ECE385

Spring 2020

Experiment #2

Data Storage

Yuantao Lu, Yizhen Lu

Section ABD

TA: Nicholas Cebry, Wenjie Pan

Introduction

In this lab, we used two 4-bit shift registers and flip-flops to represent a single memory unit. The memory can store four words with each word having width two. The memory has the operation of storing and fetching data, each time 2 bits. We designed a control logic with five input bits and three output bits used as selecting bits in the MUXs. The design process and connecting the circuit took a relatively short amount of time but getting familiar with various ships took us a much longer time.

Operation of the memory circuit

To match the input address with the relative spots in the shift registers, we use a counter to generate a periodic 2-bit up-counting sequence. Then we load both the address and the counter output into a comparator, which is wired in a way to output HIGH only when the two input matches. Since shift registers and the counter are connected to the same clock signal, so they are synchronized. Thus, the spots in shift registers always match one pattern of the counter. Therefore, this output signal from the comparator can be used as indicators for the matching of address and its corresponding spot in the shift register. In this manner, we get to solve the addressing issues by building the logic based on this comparator output.

The circuit can perform two(2) functions:

1. Read from the memory
2. Write into the memory

To read from memory, we first adjust switches indicating SAR1 and SAR0 to refer to the address where we want to read the data from and then turn on FETCH switch, then the data stored in the address indicated by SAR will immediately be picked from the shift registers and loaded into SBR and then displayed on the LEDs.

To write into memory, we first need to flip switches for DIN1 and DIN0 to represent the data we want to be loaded, then turn on LDSBR switch, and the value will be loaded into SBR and displayed by LED. The address is also adjusted according to the request for where the data needs to be stored. After confirming that SAR and SBR are both loaded with the desired value, we eventually turn on the STORE switch and the value stored in SBR will then be loaded into the address indicated by SAR and override the original data in the address.

Description and block diagram of the memory circuit

Without considering the logic unit, several components are required to perform the operations of this 4-to-2 words memory unit. First of all, since we are using the shift registers to represent the memory, each shift register only has a width of one bit. Two 4-bit shift registers are required to represent the word with a length of two bits. Secondly, we will need 2 corresponding 2-to-1 MUXs to select data inputs for the shift registers. Thirdly, the output of the shift registers becomes width 2 and we need two 4-to-1 MUXs to select the output of the shift registers, DINs, and reloading data from SBR (note that we only used 3 input for this MUX and the last one is not used). We will also need two flip-flops to store the value of 2-bit SBRs. In addition, we will need a 2-bit counter to count the inner cycles of the shift registers and use a comparator to output to the control logic when the counter shows that shift register is at the circle corresponding to the value of SAR. At last, we used several logic gates for the control unit.

To operate FETCH, the counter will count until equal to the value in SAR and the comparator will output one at that cycle. Through the control logic, the output of the 4-bit shift register is selected and was loaded to SBR through the 4-to-1 MUX.
To operate STORE, the counter will count until equal to the value in SAR and the comparator will output one at that cycle. Through the control logic, the output of SBR was selected by the 2-to-1 MUX and was loaded into the 4-bit shift register.
To operate LOAD, through the control logic, DIN was directly selected by the 3-to-1 MUX and was loaded into SBR.
We used 2 LEDs connecting to the SBR to check for the output. We also used 7 switches for the 7 possible inputs (used resistors to connect to the ground).

In conclusion, for the full operation of the circuit, we need: two 4-bit shift registers, two 4-to-1 MUXs, two 2-to-1 MUXs, two flip-flops, one counter, one comparator, and several simple logic gates.

