ECE 385

Fall 2020

Experiment #2

Data Storage

Alex Wen (acwen2), Manav Agrawal (manava3)

Section ABF

Xinbo Wu

Introduction

This lab is designed to implement a 2 bit by 4 words data storage through the use of CMOS logic; more specifically, the storage implementation utilized shift registers as the units to hold the data. The purpose of this lab is to design the storage units in such a way that it can take the concept of random access memory (RAM) in storing four unique addresses and alternatively design a similar combinational logic through the understanding of CMOS.

Circuit Operation and Implementation

The combinational logic circuit implements a 2-bit word from a storage access register (SAR) to the storage buffer register (SBR) through 2 functions - fetch and store; alternatively, data can be directly inserted into the SBR through the load function. The inputs are selected using 7 DIP switches for SAR0, SAR1, DIN0, DIN1, and the functions STR, FETCH, and LDSBR.

(1) The circuit first implements the load function (LDSBR). When the circuit implements the LDSBR, it takes a 2-bit word from the data-in (DIN0 and DIN1) inputs and loads them directly into the SBR; the bits that are 'high' are shown with the DIP LEDs. When loading data into SBR, no other functions should take place, so STR and FETCH are set on 'low' while SAR0 and SAR1 do not matter.
(2) The contents of the SBR is then *stored* in the address specified by the SAR. As such, a memory address is determined by 2 switches (0 and 1 bit) and the STR switch is set on 'high'. The data bits in the SBR are then respectively stored into the shift registers at the selected address. As such, when data is stored, LDSBR and FETCH are set to 'low' while the selected address is implemented through SAR0 and SAR1. DIN0 and DIN1 do not matter. So when the LDSBR and FETCH is off and the store is turned on, at the first rising clock edge, the counter increments and compares the value with the comparator. If it is equal, the output from the D Flip Flops are added to the shift register. It can take upto 1-4 rising clock edges before the counter and the address pins find a match.

The implementation of (1) and (2) is done for each word, and the circuit can hold data for 4 words for the 4 separate addresses (00, 01, 10 11). These 4 separate addresses are kept in track with a synchronous counter to relatively assign the addresses. As such, the process of loading and storing data can be done up to 4 times.

(3) The 2-bit words can be *fetched* from the shift registers (storage units) at the selected address determined by SAR0 and SAR1. When FETCH is 'high', the SAR is compared with the counter values. After one to four rising clocks four edges, when the counter

values and the SAR1 and SAR0 match, the content of the selected address is written to the SBR. LDSBR and STR should be set to ‘low’ and DIN0 and DIN1 do not matter.

While implementing the functions, the storage units aptly must maintain its data hold when none of the functions are implemented, so the SBR must maintain the data consistently when FETCH, STR, and LDSBR are held at ‘low’.

The general operation of the circuit is implemented through the logic of the block diagram (Figure 1).

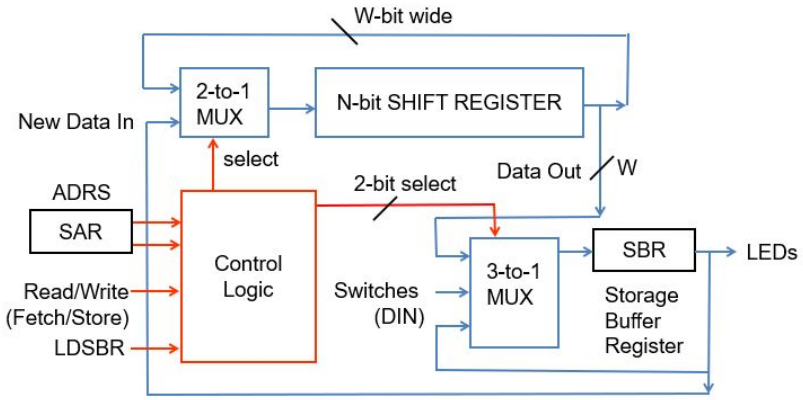

Figure 1: Block Diagram of the Storage Unit

The SAR is fed into a comparator, which compares the input with a counter which keeps track of the addresses in the shift registers. The result of the comparison is implemented into the control logic, which decides which data are to be processed through into the shift registers and the SBR. A 1-bit select signal is implemented to determine which data is to be put into the 2-bits shift registers at a selected address. A 2-bit select signal is to determine what kind of data the SBR should take. As such, multiplexers are implemented into the design to take care of the selection.

