# ECE 385

Fall 2019

Experiment #2

# Data Storage

Eric Dong, Yifu Guo
ericd3, yifuguo3
Section AB3, Thursday 2pm
Gene Shiue

**Introduction**

Our circuit contains shift registers as well as flip flops, these can hold temporary data which is a really rudimentary model of random access memory(DRAM). Given a specific address, our circuit is able to store desired data into shift registers, and we are also able to retrieve data back from an address that we chose. Our memory is able to store 4 words at a time, each of 2 bit length.

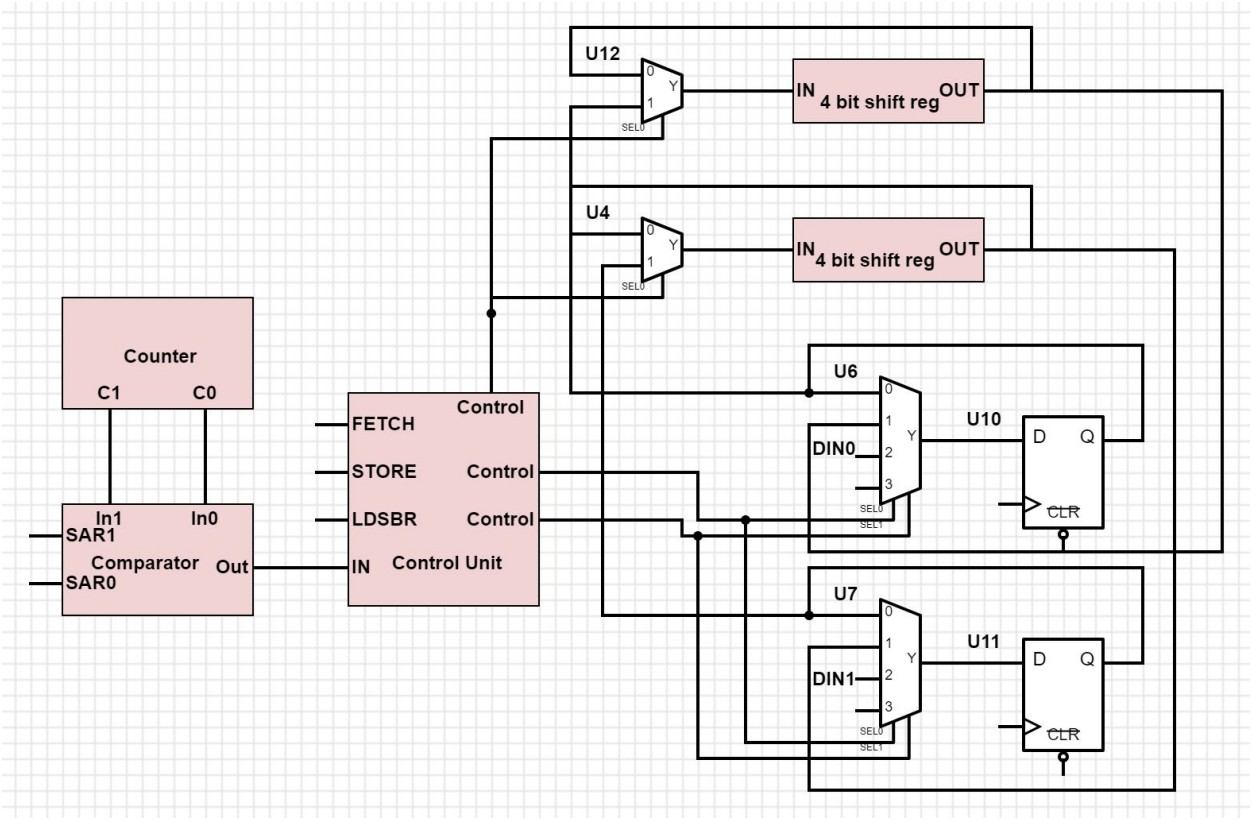
**Operation of the memory circuit**

1. Addressing is implemented when the counter's current count matches the desired address. The comparison is made by a comparator, when the comparator outputs a 1, we know that the current address is the one that we desire. The circuit performs a read from the input switches (DIN) when the LDSBR switch is a logic 1, the DIN data will be read into the SBR. Then, when the STORE switch is a logic 1, the data in the SBR will be stored into the register corresponding to the correct address that we chose.
2. In order to write data into the memory, we have to first choose the desired data, thus we first set the DIN1 and DIN0 switches into the desired data. Next, we have to load the data into the SBR, to do that, we have to turn on the LDSBR switch and toggle the clock by one cycle. Next, in order to load the data into the registers, we would need to turn off the LDSBR and then choose the address that we would like to write to. To do that, we have to set SAR1 and SAR0 to the address we want. Following, we turn on the STORE switch and toggle the clock until we reach the desired address. Turn off the STORE switch and now we have successfully stored the desired data into memory.
