ECE385

SPRING 2019
Lab #2

Data Storage

Zuling Chen, zulingc2
Austin Lee, sal3
Section ABE

Introduction

The main purpose of this lab served as a review of ECE120 concepts regarding multiplexers, flip-flops, shift registers, simple circuit design and combinational logic. In this lab, we were to build a 2-bit four word shift register storage unit. The circuit had to be able to load data into one of four memory addresses represented by clock cycles. It then had to be able to fetch that two bit data from one of the four memory addresses.

Description of Circuit Operation

The first thing that we had to design into the circuit was a part that handled the addresses for where data would be put in. To do this, we made use of a counter and a comparator. The least significant bits of the counter would go through 00, 01, 10, and 11 while the comparator took input from switches and then sent out a high signal whenever the counter output matched the switches input. This served the same purpose as finding an address that matched where we wanted to place data as the counter was in sync with the overall clock of the system. As data is constantly being shifted in the shift registers, whenever we have a match with the comparator, we chose that as the correct "address" and then we perform either a store or fetch operation with 2 bits at that address. However, if the address is matched and neither of those operations are selected, then we remain in the state where the data in the shift registers is still being shifted. To make sure that the data is not lost though, we make sure that in that null function state, data that is shifting out is shifted back in.

The first operation that our circuit had to be able to do was load data into the SBR. To select this operation, we had the control unit take in the input from the switches and pass it through some control logic to choose the first output of the MUX. This occurs when the LDSBR switch is at a high and STORE and FETCH switches are low. It should also be stated that for every operation on one bit, the same is being done to the other bit in a identical parallel structure (MUX, SBR, registers, etc.). By selecting the 00 output of the MUX, we take data directly from two switches that serve as our data input and then put it into our SBR represented by LEDs.

From the LOAD function, we were then to move onto the STORE function. STORE happens when LDSBR and FETCH switch are at a low and STORE is at a high. The function will take the address specified by two address-indicating switches and store whatever data is currently being displayed in the SBR to the shift registers at that address.

Lastly, we had to implement a FETCH function which when selected should take whatever data is in a specific memory address and place it into the SBR. This should occur when the FETCH switch is at a high and the LDSBR and STORE switches are at a low. Regardless of what data was previously in the SBR, the SBR should now be displaying the data inside the shift registers at that address, or rather, at that position of the counter.

Design Considerations

4:1 MUX

The 4:1 multiplexers (SN74153) in this lab was used as a way to select one of the 3 functions that the circuit had to perform. Because there were two bits of data, we had to use two multiplexers to select the two bits of data that would be passed onto either the SBR or the shift registers. After taking input through two function switches and passing them through some control logic, we then took the two select bits from that control unit to select our outputs. For our inputs, we needed to have new data straight from the switches, data from the shift registers and data from the SBR. Technically for this part of the design, we only needed 3 functions but because of the lack of access to a 3:1 multiplexer, we ignored one of the inputs to the 4:1 and made that state unreachable given the bounds that only one function would be selected at a time (output at 11 was ignored).

2:1 MUX

Two 2:1 multiplexers (SN74157) were used to control what data was being shifted into the shift registers. Because we wanted the shift registers to be continuously shifting with the clock cycles, we needed to make sure that data that was being shifted out would not be lost. To prevent this, we wired up one of the inputs to the 2:1 MUX to the output of the register. This way, when select was LOW, data that was being shifted out of the registers was going right back in. However, when select was high, we had data from the SBR going into the shift registers. This should only occur whenever we call the function STORE.

SBR

For the purposes of this lab, we used two flip flops (SN7474) connected to two LEDS as our SBR. The LEDs would display the data inside the flip flop at every clock cycle. Whenever we called the FETCH operation, data from a chosen address would be displayed on the SBR. Whenever we called the LOAD operation, data from the input switches would be displayed on the SBR instead. Because the circuit needed to be a 2 bit four word register storage unit, we needed to make use of two flip flops and two LEDs, one LED to represent the most significant bit and one LED to represent the least significant bit.

