ECE 385

Fall 2019

Experiment #2

Data Storage

Xinglong Sun Churan He
ABD T 8AM
Mihir Iyer

Introduction (High level function of the circuit)

In this lab, we designed a simple 2-bit, four-word shift-register storage unit. The storage unit is able to store two bits input by user using switches into an address also defined by user using switches. User can also later fetch/read the data stored in a specific address.

A General Design Description

We first come up with what components to use based on the requirements of the functions that can be achieved by this storage unit. The user is able to both store and fetch the data at an address. Crucially, since the register will keep shifting once the clock signal is on, we need to figure out a way to both keep the original data and to write the new data. Otherwise, the stored data will just be shifted into the void. That's the reason we chose a 2-to-1 MUX in our design. The inputs of the 2-to-1 MUX are the serial output of the register (old data) and the new data to be stored. Thus, the storage unit can either keep the original data stored or write the new data. In order to get a visual display of the data stored and also to temporarily store the data we want to read, we choose to use a buffer register SBR. We use a flip-flop for the SBR. Another alternative would be using a register with a load signal. The SBR serves two purposes: loading new data from the user-input switches and reading from the shift register. We also use LEDs to get a visual display of the values inside the SBR. Similar to the shift register, since our SBR also needs to serve multiple purposes, we again choose a MUX to select between different values. One of the inputs is used for reading the register value and is connected to the serial output of the shift register. Another input is used for loading the user-input data. Also, since we need to keep the data stored, an input is used to cycle the data. We do this by implementing a 4-to-1 MUX. How we handle the fourth input will be explained in the Design Consideration section. The select signals of both MUXes are determined by our control logic. We use counter, comparator, and a few gates in the control unit. The choice for these components has to do with a particular addressing we implement and will also be elaborated in the next sections. Generally, the operations our circuit performed can be classified into four classes : WRITE/STORE, READ/FETCH, LOAD, DO NOTHING.

Operation of the Memory Circuit

- How the addressing is implemented

To build a 4 * 2 bits storage unit, we need to create 4 addressable spaces with our circuit. The actual storage chip we use is SN74LS194, a shift register, and we used two of it for 2 bits. Without using the parallel load and output capability of the chip, we only let data in the register circularly shift right. There are no absolute addresses in a serial-shift register, because every bit is changing its location at every clock cycle. So using the relative address (location) is a more practical way to keep tracking every bit we store in the memory unit.

In this case, we want to use a 4 cycle up-counter to achieve the relative address design for the 4-words shift registers. To specify in bits, the bit patterns of the counter outputs will be 00, 01, 10, 11, and back to 00 to start a new counter cycle. This specific counter pattern matches the working cycle of the shift register.

- How “write” is performed

The write operation contains two steps. The first step is to store the data of the switches to the SBR (implemented by flip-flops), and the second step is to store the data from SBR to the actual shift registers at a specific relative address.

In order to store the switch data to the SBR, the LDSBR switch needs to be turned on. The 4-to-1 MUX connects the data from switches to the SBR. On the next clock signal, the SBR would store the data from the switches. Since the 2-to-1 MUX still connects the serial output of the shift register with the serial input, the data inside the shift registers would still be the same. Afterwards, we need to input the address with the SAR switch, and then turn on the STORE switch. The 4-to-1 MUX at this time would keep the current SBR data. The input SAR address will be fed into the comparator, and be compared with the counter’s value (current relative address). In less than 4 clock cycles, when the input SAR address matches the current shift register address, the 2-to-1 MUX will connect the SBR with the right shift serial input of the shift registers. At the next clock cycle, the data would be securely stored.

- How “read” is performed

In order to read data from the memory, the address should be specified, and the FETCH switch needs to be turned on. The comparator would compare the input address with the counter’s value (current relative address). In less than 4 clock cycles, when two addresses match, SBR would be loaded with the data at the specified location of the shift registers at the next clock cycle.

Written Description & Block Diagram of Memory Circuit

- High-level description

The memory module contains switches, flip-flops (SN74LS74), MUXes (SN74LS153, SN74LS157), and shift registers (SN74LS194).

The switches are responsible for selecting the data process mode (STORE, FETCH, and LDSBR) and specifying the SAR address and the data inputs.

The flip-flop acts as storage buffer register, and it stores the data input from the switches or the data read from the shift registers.