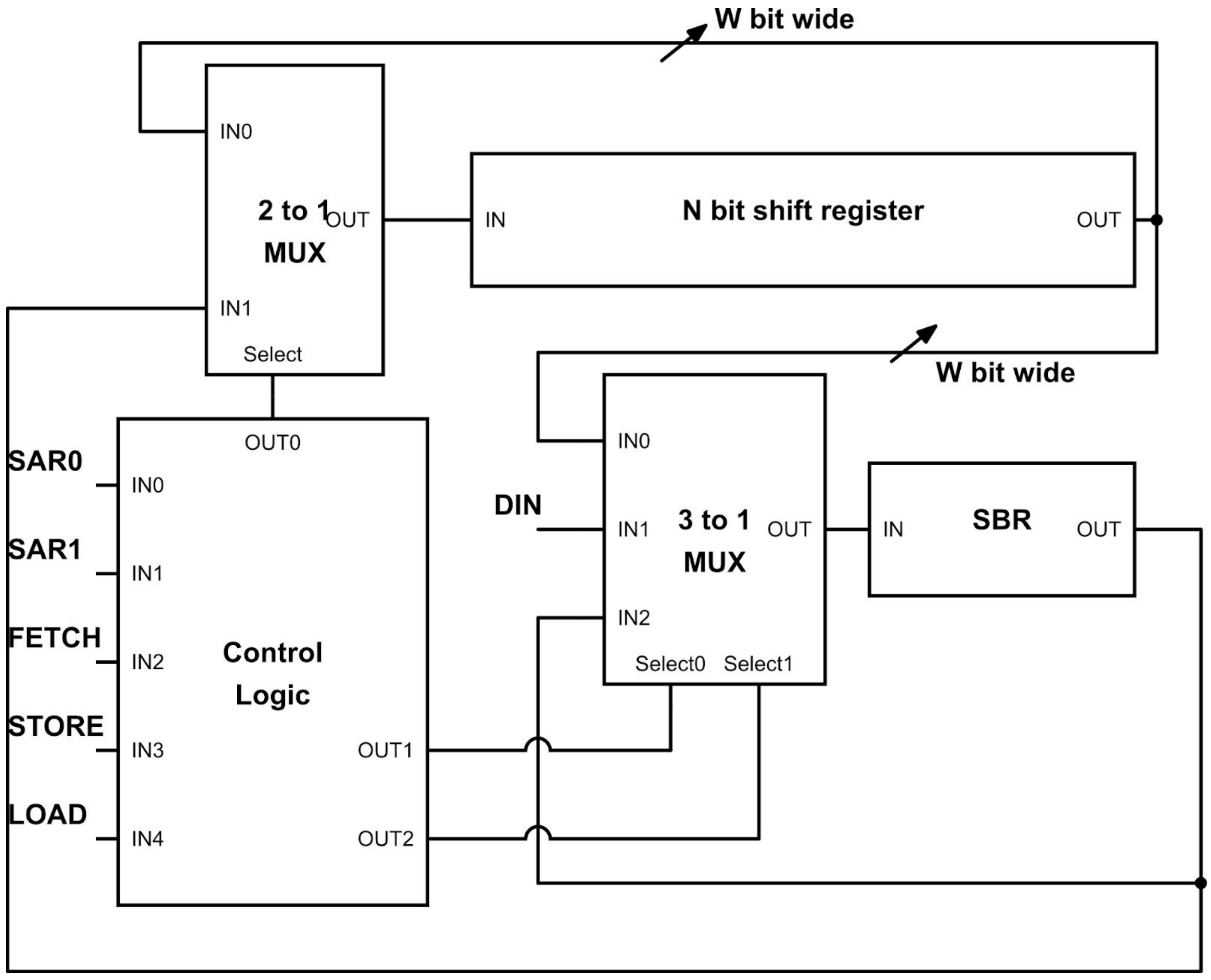

Figure 1. High-Level Block Diagram

We used chip number 74194 for this lab to represent 4-bit shift registers(N bit shift register in the diagram), chip number 74157 to represent the 2-to-1 MUX, chip number 74154 for the 4-to-1 MUX(3-to-1 MUX in the diagram), chip number 7485 for the 4-bit comparator(we only used the 2-bit functionality for this chip), chip number 7474 for the flip-flops and chip number 74169 for the 4 bit counter(we only used 2-bit functionality of this chip). *W* has a width of 2 in the diagram and *N* has a value of 4.

Control unit

The control logic is essential in the circuit, it takes in the 5 inputs and directly controls the selecting bits of the 2-to-1 and 4-to-1 MUXs. It turns out it is simple to derive the control logic for this lab once we understand the operations of STORE, FETCH and LOAD.