3. In order to read the data from the memory, we would first have to choose an address to read from, to do that we have to set the SAR1 and SAR0 to our desired address. Next, we need to turn on FETCH switch, while making sure that LDSBR and STORE switches are not on. Toggle the clock until the address matches with the desired address, then we do one more clock cycle for the data to get through the SBR flip flops. Now turn off FETCH and we have successfully read the data from memory.

**Written description and block diagram of memory circuit implementation**

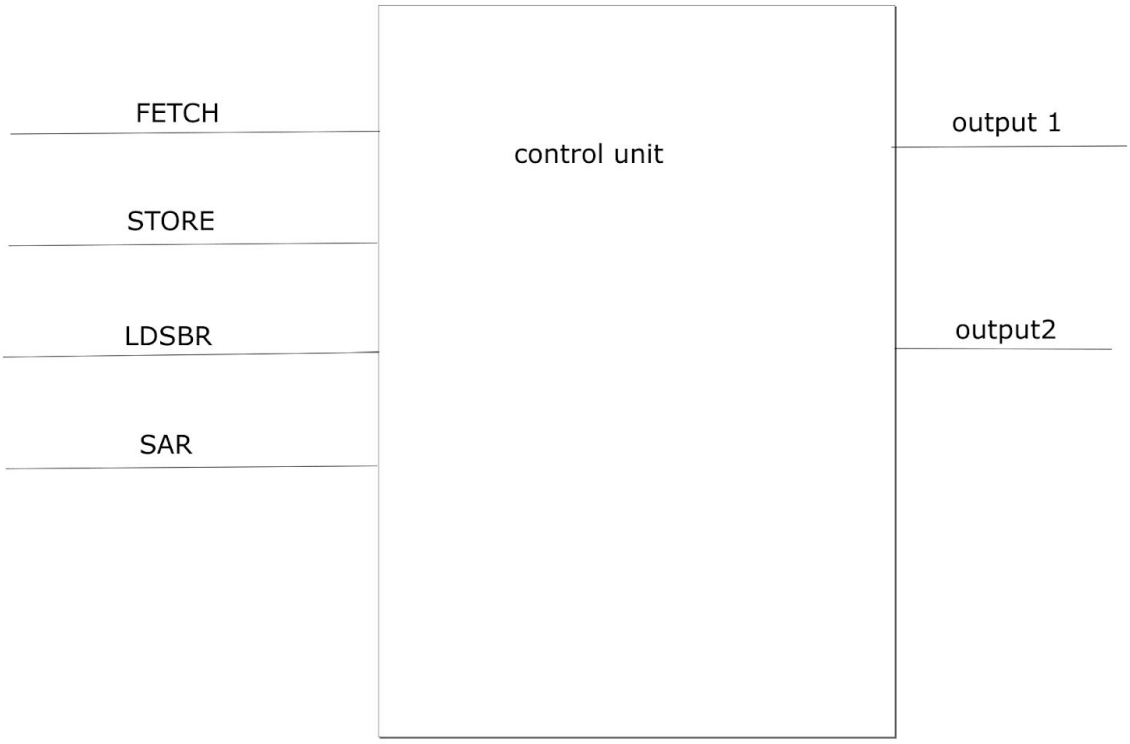
1. In order for the memory circuit to operate, we would need to have 2, four bit shift registers where the 4 words are stored. In order to create the buffer register, we would need two flip-flops to store 1 word at a time. To select what goes into the buffer register, we would need to use a 4:1 mux. Also, in order to determine what goes into the shift register, either cycle data or store new data in, we should use a 2:1 MUX. Finally we also would need various gates to implement the control unit which gives the MUXes the correct select bits. The control unit should consist of a counter as well as a comparator. When the counter's output address is the same as the input SAR, the comparator should output HIGH, allowing us to perform FETCH or STORE.