Shift Registers

To hold data, we made use of two shift registers (SN74LS194A). The lab specified that the shift registers must be continuously shifting, so we connected each register to a clock. To represent addresses in the shift registers, we had a comparator and a counter. When the comparator and counter matched, we would count that as having matched addresses. The shift

registers' main purpose was to continuously shift old data back into itself whenever STORE was low but then write new data in from the input switches whenever STORE was set to high.

Control Unit and Control Logic

The control unit used for this lab had to do a few things. The first thing it had to do was make a select to the 3:1 MUX for which function needed to be implemented, FETCH, LOAD, or STORE. For our design, we wanted the LOAD functionality to correspond to the 3:1 MUX's S1S0 select of 01 and the FETCH to be 10. For STORE or when all three functions were off, the MUX stayed in the 00 output. We also wanted S0 select bit for the 2:1 MUX to only be high whenever STORE was flipped. Going with this, we wrote two truth tables. We worked under the assumption that only one of the functions would be called at a time.

Truth Table 3:1 Mux

LD	FETCH	COMP	S1	S0
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	X	X
1	1	1	X	X

This corresponds to the K-maps of S1 and S0:

LD FETCH/CMP(S1)	0	1
00	0	0
01	0	1
11	X	X
10	0	0

LD FETCH/CMP (S0)	0	1
00	0	0
01	0	0
11	X	X
10	1	1

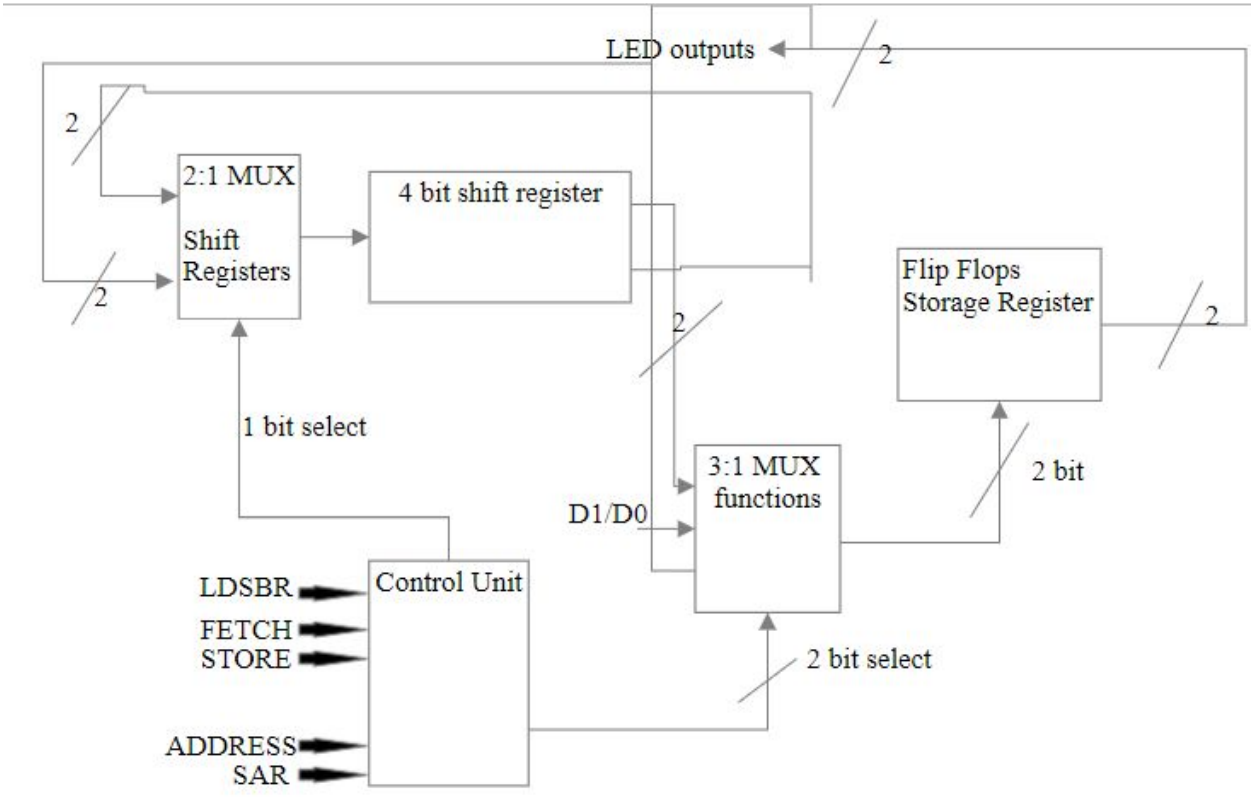
From these K-maps, we saw that we could make S1 = FETCH and COMPARATOR and S0 = LDSBR.