When the necessary data is implemented into the circuit, its output is controlled by the SBR, which is utilized by a flip flop to store the signals. In this case, a flip flop is used so that it’s ready to change the state of signals depending on the data inputted into the register.

Since the storage unit can hold up to 4 words, a synchronous clock cycle is needed for the transition between addresses. The clock cycle is implemented with a debouncing switch configuration as shown. So in summary, a comparator, a counter, logic gates for control logic, 1 2:1 MUX and 1 3:1 Mux, 2 shift registers, a D Flip Flop and around 8 switches with clock being debounced. The LEDs will be used as an output display.

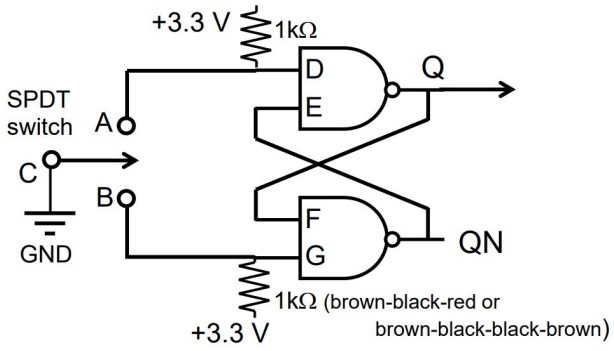

Figure 2: Debouncing Switch

The implementation of the debouncing switch is done with logic NAND gates - it can also be done with SR Latches - but the inventory dictates that logic be used (since no SR Latches are available). The pull-up resistors are included to prevent connecting a short between power and ground when the switch is flipped.

Finally, due to the configuration of transistor chips, decoupling capacitors are allocated near each of the chips from Vcc to GND, such that it can delay the slight glitch in which the power and ground is connected when the chip logic is turned on or off.

Control Unit, Design, and Circuit Schematics

The control unit dictates what data enters the shift registers and what data enters the storage buffer register. Therefore from the intuition and explanation above, the truth table for the select signals for the 2 different multiplexers are below.

The first is a quad 2:1 multiplexer that enters data into the shift register which was shown above in the block diagram. The select signal will depend upon the STORE signal and whether the address pins are equal to the counter which is relatively keeping track of the address. The purpose of this is when the store is active high and the address that we need to store it at is found, then we load the data into the shift register.

A=B From Comparator (E)	STORE Switch	Select Signal
0	0	0
0	1	0
1	0	0
1	1	1

Table 1: The Truth Table for the Select Signal of a Quad 2:1 Multiplexer

If we observe from the truth table, it represents an AND gate. This can be verified if a 2x2 k-map is drawn to obtain the logic expression.

	STORE = 0	STORE = 1
E = 0	0	0
E = 1	0	1

Table 2: K-Map with highlighted term

The expression for this select signal A = E AND STORE = ES

This is only one part of the control logic. The other part of the control logic is to control the 2 bit select signal for a 4:1 multiplexor. There are multiple operations in regard to the SBR.

E	FETCH	LDSBR	SELECT SIGNAL (S1S0)	Meaning
0	0	0	1X	Maintain the SBR DATA
0	0	1	01	Load the new data from the switch inputs
0	1	0	1X	Since we don't have the correct address X, maintain SBR Data
0	1	1	1X	Even though this will never happen, let it allocate to maintain SBR Data
1	0	0	1X	No operation related SBR, maintain data
1	0	1	01	Load in the new data
1	1	0	00	Now the data can be correctly fetched from the SBR as A=B and FETCH is active high
1	1	1	1X	Undefined behaviour hence maintain SBR Data

Table 3: Truth Table With meaning for 2:1 Multiplexor

To obtain the logic expression and make the logic for it, the k-maps are constructed.

