Philip Nevins
4/8/2022
ENGR 271 Digital Logic II

# Lab 0x08: Digital Lock

## Introduction

In this lab, we are building a digital lock, using the given digital lock schematic. At first, it seems very complex, however we will be breaking it down into individual components to provide a more in depth analysis of how it works. In our given schematic, we are told that it has two 3-bit inputs that must be entered in order. You must put in the first 3-bit code then manually toggle the clock. Next, you must enter the second 3-bit code and manually toggle the clock again. If both codes are entered correctly, in order, a LED will turn on and signal the lock is unlocked. Then we will expand the design to require four 4-bit inputs.

The objectives of this lab are to learn additional features on Multisim such as: setting properties and angling wires, using buses and creating subcircuits. We are also exploring the advantage of using sequential logic to reuse hardware and to see how flip-flops can be used to design complex circuits with real world applications.

## Part A: Asynchronously Triggered Clock Signal

In this section, we are building the clock signal for our digital lock. We are using the VCC (5V) component for our power source, that has two 1k ohm resistors in parallel. This leads to the input of two 74LS00N ICs, that we connect as shown in Figure 1. The switch we are using is the SPDT, which can be found in Group -> Basic. The switch is connected between the two resistors and the inputs of the two 74LS00N ICs. We add the output connector by selecting Place -> Connectors -> On-Page Connector (or Ctrl+Alt+O), which is connected to the output of the two 74LS00N ICs. We name this output as CLK.

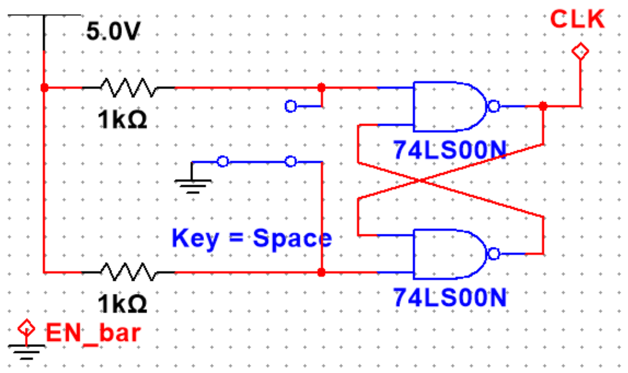

Figure 1: Asynchronous Triggered Clock Signal

We then perform some basic cleanup. We turn off RefDes in the Sheet Visibility tab and change the key that actuates the switch to the Space bar. We also angle the wire between the two 74LS00N ICs to make it more visually appealing.

## Part B: Bus

In part B, we design the Bus with the three 3-bit inputs. After building the first input (Key = A), I utilized a copy and paste function to build the other two inputs. The first input (Key = A) can be seen in Figure 2. In Figure 3 (next page), is the schematic that was created from the lab procedures. Figure 3 shows where I connected the 3-bit input into the Bus line.

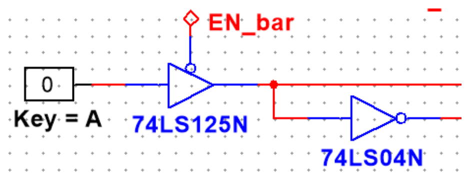

Figure 2: 1-bit Input Key=A

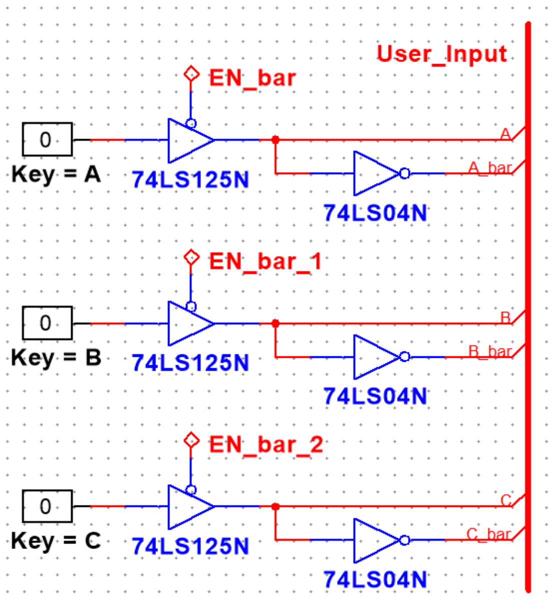

Figure 3: Bus with 3-bit input

## Part C: Circuits as Subcircuits

This is where we made the subcircuits for Combo 1 and Combo 2, displayed in Figure 4.

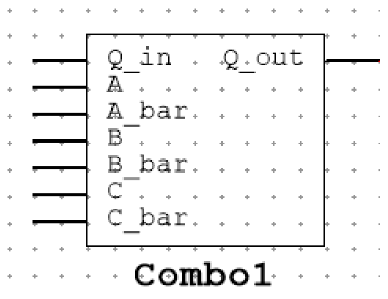

Figure 4: Subcircuit Combo 1

Inside the subcircuit, we made the two 3-bit inputs that need to be entered, in order, to open the lock. Combo 1 and Combo 2 subcircuit internals can be seen below. The inputs inside the subcircuit that are not connected, are not being used. For example, in Combo 1 we have A, B_bar and C connected, which correlates to '101' being the first 3-bit binary input. Then for Combo 2, we have A_bar, B, C_bar connected, which correlates to '010' being the second 3-bit binary input. In figure 5, you can see both Combo 1 and Combo 2 subcircuits with the correct 3-bit binary inputs.

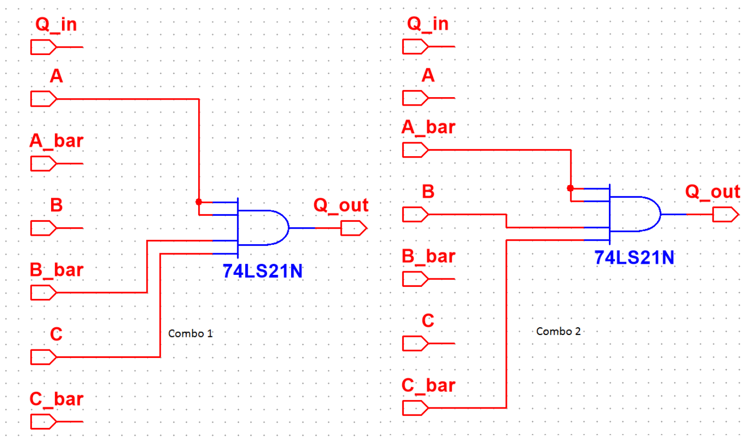

Figure 5: Combo 1 (101) and Combo 2 (010) subcircuits

## Part D: D Flip-Flops

In Part D, we add the two D Flip-Flops (DFFs), the output LED with a 5V VCC source and a 220 ohm resistor. I set the first DFF up to ripple to the steering logic of the next DFF. This way, the input bigs are reused for the next number in the sequence used to unlock the lock. I then tested then circuit. Figure 6 shows the fully completed circuit, Figure 7a shows the first sequence being input and Figure 7b shows the second sequence being input with the resulting LED turned on, signifying the lock has been opened. It works!