Counter and comparator are used here to determine when the data should be fetched out of and stored into the memory unit.

MUXes are controlled by the control unit, which consists of the counter and comparator, to select data path.

The shift registers are the places that store the actual data.

- Block Diagram

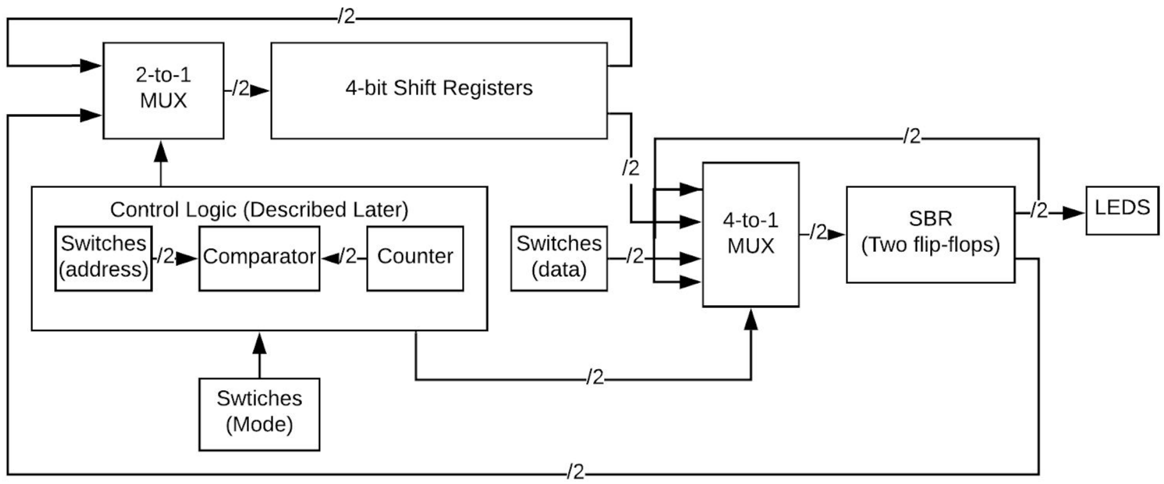

Figure 1 Block Diagram of Memory Circuit

Control Unit

- Written Description

The control unit takes the signals STORE, FETCH, LDSBR, SAR0, SAR1 as its inputs and outputs the select signals for the 2-to-1 MUX and the 4-to-1 MUX. Basically, the control unit decides what data should be stored into the memory address and our buffer register based on the user-defined actions.

Because of the addressing method we implemented, we need to use a counter and comparator as the main components of our control logic. In our design, we use SN74LS169 chip for the counter and SN74LS85 chip for the comparator. Since all the addresses are relative, the counter keeps track of the address associated with the data to be shifted out from the serial output or into the serial input of the shift register at the up-coming clock edge. For the address to match the user-input address, the comparator is used here to compare the counter value and SAR (two-bit address) switch inputs. When the STORE signal is on AND the address is matched, the new value in the SBR is going to be chosen for the serial input value of the register and get stored in the next clock cycle. When the FETCH signal is on AND the address is matched, the value in the specific address is going to be chosen for the SBR and be shown on the LED in the next clock cycle. When the LDSBR signal is on, the user input data will be stored into the buffer register SBR waiting for further actions. When no signal is on, the data will just keep being cycled.

- Block Diagram
  A high-level block diagram of our control logic is shown as follows:

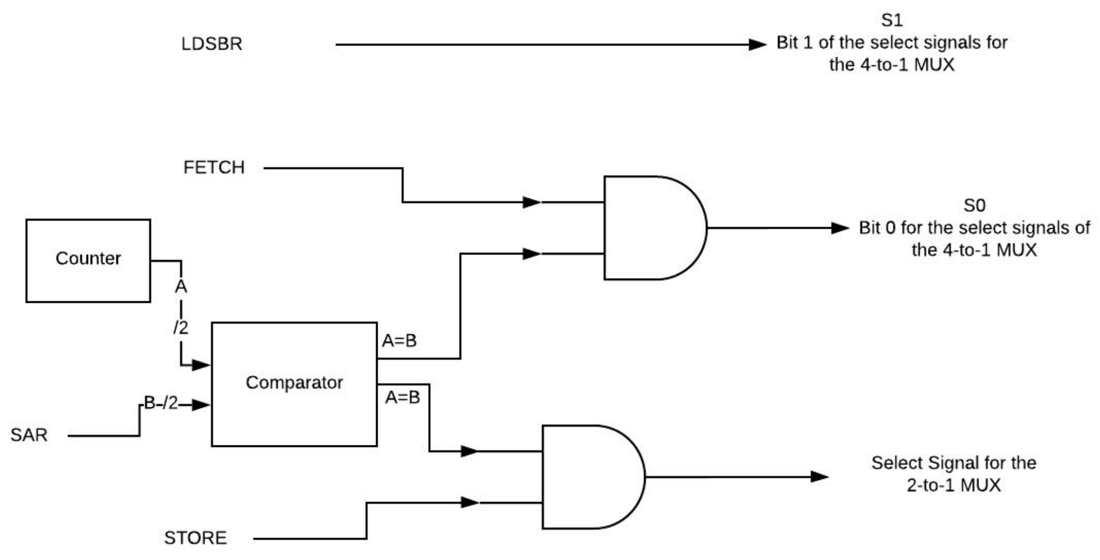
LDSBR
S1
Bit 1 of the select signals for the 4-to-1 MUX
FETCH
S0
Bit 0 for the select signals of the 4-to-1 MUX
Counter
A
/2
Comparator
A=B
A=B
SAR
B /2
Select Signal for the 2-to-1 MUX
STORE

Figure 2 Block Diagram of Control Unit

Design Steps & Considerations

- Steps
  Since the control logic in the circuit is very easy, we didn't derive any boolean algebra expressions from truth tables and K-maps. The reason to AND the comparator output and the STORE/FETCH signal is explained in the above section.
- Considerations & Notes
  There are several things that are worth being noted here in order to avert similar problems in the future.

1. Clock Gating (Pre-Lab question)
   It's a bad practice to gate the clock in digital design. The enable signal for the clock-gating has to be retained for the entire clock cycle and should not be affected by glitches caused by the propagation delay of our gate. This needs extra care in practice, and the glitches are not guaranteed to be suppressed. In a synchronous design like we did for this lab, all components share a common clock signal, and clock-gating increases the risk of asynchronous operations.
2. Debounced Switches (Pre-Lab question)

When we build the circuit at home and do not have the function generator for creating pulses, we use switch for testing and debugging. Particularly, only the clock input needs to be de-bounced. The concrete reason is as follows. A bounce away from the 5V is considered as a falling edge of the clock signal, and a bounce away from 0V is considered as a rising edge of the clock signal. Therefore, if the switch bounces, your circuit could possibly go through multiple clock cycles and does not work in the way as expected.

3. Usage of LEDs

Using LEDs to display some signal values is very important for testing and debugging the circuit. When we built the circuit, we choose to display the SBR values ,Counter values, and Comparator Output values on LEDs. The reason we choose to display the counter and comparator values on the LED is that we can keep track of the current address and verify that the STORE/FETCH do indeed get access to data of the specific address that user defined. This is very helpful to test our control unit.

Detailed Circuit Schematic

- Memory

The actual memory circuit diagram is shown as follows, and the interconnections between the components are based on the block logic diagram shown in the **Memory Circuit** section.