For the S select of the 2:1 mux, we figured that we only wanted it to be enabled whenever STORE was high and the comparator matched. For this part, it boils down to an AND between the two inputs.

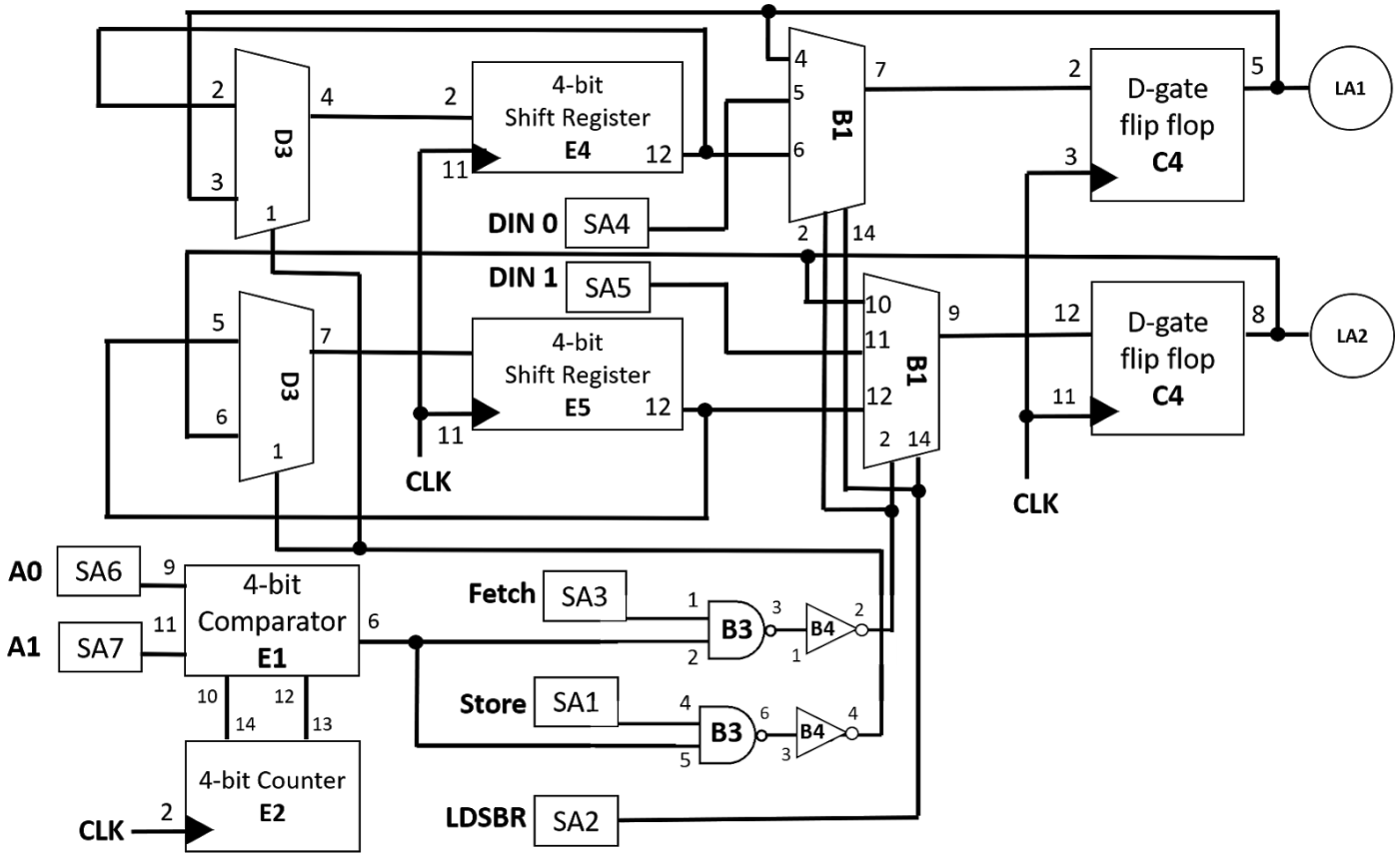
STORE	COMPARATOR	S
0	0	0
0	1	0
1	0	0
1	1	1