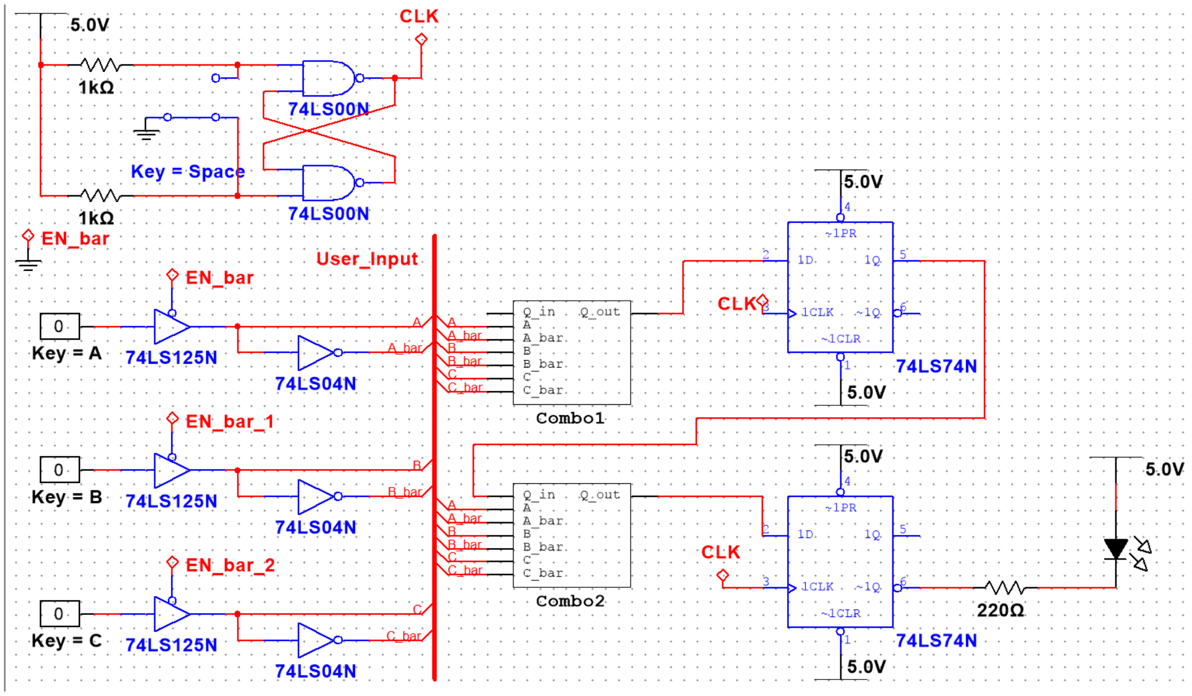

Figure 6: Completed 3-bit input Digital Lock

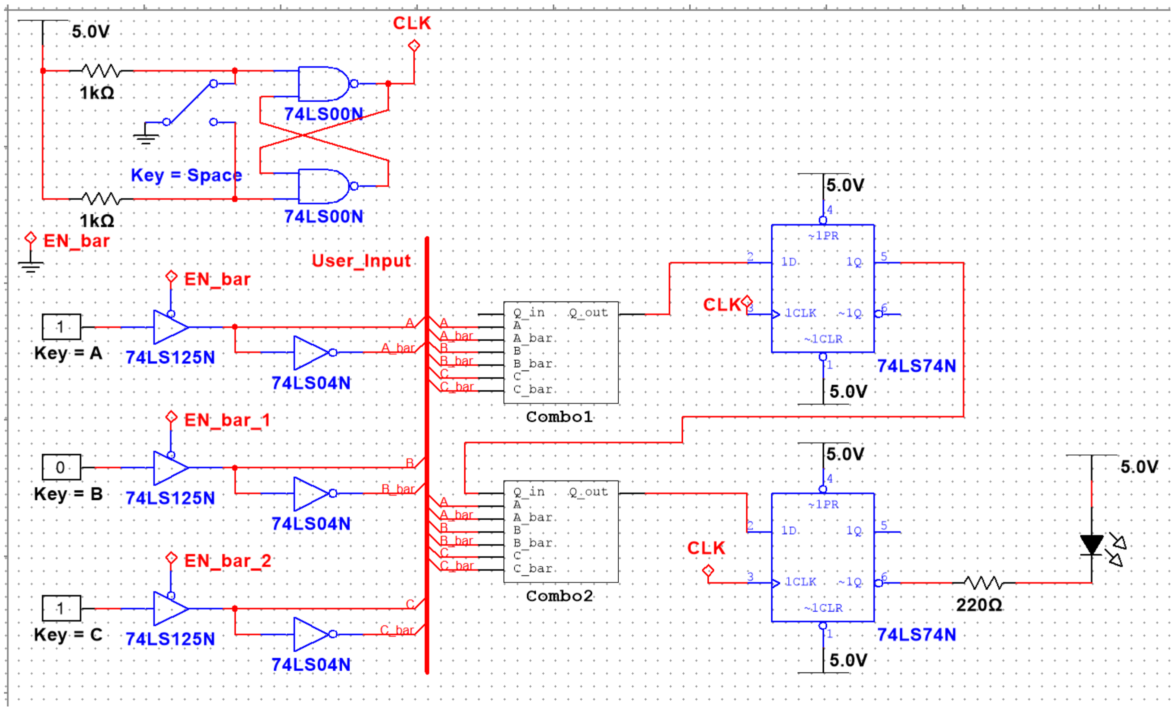

Figure 7a: Combo1 (101) input and clock pushed

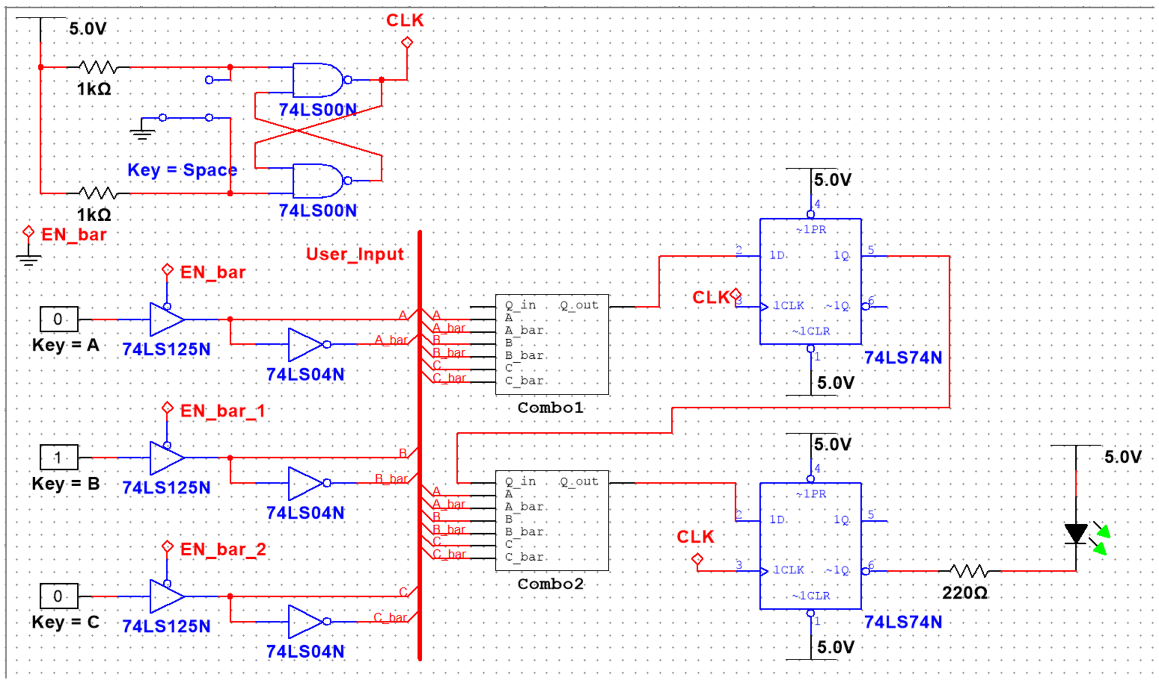

Figure 7b: Combo2 (010) input, clock pushed and LED on / digital lock open

## Part E: Expanded Design

In the expanded design, I was asked to expand the digital lock to four inputs (A, B, C, D) and two additional DFFs, making the circuit composed of four DFFs that are rippled. I was asked to set the combination to 0x3A3B. I decided to use B, LSD as the first input and 3, the MSB as the last input, making the input order B, 3, A, 3; or 1011, 0011, 1010, 0011.