	FL = 00	FL = 01	FL = 11	FL = 10
E = 0	1	0	1	1
E = 1	1	0	1	0

Table 4: K-Map of the S1 Select Signal

	FL = 00	FL = 01	FL = 11	FL = 10
E = 0	X	1	X	X
E = 1	X	1	X	0

Table 5: K-Map of the S0 Select Signal

If we look at the expression for the select Signal S1: We get this expression (FL)' + FL + E'F. These were 3 loops of 2 elements. The second K-Map had a lot of don't care outputs due to the design choices for the multiplexor hence the easiest loop will be to circle when LDSBR = 1. Hence this is the second part of the control logic.

Now, this needs to be constructed with logic gates. The first part of control logic had ES or E AND STORE. This is a simple AND gate but the circuit CMOS chips has NAND do this can be constructed by two NAND. This will be shown in the diagram below.

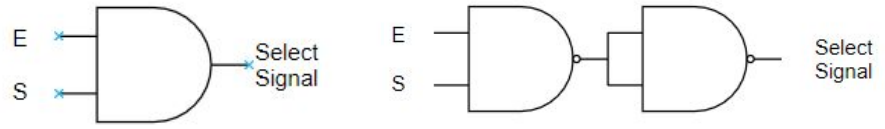

Figure 3: Gate Diagram

The second part of the logic has a larger expression. S1 = (FL)' + FL + E'F. This has the presence of AND/OR. We can convert to NAND gates but that will use a lot of NAND and inverters. So this expression can be simplified in a different way. If seen, (FL)' + FL will give the same output as a XNOR gate. This simplifies to (F XNOR L) OR (NOT(E) AND L). If we convert the AND to a NAND and bring another inverter to the OR's input then we can transform this to (F XOR L) NAND (NOT(E) NAND L). The NOT (E) can be converted with a single NAND gate. This saves 2 NAND Gates. The diagrammatic expression and simplification is shown below. (The comparator signal is shown as X as the diagram maker had a problem with E)

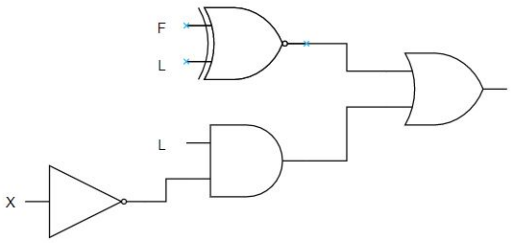

Figure 4: A simplified version from AND/OR which includes XNOR (Output: Select S1)

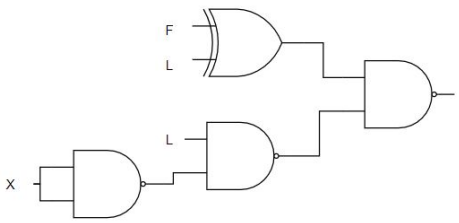

Figure 5: Only NAND and XOR Schematic for S1 of 4:1 Mux (Output: Select S1)

This will be the schematic for the S1 of 4:1 MUX. The S0 is a direct LDSBR signal hence we do not need the diagram for that. Now the control logic is complete. This completes the entire block diagram as shown in Figure 1.