For operation LOAD, we have input LDSBR directly connecting to the selecting bit 0 of the 4-to-1 MUXs. Since the 11 pin for the 4-to-1 selector is not connected(see the detailed circuit diagram), the only possible value for the two selector bits when LDSBR is high is *01*. We use this value for the 4-to-1 MUXs to select the DIN1 and DIN0 and load the value into flip-flops representing 2 digit SBR. (see OUT1 in the diagram below)

For operation FETCH, we need to fetch the value in the shift register corresponding to the value in SAR(address). This is where the Counter and the Comparator took part. To fulfill the goal, we used the 2 bit counter to represent the inner cycle of the shift registers and a 2-bit-2-bit comparator targeting the inner cycle of the shift register corresponding to the SAR value. The output of the comparator is connected to the A = B pin of one of the outputs of the comparator. When SAR1 = C1 and SAR0 = C0 (see the diagram below), the out bit of the comparator will become High. When input FETCH is High at the same time (through 2 NAND gates), OUT2 which is connected to the high bit of the 4-to-1 MUX will become high. Since the 11 pin for the 4-to-1 selector is not connected(see the detailed circuit diagram), the only possible value for the two selector bits when OUT2 is high is *10*. This value is used in the 4-to-1 selector to select the output from 4-bit shift registers and loaded into the SBR through the MUX. When both LDSBR and STORE are low, the 00 pin of the 4 to1 MUX is simply the SBR value itself reloaded back.

For operation STORE, similar to FETCH, the comparator will output 1 when the inner cycle of the shift register is corresponding to the value of SAR. When input STORE appears to be high at the same time, OUT0 will be high(through the NAND gates, see the diagram below). OUT0 is used as the selecting bit of the 2-to-1 MUXs. When this selecting bit is high, the two to 1 MUXs will select the value in pin 1, which is the value of SBR and load the value into the shift register through the 2-to-1 MUX. When this bit is low, the rightmost value of the shift register is simply fed back into the left side of the shift register again.

Based on the operation above, we can see that only one AND gate AND the output of comparator and STORE, one AND gate AND the output of comparator and FETCH will be sufficient for the logic unit of this lab. We used two NAND gates to represent the one AND gate in this lab.

In conclusion, input SAR1, SAR0 was fed into the counter that contributes to the output of the comparator. Input STORE is only used to control the selecting bit of the 2-to-1 MUXs that is selecting the input into the shift register. Input LDSBR is only used as the selecting Input DIN1, DIN0 and load them into the SBR in the 4-to-1 MUXs. Input FETCH is used as selecting the output from the shift register and load it into SBR.
We did not consider the condition when STORE and FETCH are both high when designing the circuit.