To complete this circuit, I again utilized the copy and paste function to make a 4th input and then built the Combo4 subcircuit internals from scratch and added the 4th bit to each of the other Combos. I also built the DFFs from scratch. Utilizing the copy and paste function within the combos and DFFs can cause the circuit to work improperly. In figure 8a and b, you can see the combo1-A, 2-3, 3-A, 4-3 subcircuits that were built. For ease of viewing, I combined Combo1+2 and Combo 3+4 into two single figures, 8a and 8b respectively. I also added in a normal-open pushbotton (PB_NO) to reset all Q inputs, shown in Figure 9 in the top right corner. Then everything was connected to the bus in the same way as Part B. The completed circuit can be seen in Figure 9. The circuit worked flawlessly the first time, and then when I tried to run through a second time, the lock was opening every time 0011 was input. I attached current probes (which you can see in the Conclusion, at Figure 11, under #5) and was able to determine that the push button had to be held down for 5 seconds to completely clear out all of the previous inputs. I assume this is because I am using a slower computer (~8 year old Microsoft Surface Pro).

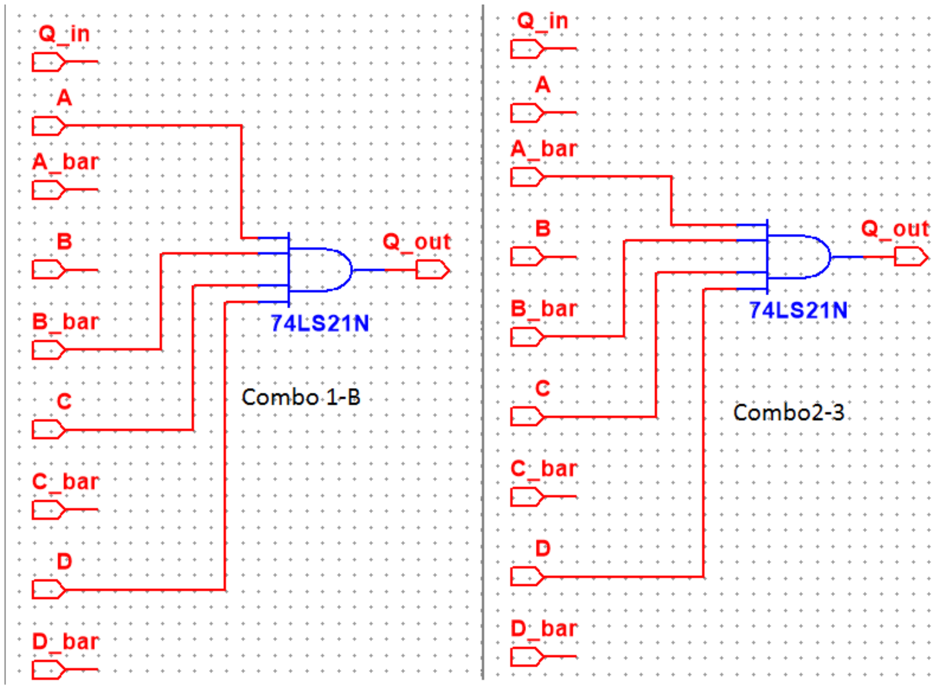

Figure 8a: Combo1-B (1011) + Combo 2-3 (0011) subcircuit

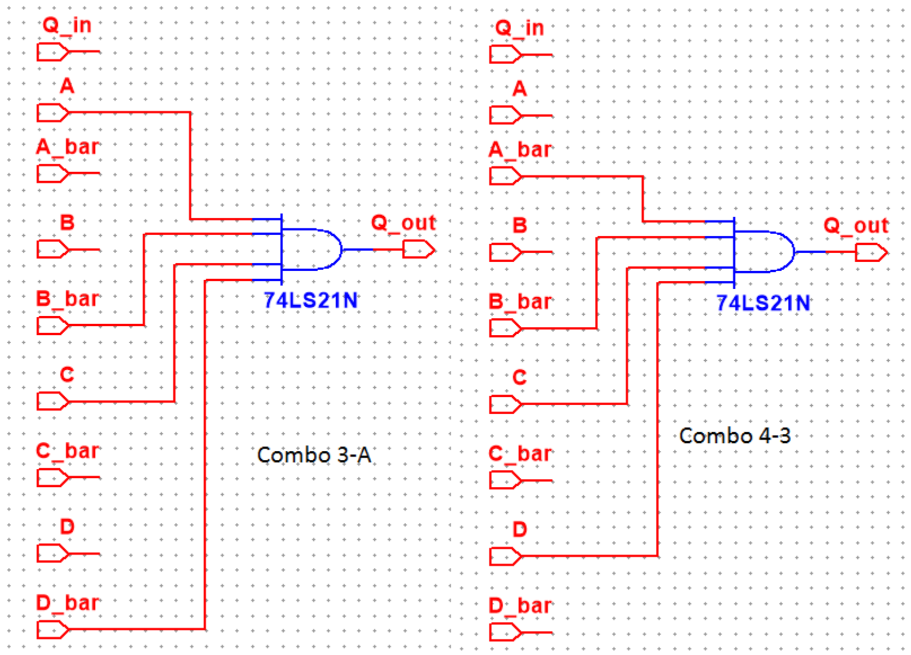

Figure 8b: Combo 3-A (1010) + Combo 4-3 (0011)