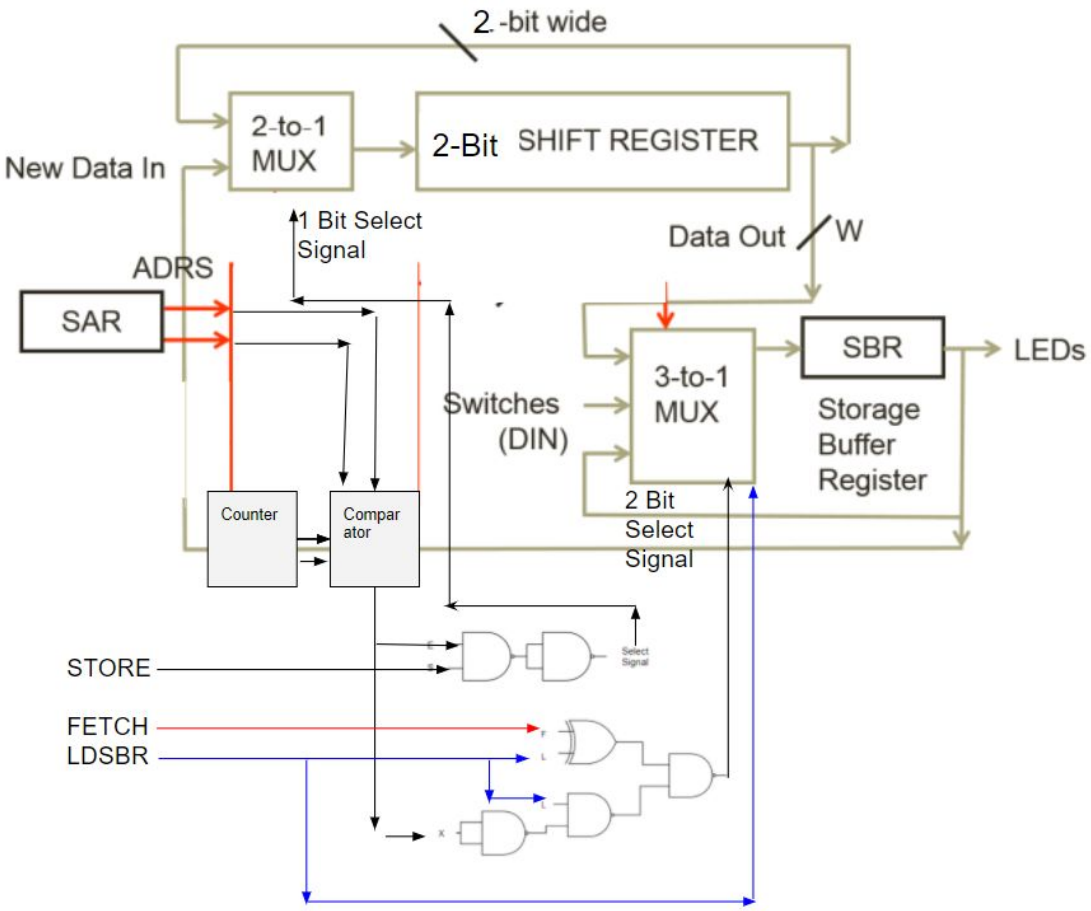

Figure 6: Updated Block Diagram with more control logic information

Before making the entire Fritzing schematic and breadboard view. There are some design choices which need to be taken. There were many CMOS chips that were investigated.

The first design choice was to choose between 74161 chip or 74193 CMOS chip. This experiment used the 4-bit synchronous counter 74161 chip rather than the asynchronous ripple counter. This is because the circuit is clocked and needs to be synchronous. The second design choice was to choose between D flip flops and JK Flip Flops. This experiment, D Flip Flop was utilized as in the SBR, there is one 2 bit input. Last design choice was choosing between the 74194 and 74195 shift register. The 74195 shift register had two inputs so the 74194 shift register was chosen. Although the 74194 shift register has modes represented by bits S1S0, since they were always shifting, the mode was just connected to power and ground.

Now the fritzing schematic can be assembled.