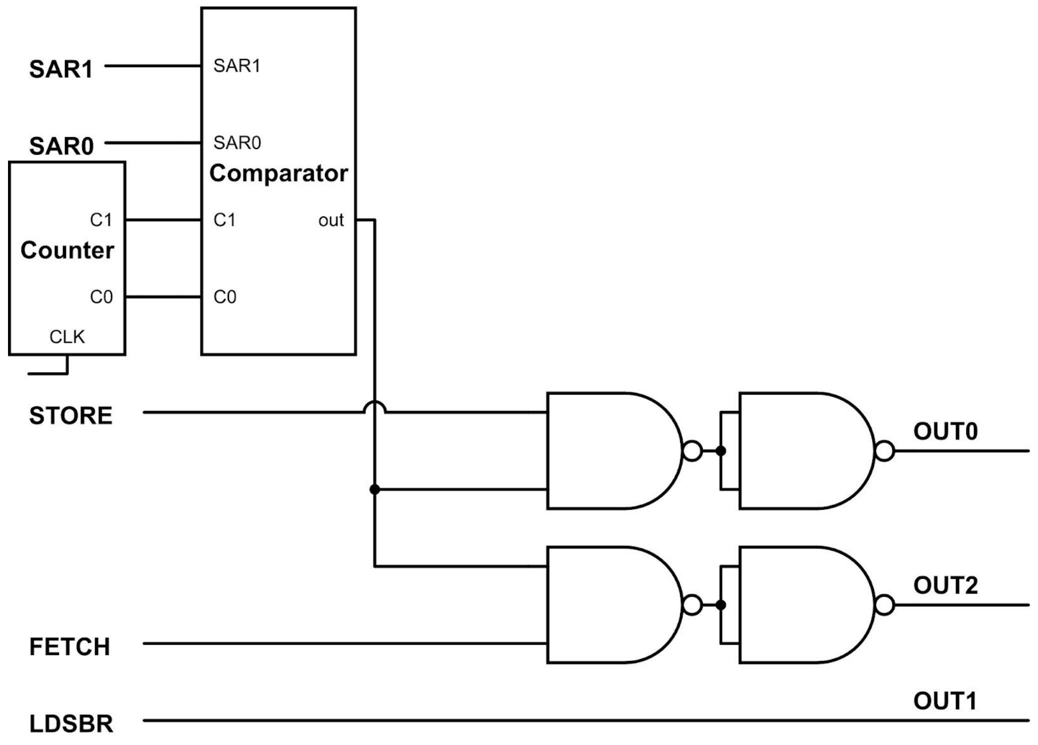

Figure 2. Control Unit Diagram

Design steps and detailed circuit schematics

We did not spend much time on the design step for this lab. At first, we did not fully understand the operation of the inputs STORE, FETCH and LDSBR. We tried directly writing out the control logic of this lab but it was wrong for the first time. Then we tried the K-Map but it indeed took a large amount of work since we have five inputs and three outputs to go. The K-Map we worked on is 4 times 8 in size and we have three to go. We figured that we have too many "don't care"s and not each input has relationships with all the outputs. We gave up the K-Map and took a more careful look at the relationships of the input and the output and developed a better understanding of the three operations. We found out that STORE is only controlling the 2-to-1 MUX and FETCH and LDSBR are only controlling the 3-to-1 MUXs. They all have a direct connection with the selecting bit except we need to take SAR value into consideration. The operations FETCH and LDSBR are only operated when the inner cycle of the shift registers is corresponding to the value in SAR and SBR. So we AND the result from the comparator and FETCH and LDSBR respectively and we are done with the logic.

The next step is to go for the circuit diagram and the wiring of the circuit. The detailed circuit diagram is easy to draw but to build the circuit, we need to get familiar with all the chips engaged. It was our first time to use most of the chips in this lab so checking the datasheet becomes the most time-consuming part of this lab. We built the circuit outside of the laboratory, so we can't test the functionality of the chips. What we could do is to follow all the information the datasheet provides. Sometimes we need to guess the functionality and to deal with the pins not used(connect to Vcc or ground, see the component layout).

We went to the lab for debugging after we had built the circuit successfully. We used the oscilloscope to test our output from different parts of the circuit but we did not encounter a lot of bugs.