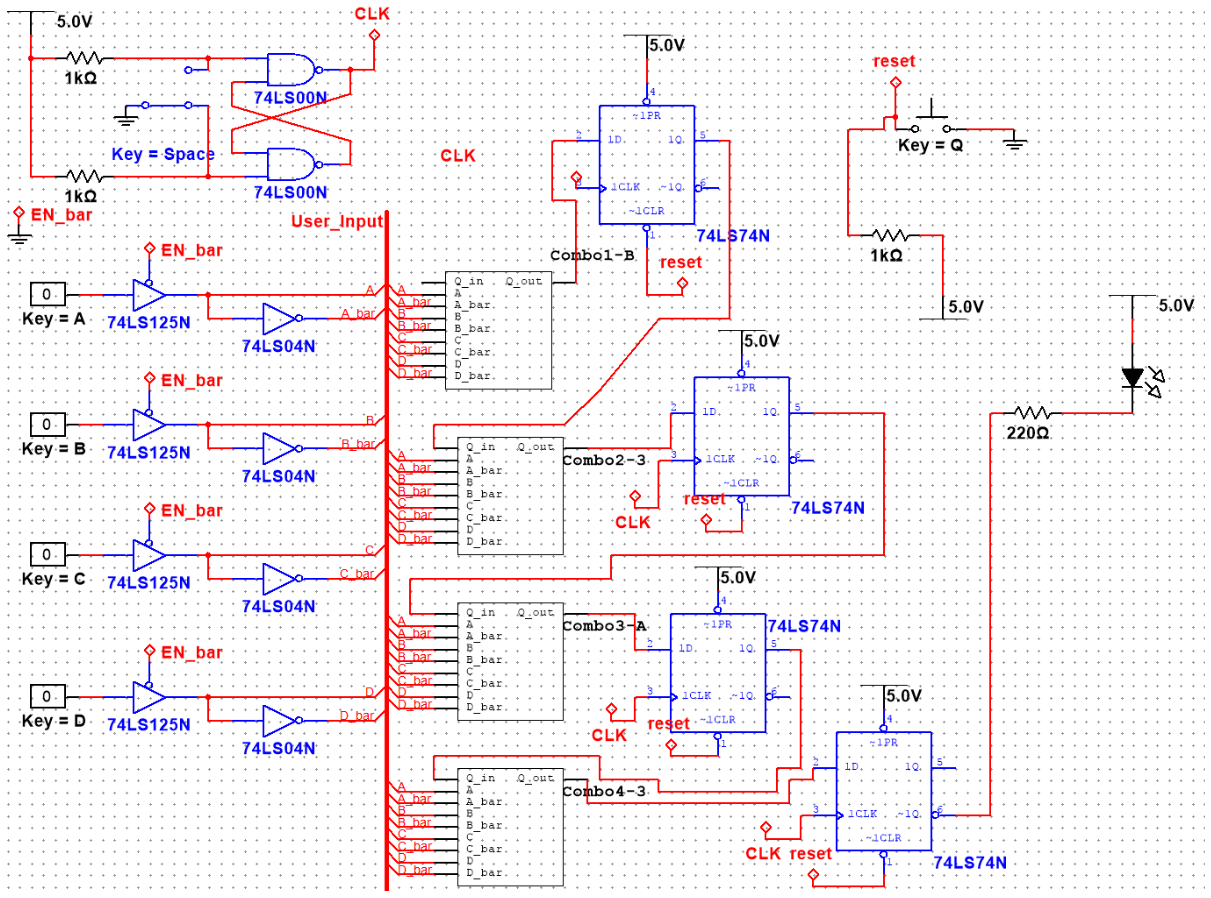

Figure 9: Completed 4-bit input Digital Lock with push button reset

In Figure 10a, b, c, d, you can see each 4-bit input being entered and the clock being pushed to move along to the next set. In Figure 10d, you can see the LED has turned on, signifying that the lock has opened. The circuit works properly! To test the circuit, I used B, LSD as the first input and 3, the MSB as the last input, making the input order B, 3, A, 3; or 1011, 0011, 1010, 0011.