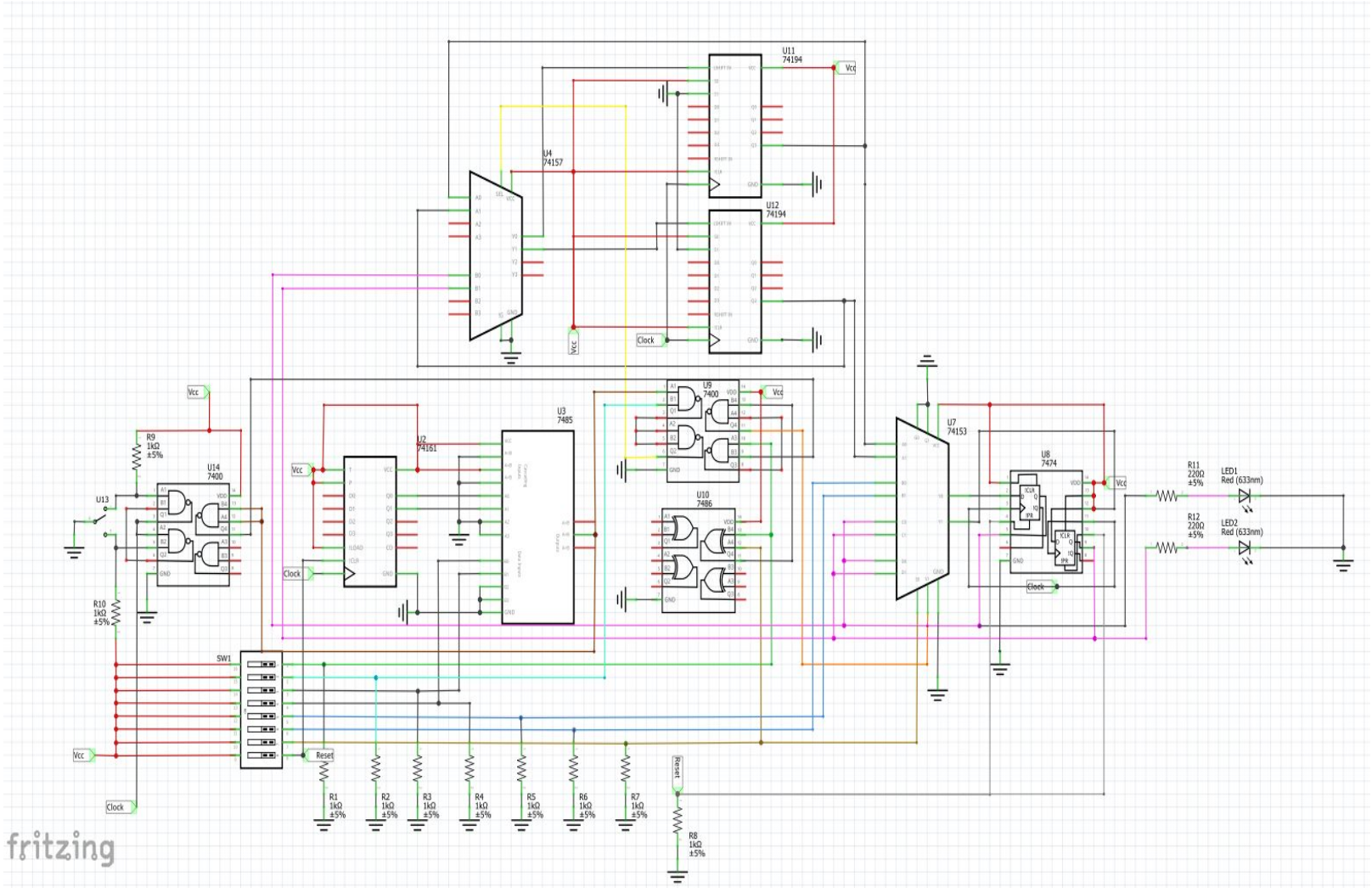

Figure 7: Fritzing Schematic (https://drive.google.com/drive/folders/1Tk8P7HcX6jvzBdrzNSSUUHVjP2KKS4LJ?usp=sharing - Clear Image found at this Google Drive Link Also)

Color	Meaning	Switch Number	Function
Green	Fetch Switch Inputs	1	FETCH
Cyan	Store Switch Inputs	2	STORE

Ochre	LDSBR	3	SAR 1
Brown	A = B from Comparator	4	SAR 0
Orange, Yellow	Select Signals from the control logic	5	DIN 1
Red, Black	Power, Ground	6	DIN 0
Pink	Output visible on LED	7	LDSBR