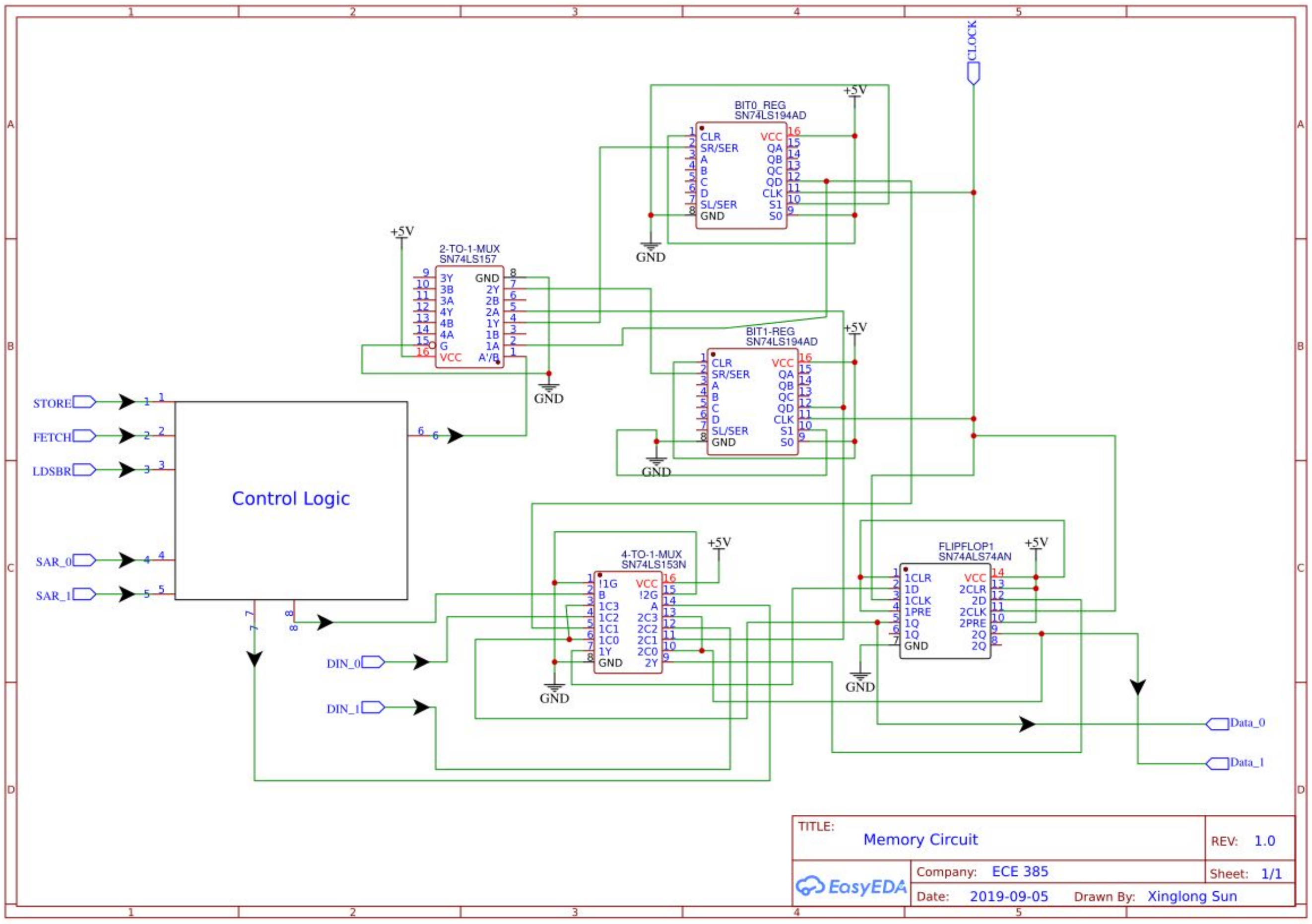

Figure 3 Memory Circuit Schematic

The arrows show the input and output of the datapath. Data_0 and Data_1 are displayed on the LEDs. **All 5V labels are connected, and all IC chips share the same common ground**. The details of the circuit within the control logic box are shown in the next subsection.

- Control

The actual control circuit diagram is shown as follows, and the interconnections between the components are based on the block logic diagram shown in the **Control Unit** section.