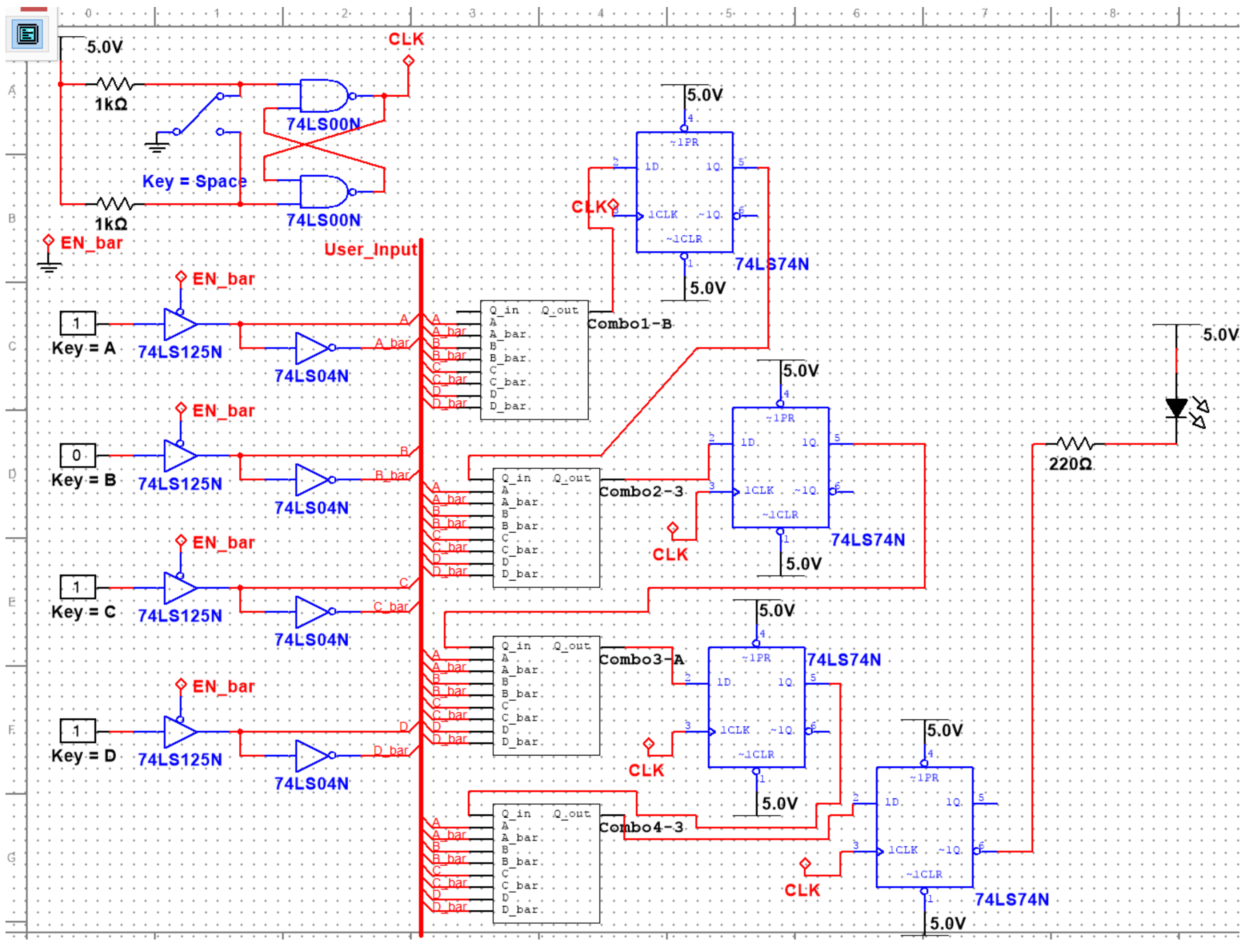

Figure 10a: Input ‘B’ (LSB), 1011

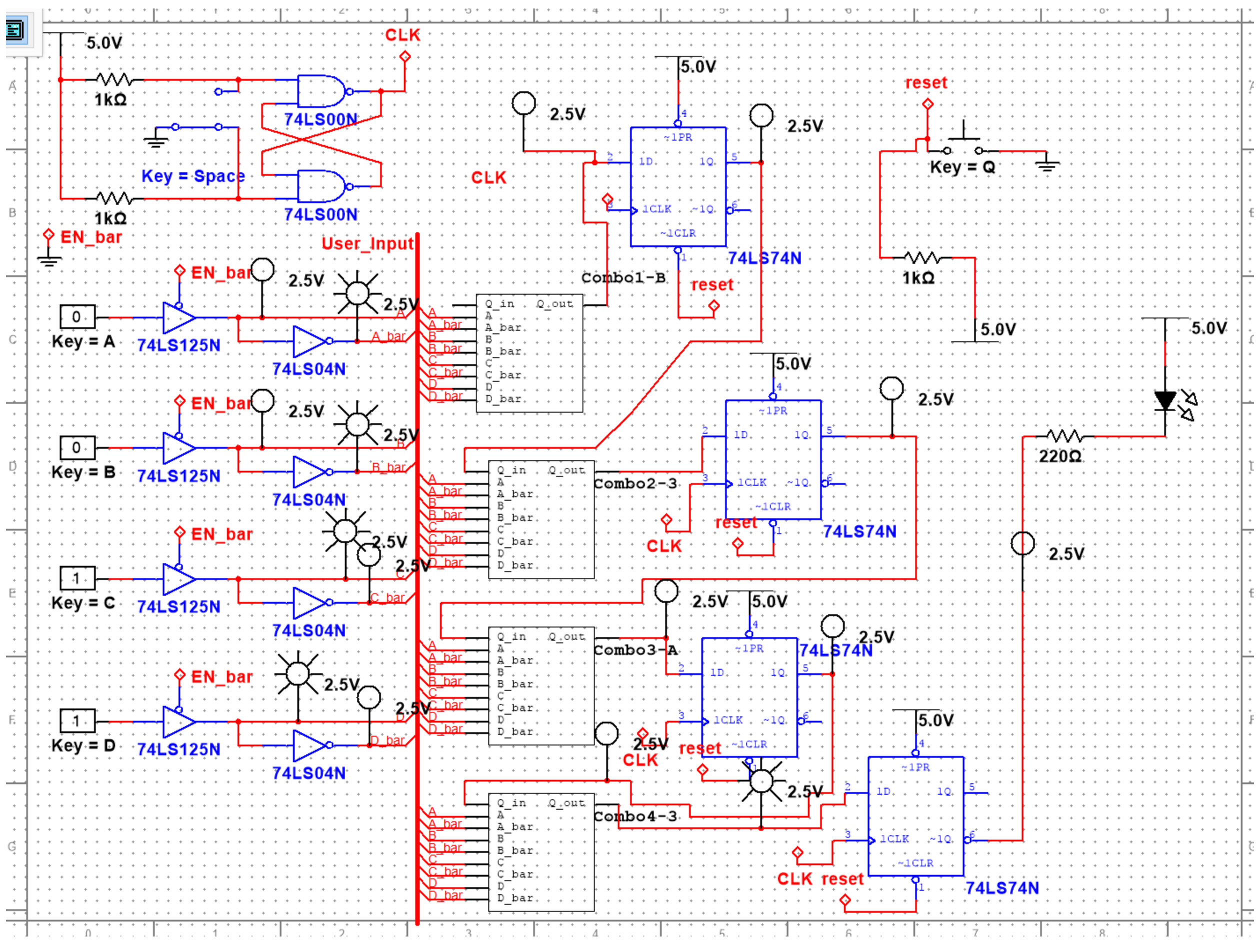

Figure 10b: Input ‘3’, 1011

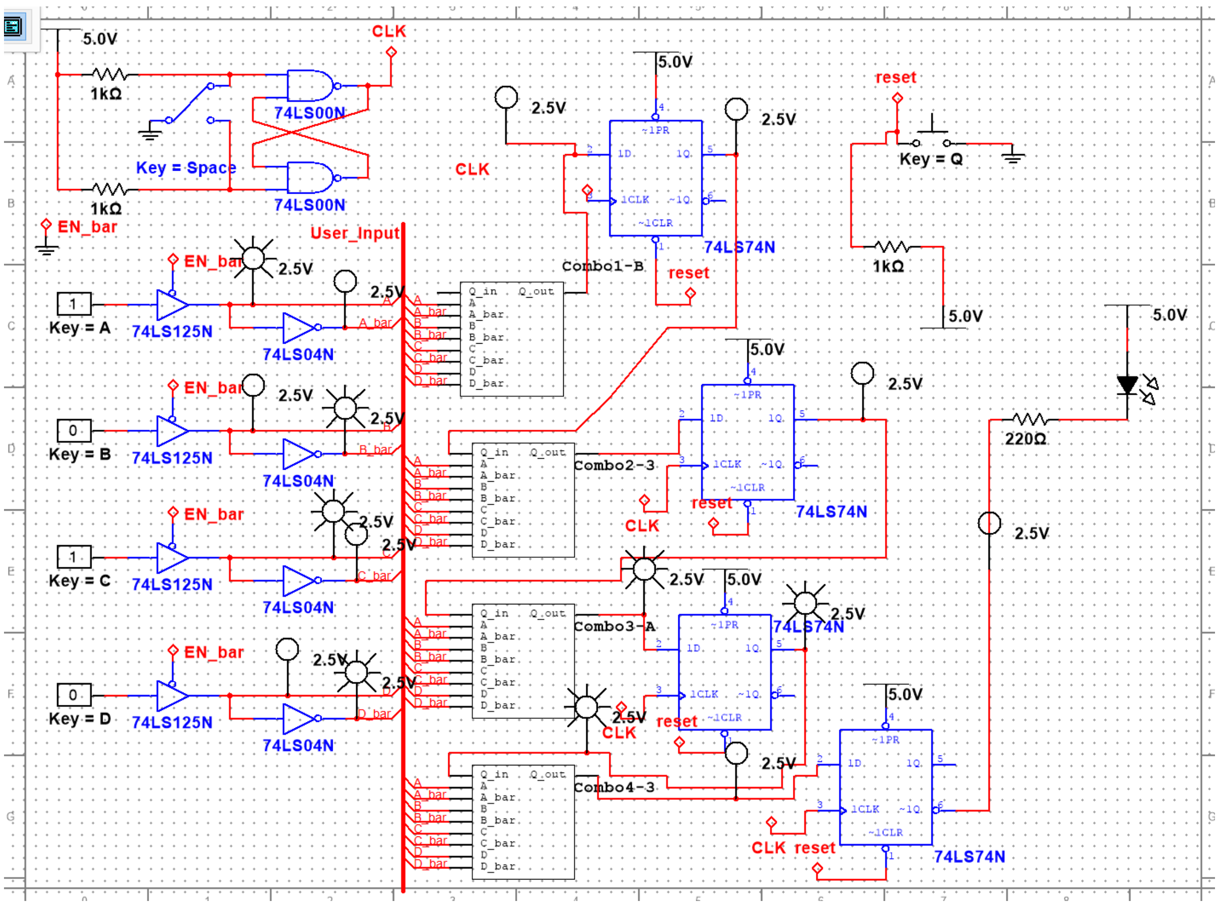

Figure 10c: Input ‘A’, 1011

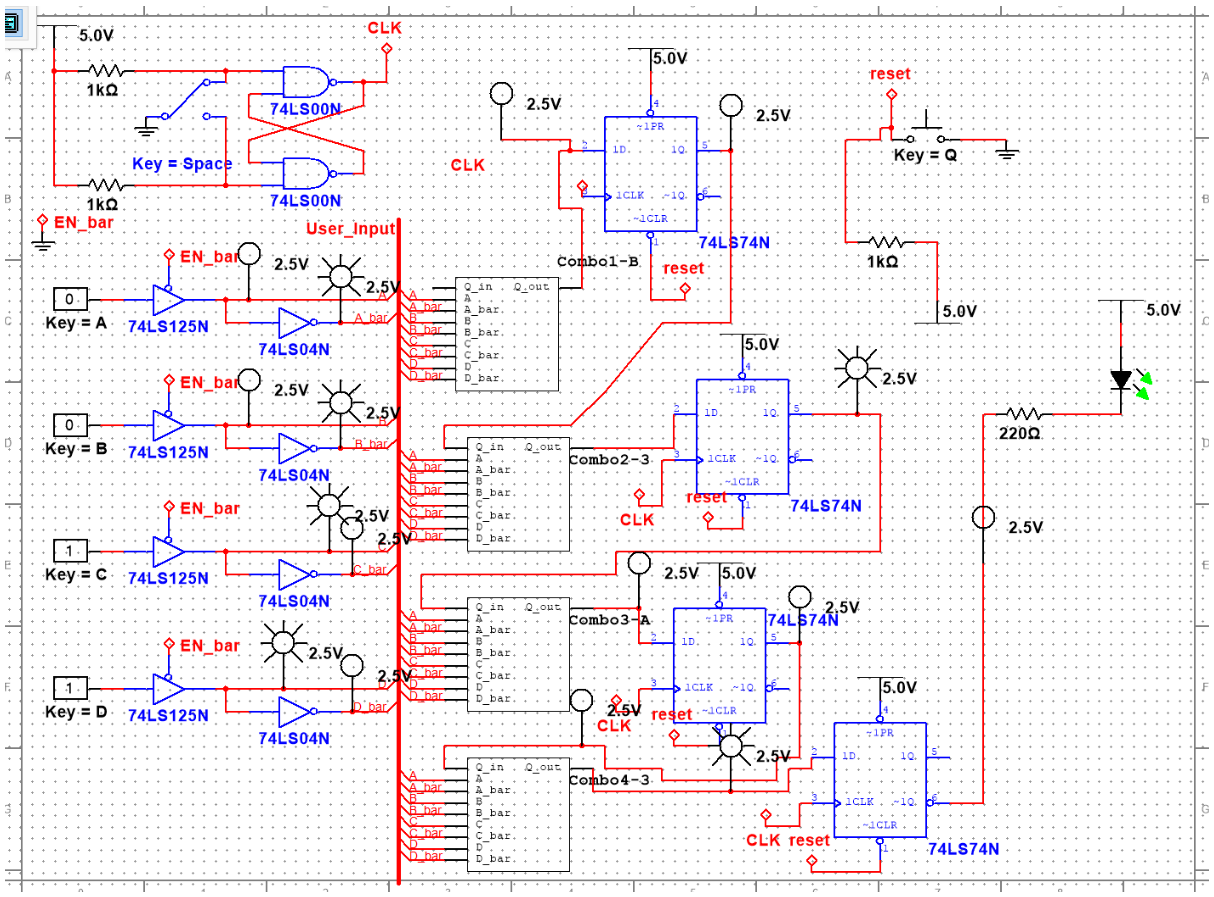

Figure 10d: Input ‘3’ (MSB), 0011

## Conclusion

It was very interesting seeing how a multi-input digital lock works. This explains how safes work, that require an input code to open them, and how door locks work. I surprisingly didn’t have many issues with setting up and testing the circuit. It worked flawlessly the first time, but as I stated in Part E, I did have issues running it after that. I was able to determine that the reset push button was not being held down long enough to completely clear everything in the DFFs. If I was going to continue with this design and make it better, I would add in 5 more DFFs, for a total of 9 DFFs and utilize a 9 digit number (971503541) to unlock it. The first input would be 9 (1001) and the final input would be 1 (0001). This would 100% guarantee the lock is un-lockable with entering random combinations, as it would take ~2179 years (**17.5 seconds * 16^9 possible combinations -> convert to years -> 2179 years**). Below, I have answered the question from the final Procedure section of the lab assignment and stated each question for ease of the reader. I have bolded the calculations done on a TI-89 Titanium Graphing Calculator

1. Create the entire circuit including the expanded design.

   This can be seen in Figure 8a, b (subcircuits for input code) and Figure 9 (complete 4-bit digital lock circuit)
   .
2. How many different lock combinations are possible by only changing the logic in the subcircuits?

   There are 4 bits in each combination, therefore we have **2^4 = 16** different bit combinations (0000 to 1111) that are possible. Since we can have duplicate combinations (0000,0000,0000,0000 or 1010,1010,1010,1010) we do not want to use the factorial method for finding how many combinations a state can be assigned unique state bits. We can simply do **16^4 = 65,536**. This is because we have 16 different combinations per subcircuit and 4 subcircuits.