Component Layout

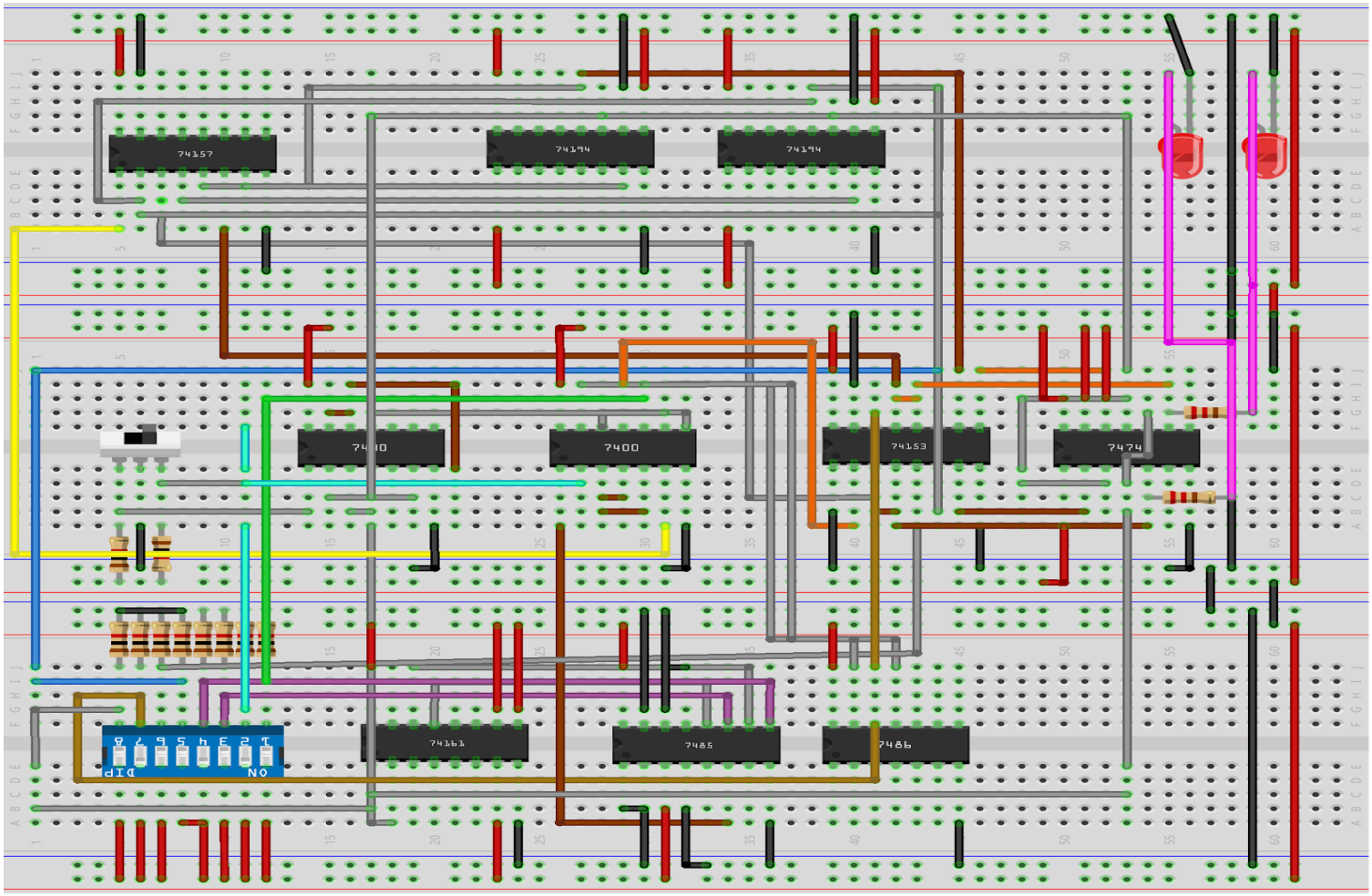

Figure 8: Breadboard Fritzing View

This exact Fritzing breadboard was not implemented on the real breadboard as some space choices and rearrangement needed to be made.

