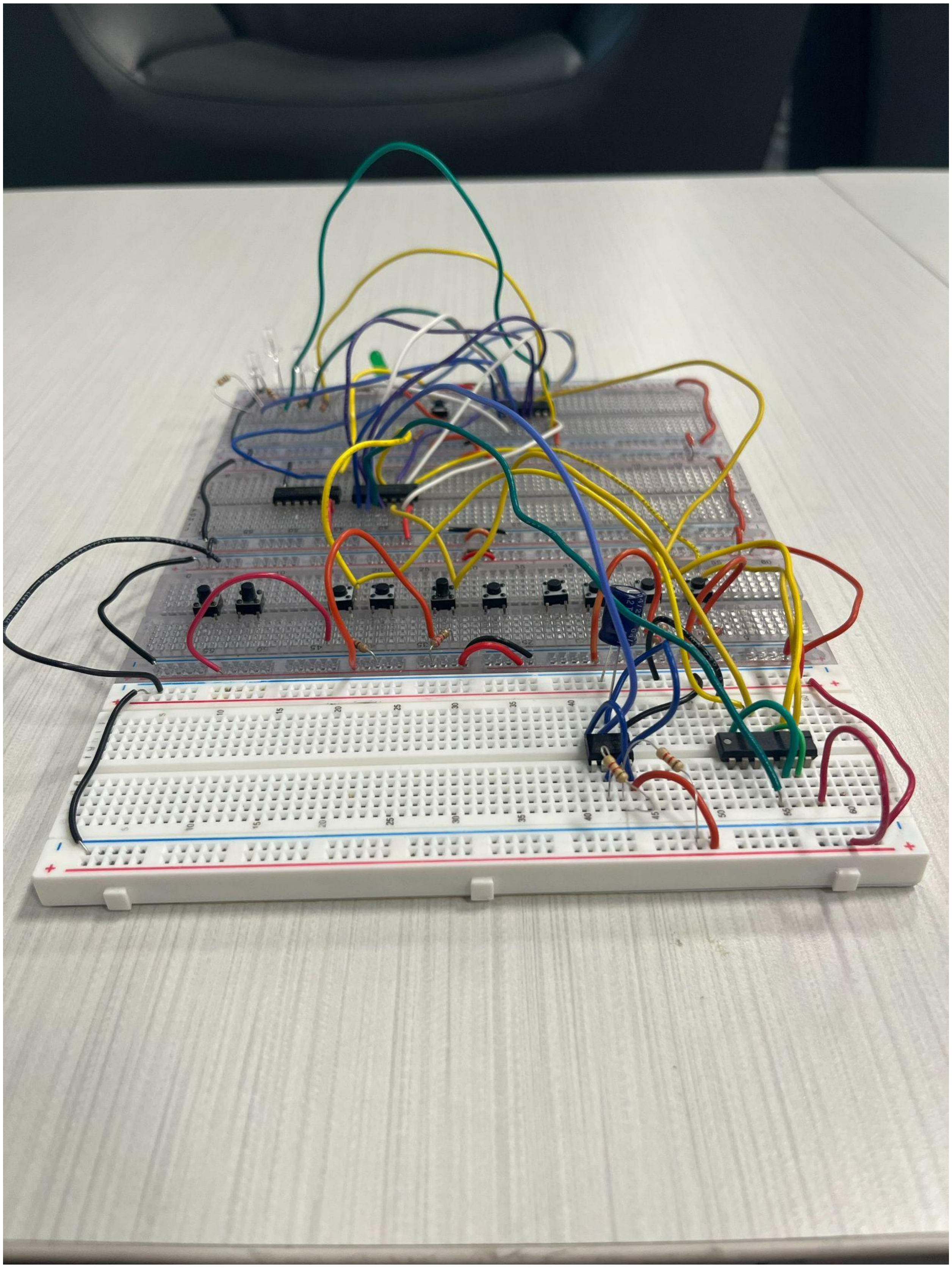


Updated/simplified schematic, representing the actual logic of the circuit, that Zeyn came up with:



Circuit Images:


DIGITAL KEYPAD



Videos of working circuit, with various different inputs, including RESET:

https://drive.google.com/drive/folders/1zz9KNBRI5rPFpJdTMtkiZMIgtpwSV5aN?usp=sharing