3. If you don't know the lock combination, approximately how long does it take to try and open the lock with any one combination? Are some combinations faster than others?

   To calculate this, I timed myself entering the test code, which took 17.5 seconds. Using the calculation from #2, we know that there are 65,536 different possible combinations. So to find the approximate time to open the lock not knowing the combination, **we multiply 17.5 (seconds) * 65,536 possible combinations, then after doing time conversions to days, we have 13.27 days**. Some combinations would be faster, yes. An example would be if all the combinations are the same, IE 1010, 1010, 1010, 1010. In this instance, the CLK could be pulsed and you could move through all 4 inputs a lot faster, therefore I calculated this as the maximum time.

4. How long would it take you (on average and at most) to find an unknown combination by trying various combinations? Repeat the calculation as if there were 5 DFFs.

   To find the average time it would take to unlock the lock without knowing the combination, I would assume a good estimate would be the average of the quickest and the longest time to achieve unlocking the lock in this fashion. The quickest would be 17.5 seconds, if you unlocked the lock on the first attempt and the longest would be the answer from #3 above, which is 13.27 days. **The average of 17.5 seconds and 13.27 days is 6.64 days. [(1.14688e6 + 17.5) /2] {Seconds}**, then convert from seconds to days.

   If we have 5 DFFs, the number of possible combinations changes. We re-calculate it, as **16^5 = 1048576 possible combinations**. Since we are only adding one more DFF and not adding another input bit to each code, we can use this with our estimated time to enter a 4-bit input code from #3 (17.5 seconds). **We then have 17.5 (seconds) * 1048576 possible combinations, then after doing time conversions from seconds to days, we have 212.305 days**.