Because it just becomes a simple AND, we did not need a K-map for this part. We set the final select for the 2:1 MUX to just STORE and COMPARATOR.

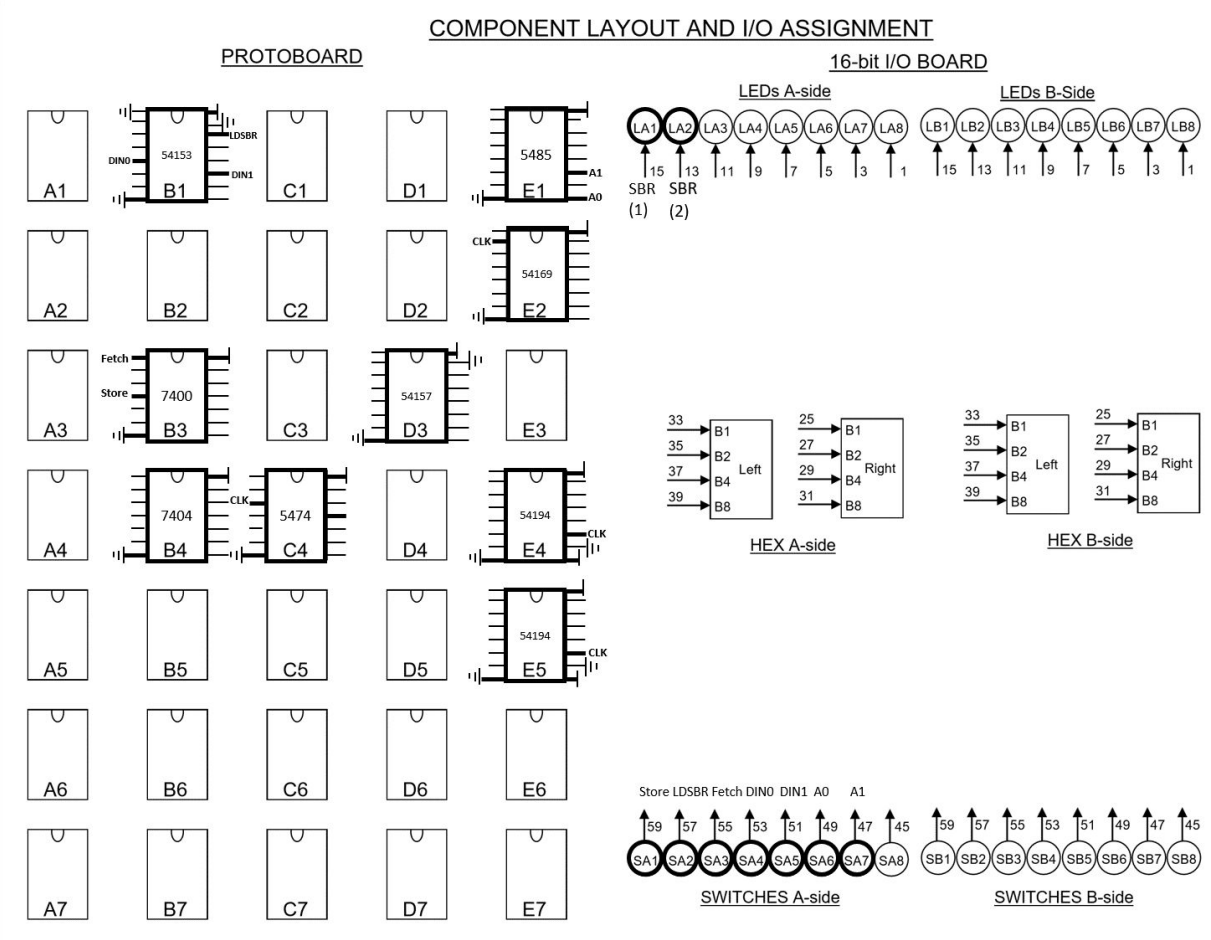
Block Diagram



Logic Diagram



Data Sheet



Final Considerations

During the finalization of the circuit, we ran into issues when dealing with bugs. The first issue we had was our control logic being incorrect. Because of poor organization and documentation, we switched up the inputs to the 4:1 MUX and ended up creating our logic so that the combinations we got from our FETCH, STORE, and LDSBR switches did not match up with the MUX inputs that we wanted to pass on. However, a simple backtracking to our control logic fixed this problem relatively easily. Another bug that we encountered was that when plugging in the wires to the clock of the flip flop, we saw that it interfered with the counter and comparator portion of the circuit. We initially used a switch to serve as our clock so that we could trace out all the steps in our circuit, but once we plugged in the function generator for the clock signal, that's when the issue presented itself. We solved this problem through the help of a

TA, who informed us that our function generator was not on High-Z mode. We are not sure why this solved the issue as the TA provided little explanation besides us having a "cursed clock". During the final parts of debugging our circuit, our AND gate chip burned out and we had to switch it out. One of the last bugs that we had to deal with was when we tried loading and writing 01 or 10, our SBR would continuously alternate lit LEDs. We found the solution to this by switching up the input wires to the MUX's; the output of one SBR was going into the other MUX, causing an alternating pattern.

Post-Lab Questions

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

Our storage unit would work at a slower rate compared to an SRAM since ours depends heavily on the clock cycle. The FETCH and STORE functions may at a maximum take four clock cycles before the correct data is displayed in the SBRs or written into the registers since our registers are built with serial inputs. On the other hand, SRAM is parallel load, works much faster and it's typical access time is about 10 ns.