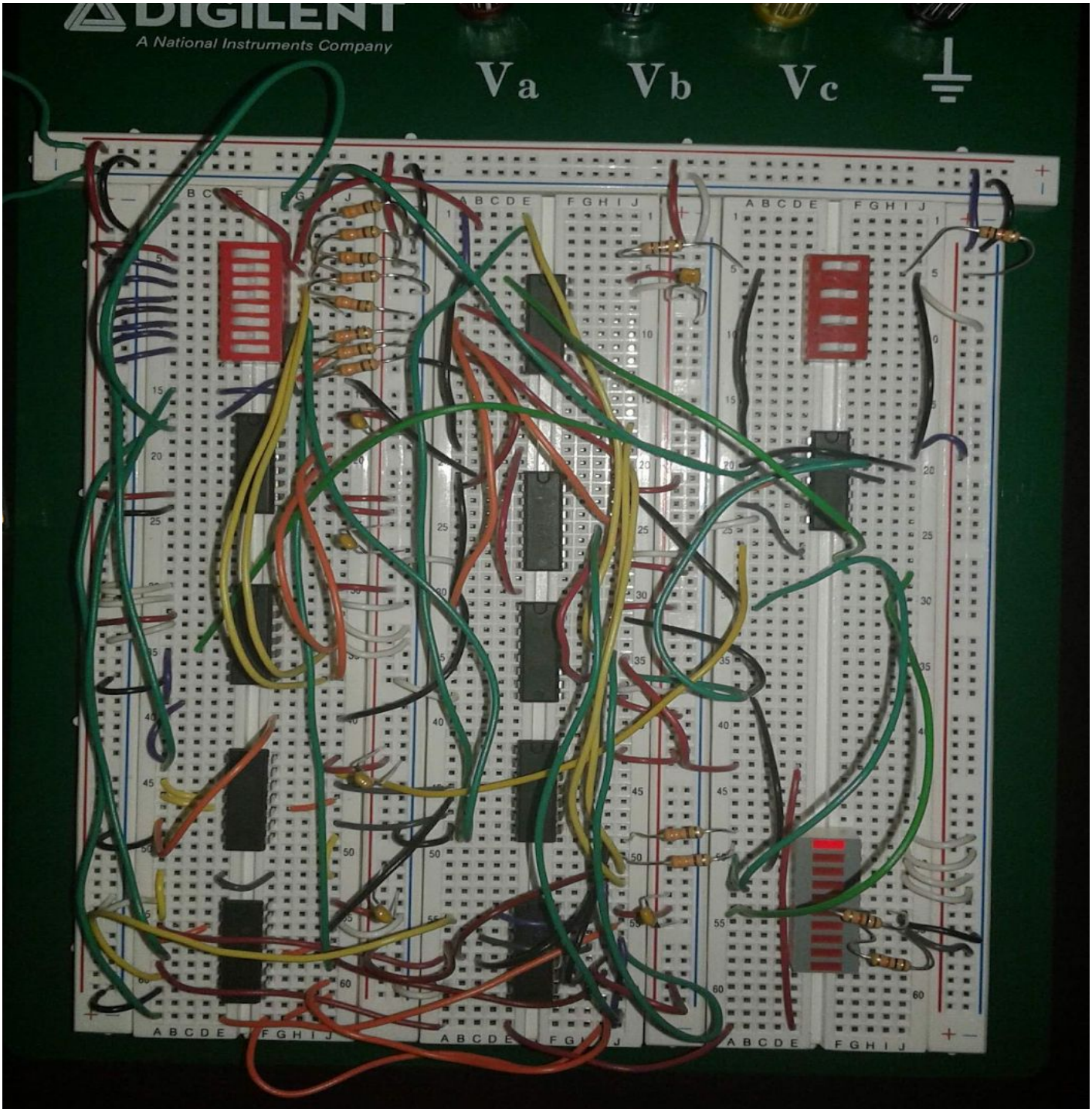

Figure 9: Actual Experiment configured and demonstrated

It was time-consuming but eventually the circuit was assembled to what is shown as above.