5. Measure the maximum current. Show how you set up the measurements and suggest methods to reduce power consumption.

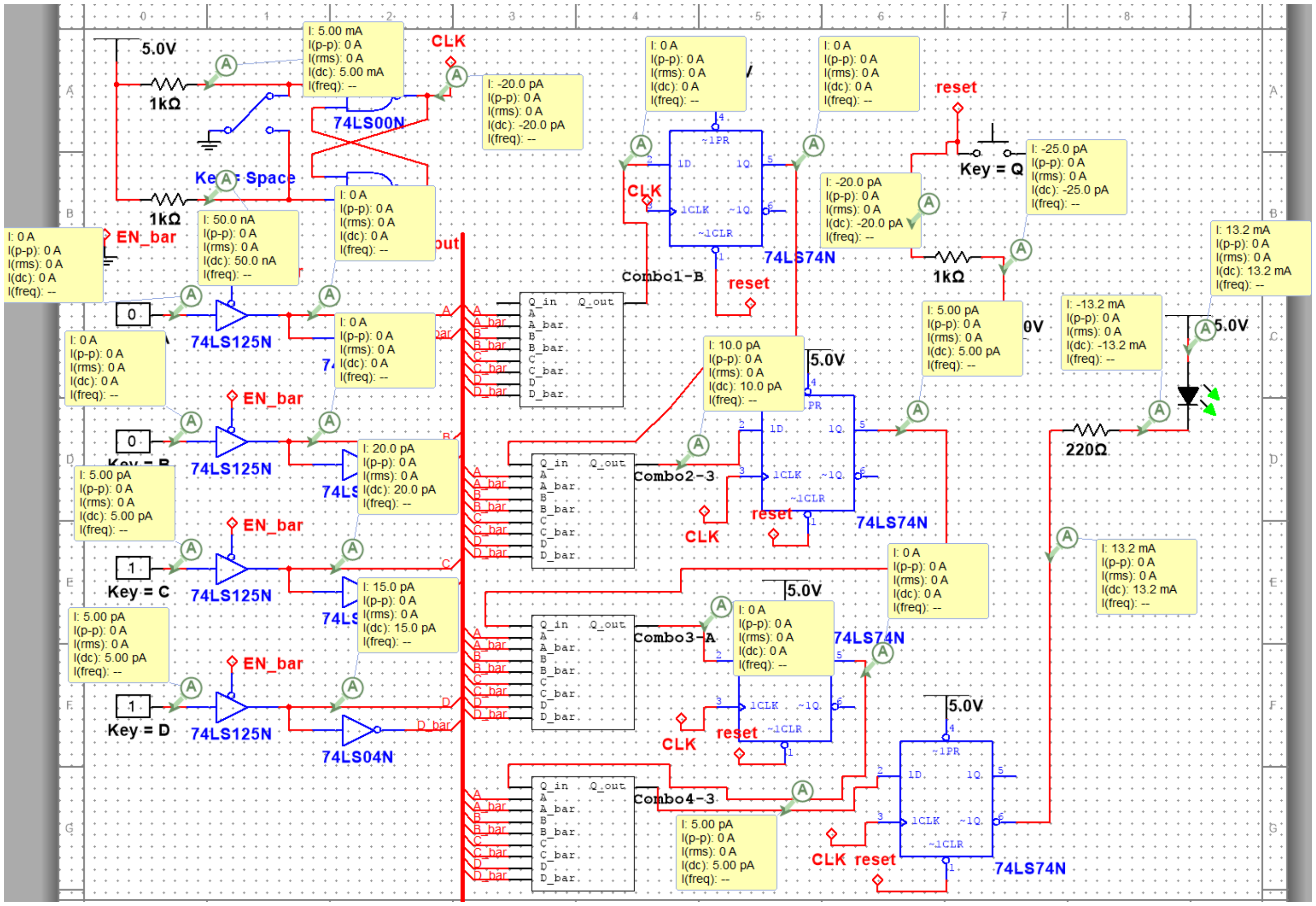

Figure 11: Current Measurements of 4-bit Digital Lock

Figure 11 shows how I set up the current measurements. I attached current probes to every input and output that transmits 1s & 0s across the DFFs and inputs. The highest draw of power is the LED and the CLK. Assuming we can change the components of the CLK, we can reduce the power draw on the CLK by changing the resistors to 10k ohms instead of 1k ohms, which can be seen in Figure 12a. I also went through the input cycle again and the lock still operates correctly. You can also change the resistor going into the LED to 440 ohms, from 220 ohms, which can be seen in Figure 12b, where I also went through the input cycle to show the lock still operates properly. If you change the LED resistor any higher than 440 ohms, it doesn't allow the LED to turn on, IE the lock doesn't unlock, which can be seen in Figure 12c. In Figure 12c, I also reduced the CLK resistor back to 1k ohms to ensure that the LED resistor was the sole cause of failure. **Therefore, I have determined that to optimize power consumption, you would need to increase the CLK resistors and the LED resistor. Doing this will reduce the CLK power consumption by 90% and the LED power consumption by 50%.**