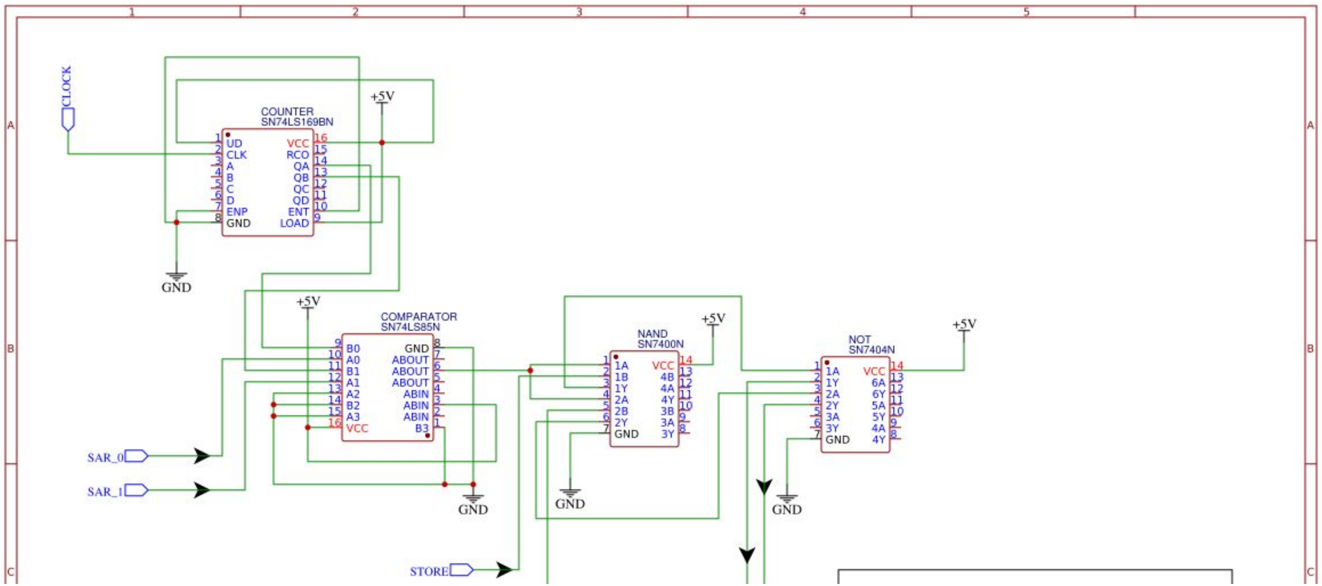

Figure 4 Control Circuit Schematic

All of the chips share the **same 5v and ground**. Also the clock signal in the control circuit is the same as that of the memory circuit.

Notice that the numbers(6,7,8) of the three outputs to the memory circuit in the above image match the numbers of the three outputs to the Control Logic box in the part a. Memory circuit diagram.