2. Block Diagram

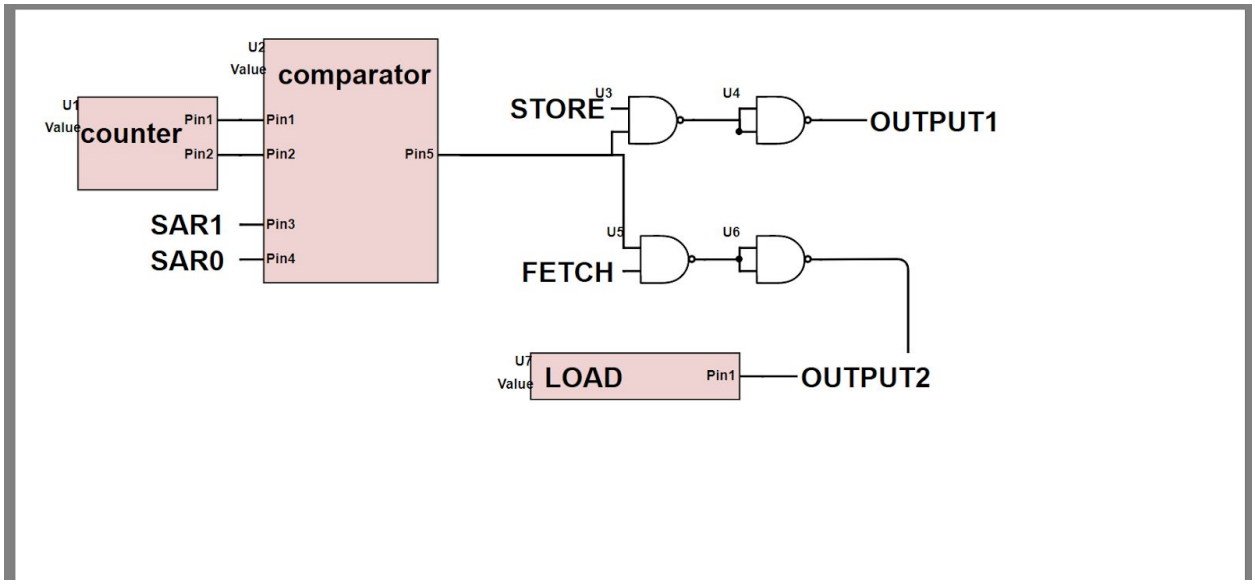



**Control Unit**

1. There are five inputs in our control unit, which are SAR1, SAR0, FETCH, STORE, LDSBR. The control unit outputs two signals. One is used to control the input of the shift registers, and the other one is used to control the input of the SBR. Those two signals are connected to several MUXs, which are two 2:1 MUXs and two 4:1 MUXs. The first output signal that is connected to the 2:1 MUX decides whether the shift registers should input new data or do the self-cycle with the old data. The second output signal is connected to the 4:1 MUX, which helps to decide whether the SBR flip-flop should do the self-cycle, input the data from shift register or DIN1 and DIN2.




2.