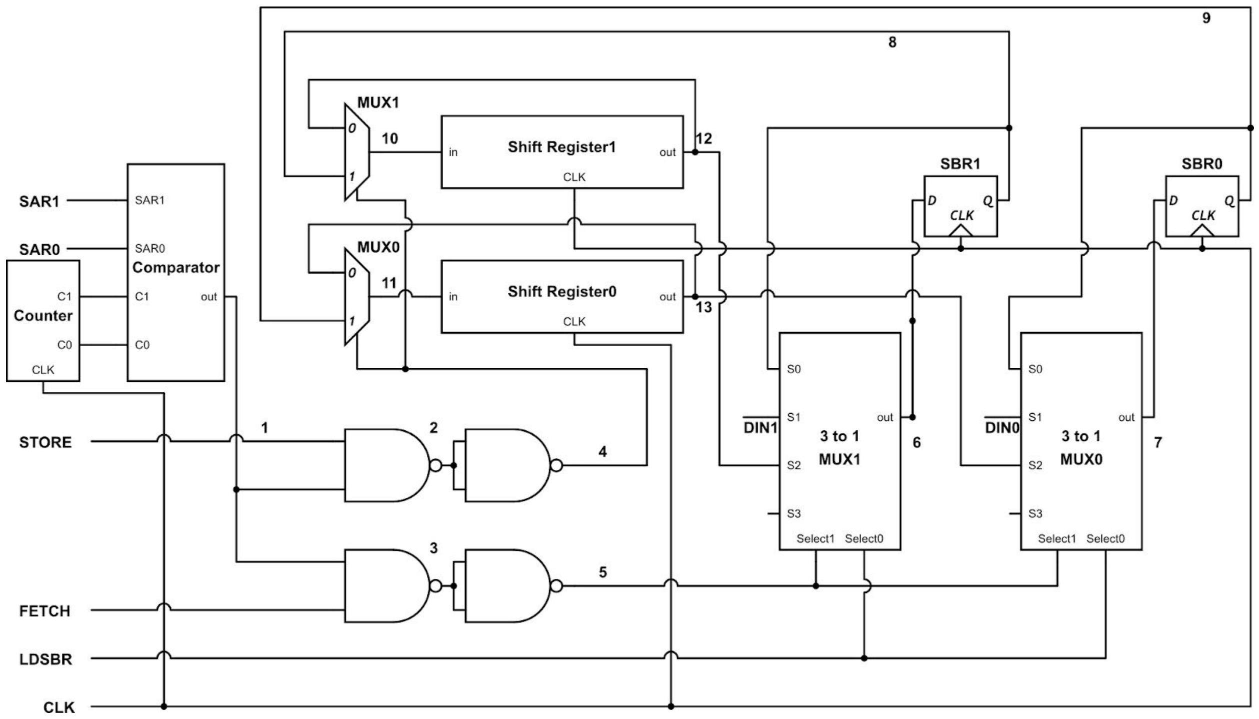

Figure 3. Circuit Diagram

This circuit diagram is corresponding to the component layout for this lab. We labeled each wire with different values with a different number, which is the same as the chip pin output number in the component layout. We labeled them out because it is easy for debugging the circuit (checking the values of every wire and find out which wire has the problem).