Component Layout Sheet

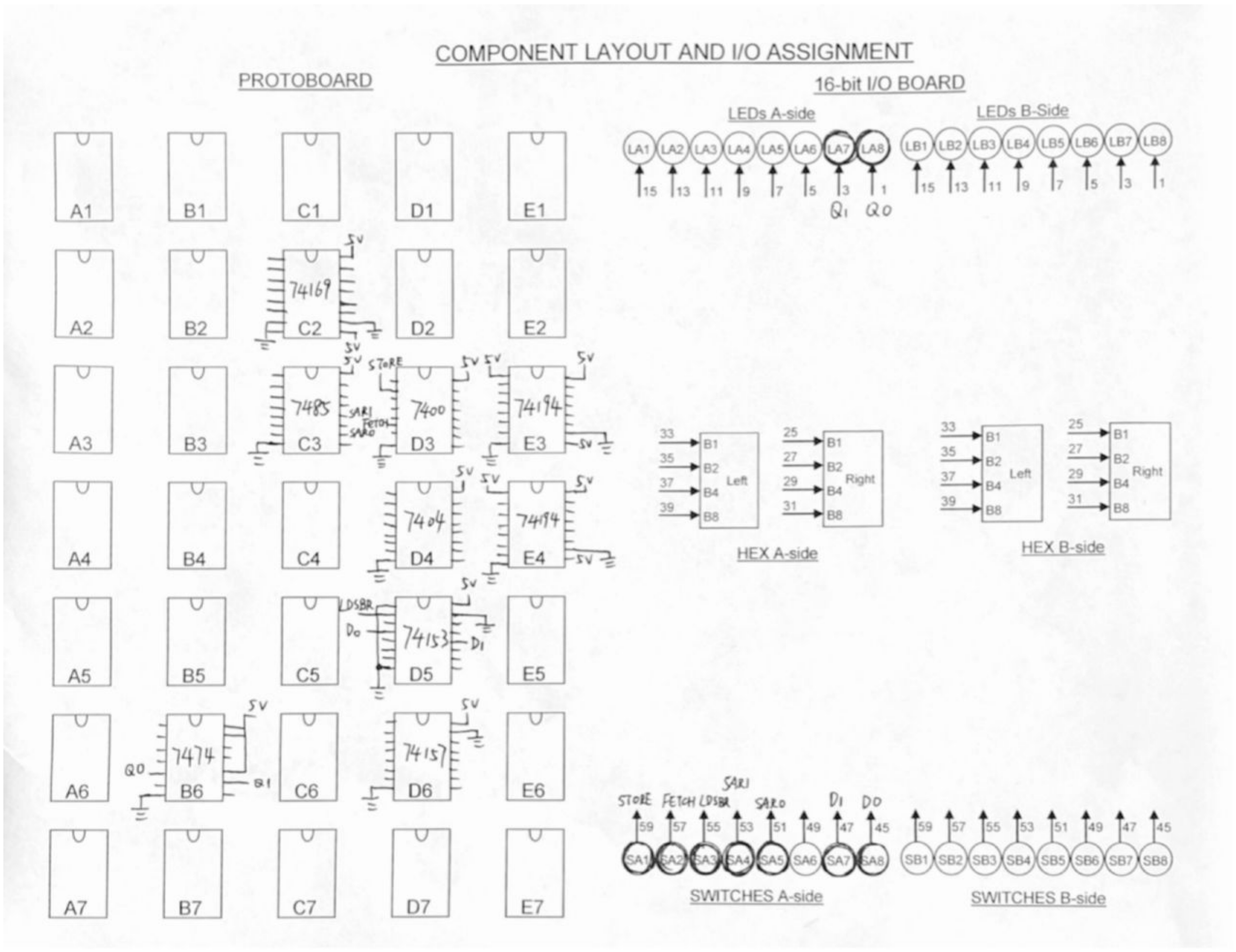

Figure 5 Component Layout Sheet

Bugs

- Floating STROBE pins on MUXes

  Initially, we forgot to connect STROBE pins of the MUXes to ground, so the MUXes could not select the signal correctly. After confirming each input signal and selecting signal are correct, we rechecked the data sheet of the MUXes and found the problem of unconnected floating STROBE pins. Then we grounded all the STROBE pins and the circuit worked as our design.

- Bad wires and connections

  Our breadboard and wires had been used by previous groups, and not all of them could work in a normal condition. We found a few holes of our breadboard were interconnected, and some wires were not conducted. To fix these problems, we rescheduled the layout of the circuit, and replaced problematic wires with new ones.

- Bounced switch as the clock

When we tested our counter for the first time, we didn't realize the importance of the debounced switch. Thus, we noticed every time we turned the switch on and off, the counter would count for 2 to 3 times. Initially, we thought we used the counter in a wrong way, but with more understanding of how the counter depends on the clock, we replaced our bounced switch with a square wave generator, which gave us a satisfiable result. We also later built a debounced switch by ourselves for debugging and testing at home.

- Interference with bad oscilloscope

As we tested the control unit, we encountered a problem that the counter only counted in a cycle of 3 when we connected the oscilloscope with the output of the comparator. However, if we disconnected the scope with comparator, the counter would count normally. As we explored further, we found connecting the oscilloscope with the output of the comparator would affect its input, and in turn affected the counter. In the end, we learnt that bad measure instruments would not only provide error results, but also may affect other components in the overall circuit.

Conclusion

- Summary

In this lab, we build a 4 * 2 bit storage unit based on two four-word shift registers. In order to assign address to each register location, we use a counter to count in a cycle of 4, which accommodates the four spaces in the shift register. Then we also use a comparator to read/write data at specified location. The comparator would control the select signal with switches' inputs with the control unit's logic, so the data could flow in our designed sequence and being written or read correctly. Our circuit eventually works normally; it can write data from the switches input to a specific memory address and read data from the specified address. We learn many useful digital design techniques during this lab, such as modular design which significantly reduce our debugging time. We also get more familiar with the IC chips including but not limited to registers, counters, flip-flops, MUXes, etc.

- Answers to post-lab questions

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

The performance of our shift register is slower than a standard SRAM of the same size. Because the shift register memory may take at most 4 clock cycles for one data write or load operation (waiting for the shift register shifting the correct address), SRAM has an absolute address for each 2 bits data and wouldn't need to wait for the location.

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

The counter determines the relative address of the shift register, so the counter's cycle needs to be the same as register's space. If counter's cycle is more than register's space, the data at a specific location of the register may be overwritten. If register's space is more than the counter's value, the extra space cannot be assigned with its specific address. Different size of counter’s cycle and register’s space would also cause asynchronous pairing of the counter’s value and shift register’s data location after the first cycle. In our circuit, we used 4 bits shift register and a counter with 4 cycles. Matched counter’s cycle and register’s space can guarantee the circuit works normally.