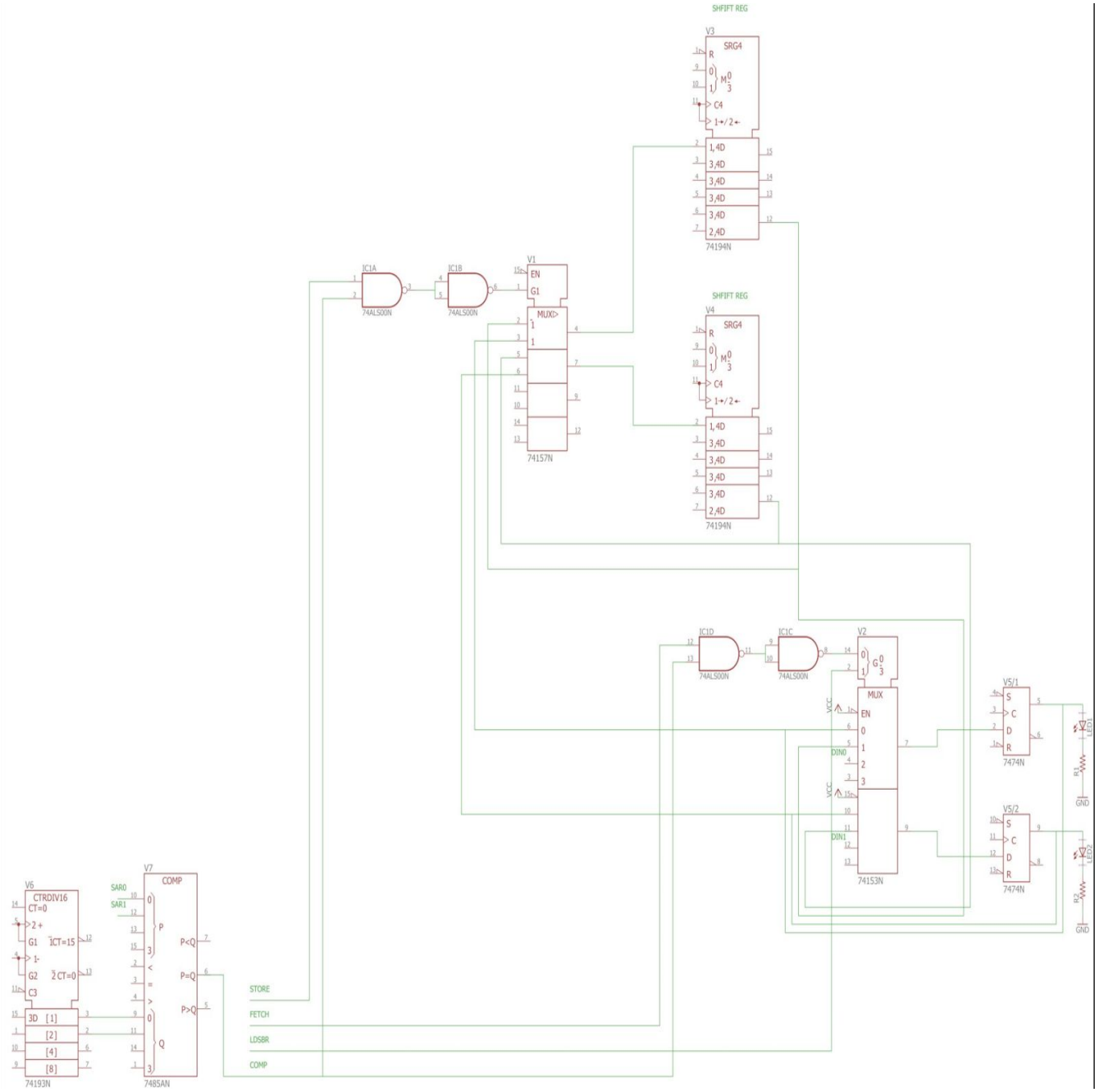



**Design steps taken and detailed circuit schematic**

1. No K-maps or truth tables used in this lab.
2. State diagram not needed

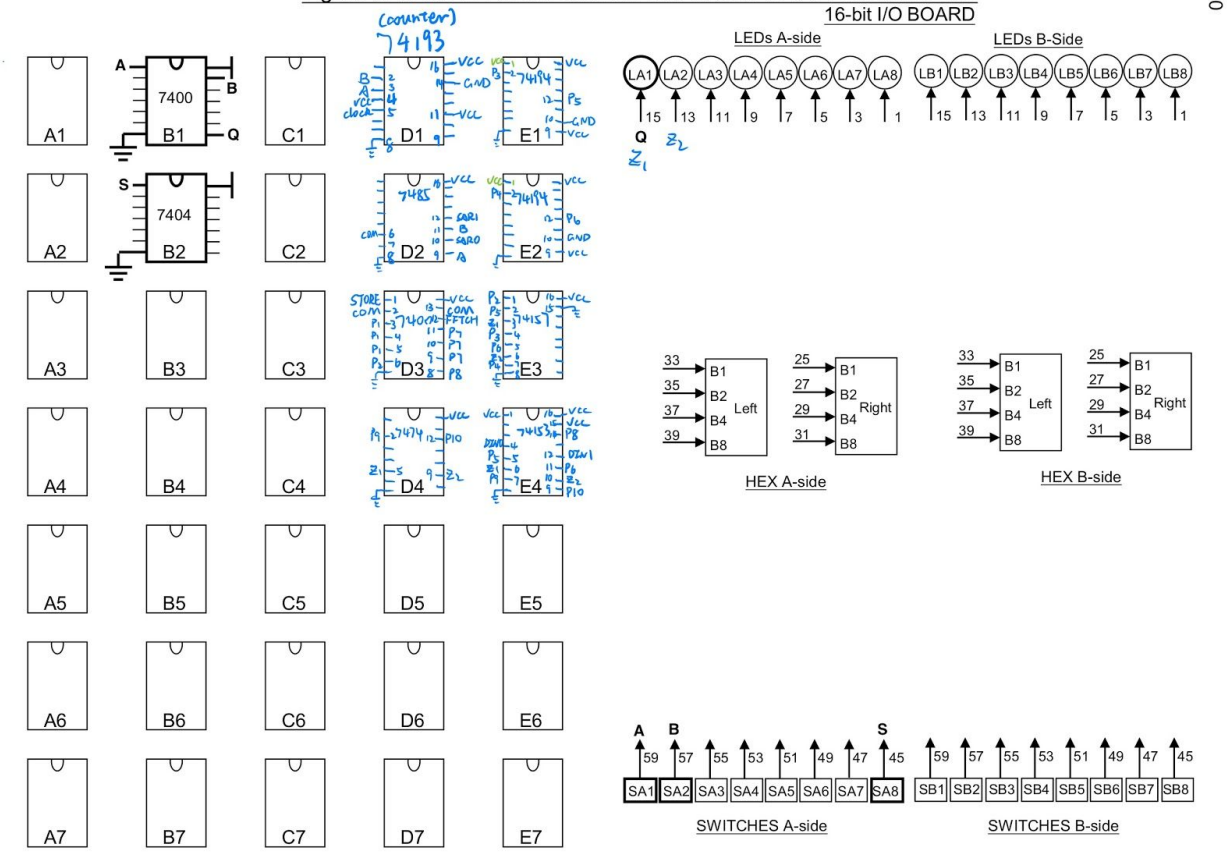
3. While designing the control unit, we tried to have the simplest circuit possible, and thus we did not need to have to use a K-map. For the 4:1 MUX, we set DIN to pin 2 so when select 1 pin is high, which is connected to LDSBR, data could be written in since select 0 pin has to be 0. We then connected data out from the shift register to pin 1 of the 4:1 MUX so that when FETCH is high and connected to select 0 pin, we can fetch data from the shift register since the select 1 pin has to be low when FETCH is HIGH. When both select pins are LOW due to none of the switches are on, then the data in the buffer self cycles which is what we want.

## Detailed Circuit Diagram



**Component Layout Sheet**

Figure 8. SAMPLE COMPONENT LAYOUT AND I/O ASSIGNMENT




**Bugs encountered**

1. For this lab, we did not encounter too many bugs since we first drew out the circuit on EAGLE and did the connections afterwards. One thing that we did run into was that we had a faulty 193 counter, it incremented from 3 to 0 on a negative edge, so to fix this issue, we simply switched out the faulty 193 with another one, and it worked nicely.

**Conclusion**

1. This lab was very interesting, it gave us an insight to how a storage unit might work. It was really interesting implement all the functions such as FETCH, STORE and LD. Everything in our lab worked as expected. I think we could've used fewer gates, but for the most part, we were accurate and didn't have to debug too much.

**PreLab Questions**

1. Only the clock needs to be debounced to prevent the switch from bouncing off its physical contacts, creating unintended clock edges, and causing the circuit to do unwanted moves. The other parts don't need to debounced since they will stabilize once the switch stabilizes.