(https://www.techopedia.com/definition/2814/static-random-access-memory-sram)

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

The main difference between the SN5495 series of shift registers and SN54194 series is that the former chip performs the right shift on the falling edge of the clock while the latter performs it on the rising edge. We made the decision to use the 194 chip because the other components of our circuit, namely the counter and flip-flops are all positive-edge triggered. In order to synchronously move the data between the circuit, our registers had to be positive-edge triggered as well. The different counters detailed in the data sheet are the 168/169 series, the 90/92/93 series, and the 192/193 series. We needed a counter that would provide a constant 4-state cycle (00, 01, 10, 11) in order for our data storing/fetching to operate consistently in minimal time. This would rule out the 90/92 series chips and the 168 series as those count in nonstandard ways (decade, divide-by-twelve, etc.). The 169 chip was ultimately chosen because it's two least significant bits reliably cycled through four states; it would count up to fifteen (maximum), then zero, one, two, etc. again. The 193 chips included a borrow/carry which added an extra state every four cycles-- the 169 series counter was the best choice.

Conclusion

In this lab experiment, we were able to implement a 2 bit four word register storage unit. Our circuit was able to successfully perform LOAD, FETCH, STORE, and we were able to continuously shift our shift registers without losing data. We gained an insight into how some simple storage circuits worked through completing this experiment. Through the process, we also learned the disadvantages and dangers of gating clock inputs; glitched signals and propagation

delays can disrupt the operation of the entire circuit. All in all, we were completely successful in achieving the objectives of this assignment.