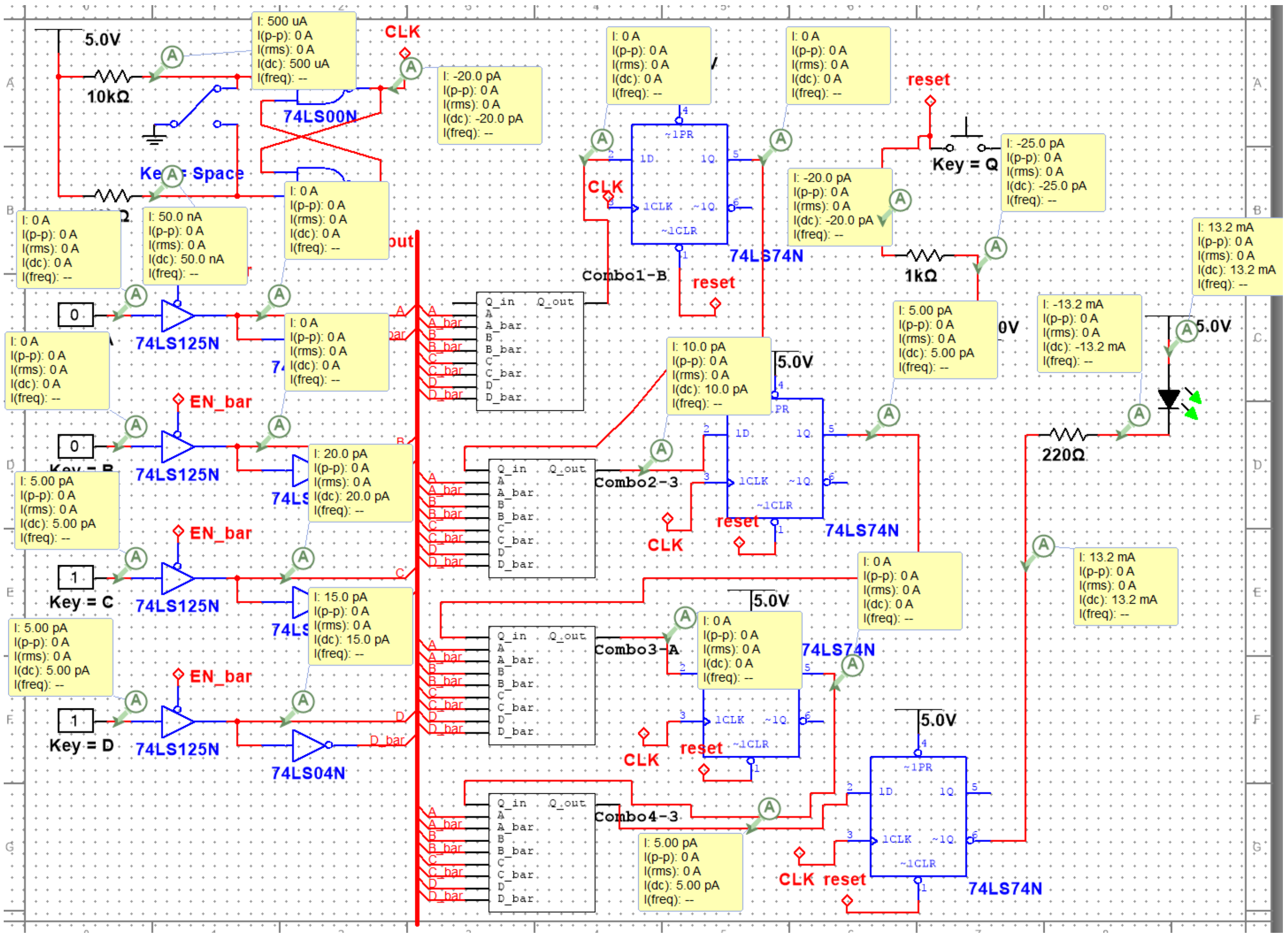

Figure 12a: 10k ohm CLK resistor

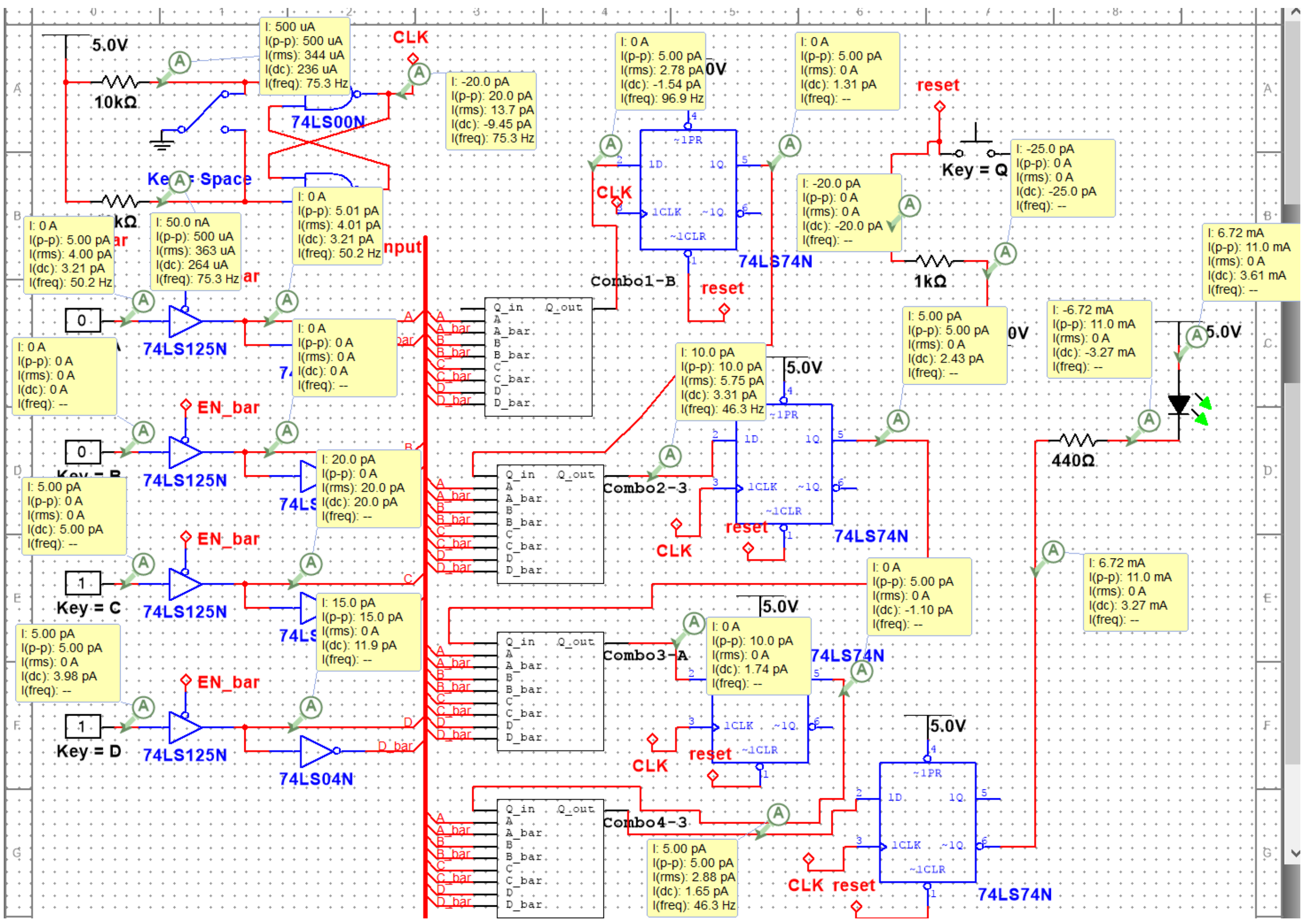

Figure 12b: 10k ohm CLK resistor and 440 ohm LED resistor (Optimal Power Consumption)

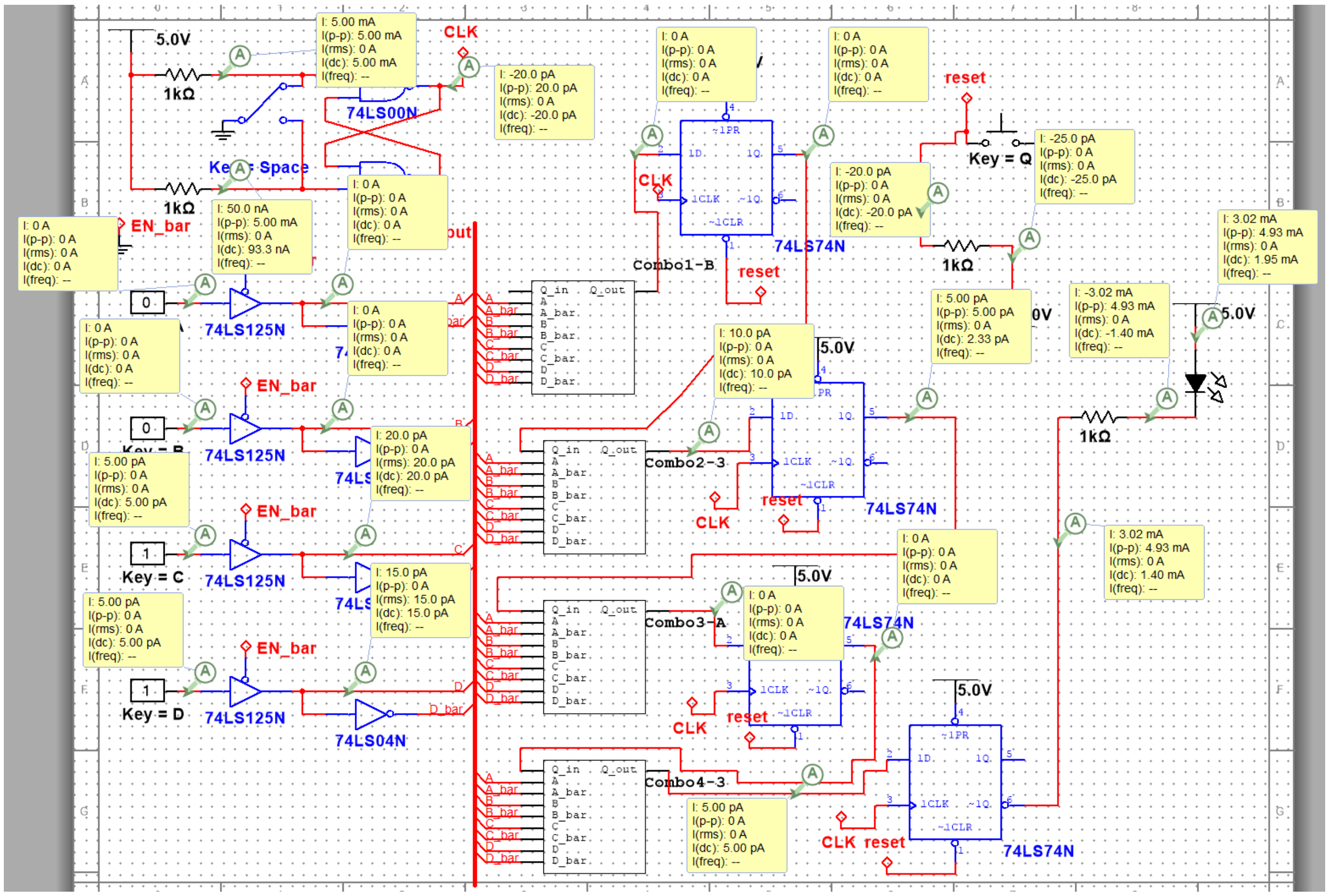

Figure 12c: 1k ohm CLK resistor and 1k ohm LED resistor