Component layout

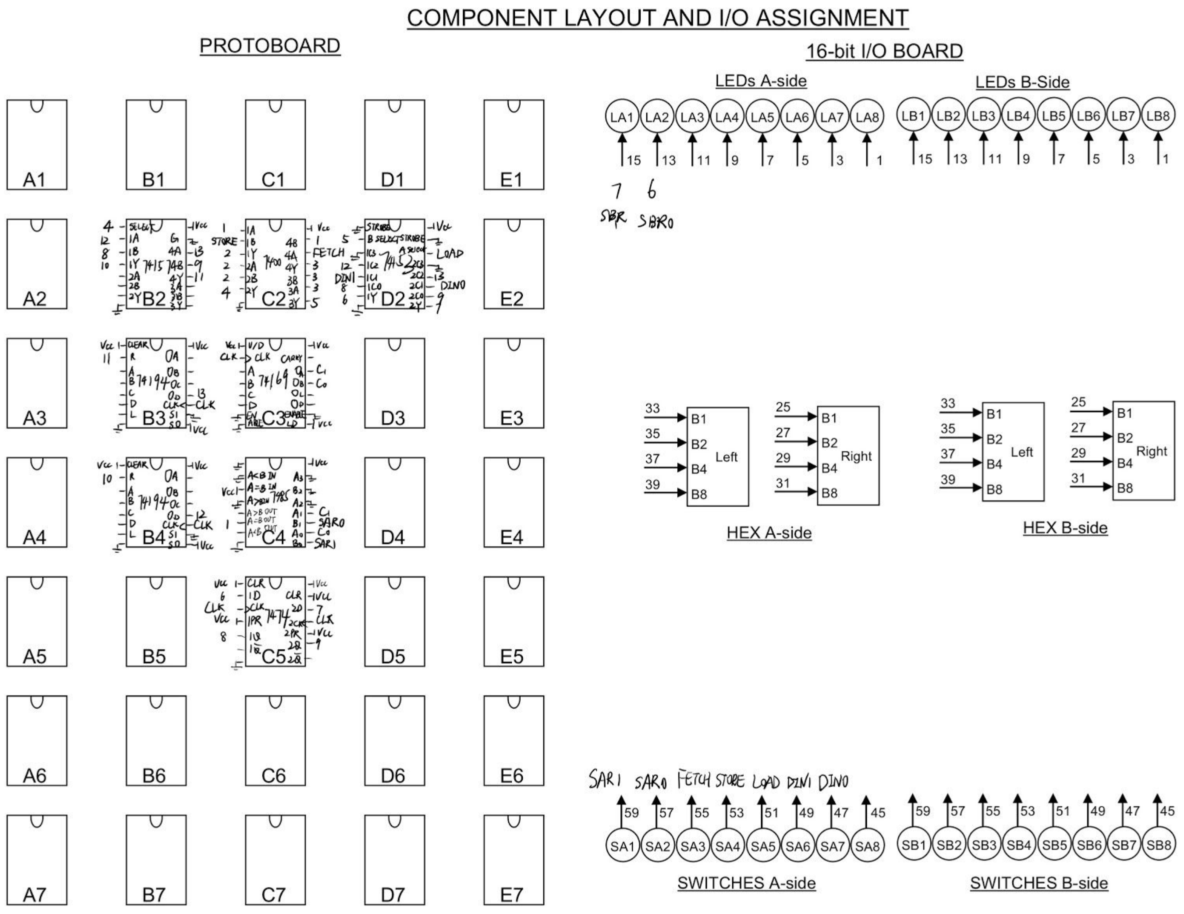

Figure 4. Component Layout

Description of bugs encountered

During the wiring of the entire circuit, it is easy to make mistakes because to implement a 2*4 memory base, we need to use two nearly identical circuits built in parallel. In this process, since all the counterparts for two subcircuits are always placed in the same area due to chips restrictions and clarity of the entire circuit, we sometimes mismatched wires for the two subcircuits. Once, we connect the output end of the shift register used to store values of the Least Significant Bit with the pin in 2-1 MUX for the Most Significant Bit. We took care of this problem using the visual examination. While one member states the designed flow of the circuit and wires, the other member checks whether the actual wiring fits the design.

Another issue is not understanding the polarity of LEDs. Since there are no clear indicators on the LED chip saying which side is the anode or cathode. We connect it in the wrong direction in the first place. And when we induce power to the circuit under that condition, no LED is on however we manipulate the switches. After figuring out that it is due to the wrong polarity of the LEDs, we find that the entire circuit works as designed perfectly and no more bugs are encountered.

Conclusion

In this lab, we designed simple control logic for the 2 * 4 memory circuit. We built and debugged the circuit to realize the functionality of FETCH, STORE, and LOAD.
We got more familiar with the digital circuits using counters, comparators, flip-flops, shift registers, and clocks. We also learned more about components in ECE385 lab kit, especially those digital chips with various usage. We learned about reading the datasheet and debugging the circuit. Most importantly, we learned about designing simple control logic of circuits.

Answers to PostLab Questions

1. *What are the performance implications of your shift register memory as compared to a standard SRAM of the same size?*
   Both FETCH and STORE need to wait for several clock cycles to complete. No absolute memory address, every address is relative.

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

   We used a synchronous counter (chip # 74169) for this lab. Unlike the ripple counter, it has synchronous design and each flip-flop inside has the same clock. Although the ripple counter is easier to design, it has some delay in practice which will create glitches. We used the synchronous counter to avoid the possibility of glitches in the output.

We used chip # 74194 for shift registers. This shift register is not simply a serial connection of four flip-flops. It has a complex inner logic that creates some advantages. When loading into the register, we can only load to the left side, but we are reading the output from the right side. This creates the problem that where we load the value may not correspond to where we read the value. For example, for the 4-bit shift register below, at a time, we load into 1, after 4 clock cycles, we want to read 1, but the rightmost of the register is 4. So it is problematic to only use four flip-flops as the shift register. We used chip #74194 instead to ensure that what we load to the left will correspond to what we read from the right as indicated by the datasheet. In the chip, since it is capable of parallel loading and has parallel outputs, we can directly read from the fourth place so that we can make sure the data input at a given address will come out again at the next appearance of the same address.

1->	2->	3->	4->