Debug Process

There were some errors made in the experiment. For the debug process, the outputs of the shift register connected to the LEDs to judge and identify where the problem was located. The experiment was tested one by one. There was block testing done initially on a smaller group of chip components to see if they are working fine and then assembled. During the LDSBR, the circuit did not encounter many issues as the boolean logic was right and the output just needed to go to the multiplexer and into SBR. The main errors that were identified were during store and fetch. As the store into the shift register was tested, it was identified that there were some possible errors. So we first started checking the output logic inputs and identified that the signal from the comparator was not working properly. So the counter and the comparator were assessed. It was found that some of the inputs of the counter need to be grounded or connected to power. Inputs like ENT need to be connected to power. After some debugging and making sure the counter and comparator were working properly, the store seemed to work fine. Initially while testing fetch, the SBR got the data from SBR but the data of SBR was constantly changing meaning it was constantly fetching data from the shift register rather than waiting for A=B. Then it was realized that a major mistake was conducted as the comparator signal was forgotten to be inverted by 1 NAND gate. After that, the experiment seemed to work but it was not very consistent as the bit data kept on changing in the 194 register and the debouncing switch was also needed to properly be configured. It took some time to start to make the circuit work for the demo. Some of the key ideas were to make sure each CMOS chip is properly powered and grounded, there is no short circuit and we are using temporary LEDs to debug the process. Incremental assessment to find errors was the main idea to find and debug mistakes.

Post Lab Inquiries

What are the performance implications of your shift register memory as compared to a standard SRAM of the same size?

The performance implications differ when the shift register memory or the standard SRAM of the same size is used. The shift register design is very simple but it is not very area efficient due to it having D-Flip Flops. Addressing the shift register and finding the cell also becomes more difficult whereas for larger data, the Static Ram is much more area efficient. The implications that it comes to is area complexity and simplicity. In terms of performance, if a data needs to be accessed from a specific address, it can take 1-4 clock cycles to access it and for data of 128 or 256 bits, it becomes 1 to 128 or 1 to 256 clock cycles. The shift register starts to lose its performance and area efficiency whereas the SRAM of the same size has transistors in each cell and addressing and storing it becomes simpler and faster.

What are the implications of the different counters and shift register chips, what was your reasoning in choosing the parts you did?

As discussed earlier, there were two main choices in the counters and the shift register chips. The first one was choosing a 4-bit synchronous counter versus the asynchronous ripple counter and the second one was choosing between the 74194 shift register and 74195 shift register chip. As explained earlier, the asynchronous ripple counter has temporary glitches before it settles to a value. Although everything else is clocked, it will be better if the synchronous clocked 4 bit binary counter is used as it does not introduce any temporary glitch after a rising clock edge. The second choice was choosing the 74194 shift register over the 74195 register. both registers would have worked for this lab. But 74195 has J - K which was responsible for data and the shifting mode whereas 74194 shift register provided a more convenient way of organizing the input and shifting mode and not mixing them together. The slides also explained the usage of 74194 counters very well so 74194 was picked over 74195.

Conclusion

In conclusion, this lab was to explore and create a data storage (RAM) for 4 2-bit words which was designed through shift registers. A storage buffer register was also utilized which was used as a way to input new values into the shift register as well as fetch the data from the shift register. Through carefully examining the function of control logic, shift registers and other inputs, k-maps were devised which helped to create the circuit schematics which enabled the construction of the circuit. Nowadays, SRAM is created from latching flip flops where around each cell has 6 registers which stores a single bit and optimizes area over the shift register. This experiment teaches the use of shift registers to store bits, the process of loading and unloading based on relative addressing by a counter and better debugging skills and CMOS logic for the future.