**Post Lab Questions**

1. Throughout the process of the lab, the only bug that we ran into was a faulty 193 counter chip. We almost found this bug instantly since we hooked the output of the counter to the LEDs and saw the bug as we were testing the counter. We simply swapped the 193 for another one that we obtained from the lab and everything after that worked smoothly.
2. Our memory is limited by the speed that we have to run it at. Nowadays, SRAM can run up to around 4000Mhz while ours is capped by the gate delays as well as the chip itself. SRAM can have such performance due to the fact that their memory system is in matrix form, which means that they don't use shift registers to hold the data, but rather have the memory in a grid. When they want to access the data, they just need the coordinates of the memory and instantly they can retrieve it. For our shift register memory system, we have to wait up a cycle's time in order to fetch the desired data out of the shift registers, this is significantly longer than the SRAM speed, especially when the shift registers can hold more bits in them, a cycle's time is significantly longer.
3. We decided to go with the 193 counter since it was the most reliable, it is synchronous, while the 93, although easier to wire, is not reliable since it is asynchronous. We also did not use the 169 counter since it required 2 enable pins, and we were confused by how one of the enable pins was already HIGH even though we did not connect anything to that pin. We also chose the 194 shift register since that was the only 4 bit shift register that we were provided with. We just had to use 2 of those chips to create a 2 bit, 4 word shift register memory system.