# ECE 385

Spring 2024

Experiment # 2

# Data Storage

**Name:** Jie Wang, Shitian Yang

Student ID: 3200112404

Jan. 26, 2024, Friday D-225

TA: Jiebang Xia

I. Introduction

In digital systems, how to store and manipulate data efficiently is a fundamental problem . In this experiment, we design and construct a simple 2-bit, four-word shift-register storage unit on the Quartus Prime + ModelSim Simulation. As shown in the logic diagram below, we need to implement 4 components:

1. Address controller on the control logic of deciding ‘read/write’ data with counter and comparator.
2. 3-to-1 MUX to decide the operating mode from ‘store’, ‘fetch’ and ‘load’ states.
3. 8-bit shift register functioning as the core storage element, which is configured using two 74LS194 data selectors to accommodate the given data.
4. Storage Buffer Register

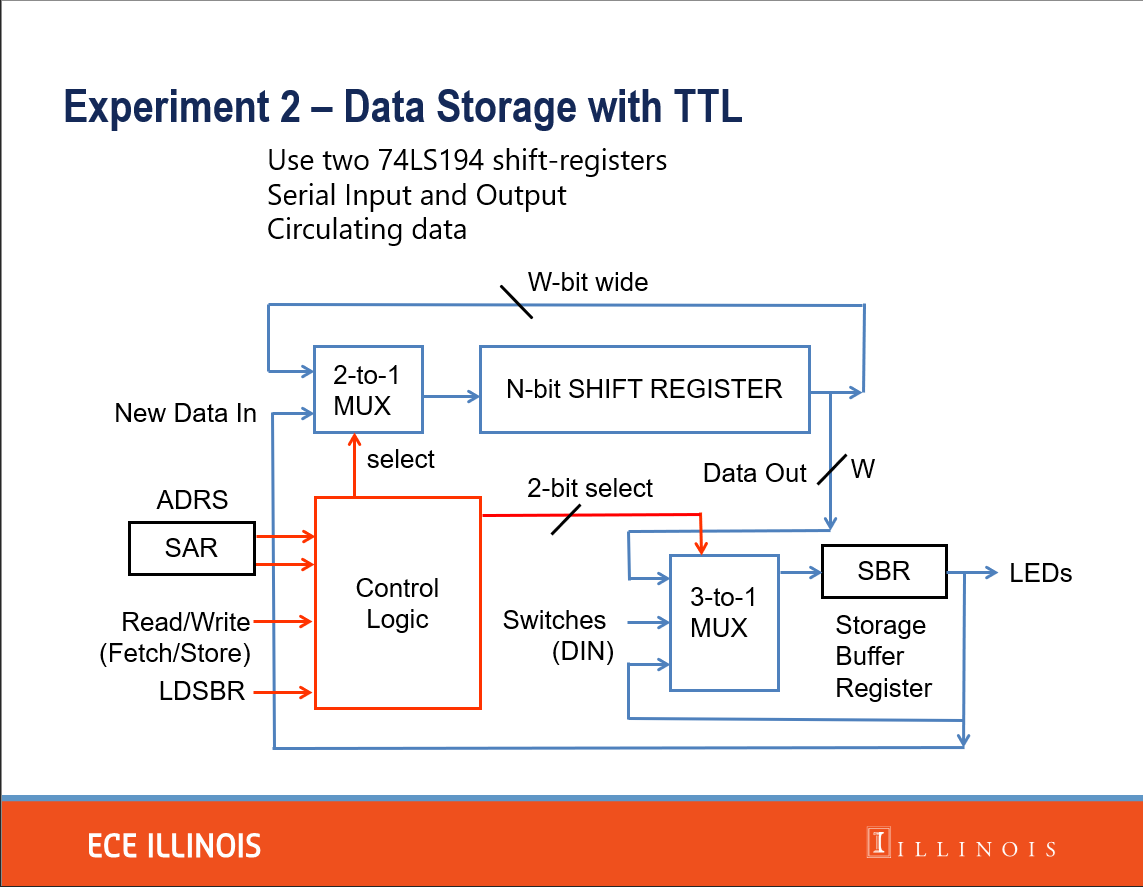


Figure 1: Logic Diagram for the who Data Storage System

II. Data storage mechanism

To store binary data in the circuit, we typically utilize the coordinated operation of an 8-bit shift register and associated multiplexers. The shift register serves as the primary repository, holding and sequentially moving data. Multiplexers, governed by the control logic, manage data flow by selecting the appropriate input based on the current mode—'store', 'fetch', or 'load'. An address controller, leveraging a counter and a comparator, directs the system's read or write actions, ensuring data is accurately placed or retrieved from the shift register. The Storage Buffer Register provides a temporary hold for data during transfer operations, ensuring consistency and stability in the system's performance. This architecture enables reliable storage and retrieval of 2-bit binary data across four distinct words, demonstrating the fundamental principles of digital data management.

What’s more, the reason why we shouldn’t gate the clock is because it may result in Clock Skew. 时钟使能（Clock Enable） 是一种更好的实践，它允许时钟信号保持连续不断，但是在电路的某些部分可以通过一个使能信号来决定是否接受这个时钟周期。时钟使能通常是一个控制引脚，在该引脚激活时，电路响应时钟信号，在未激活时，电路保持当前状态不变，即便时钟信号仍在变化。

不应该对时钟信号进行门控制，就像我们不应该随意地改变交响乐团中指挥的节拍。如果指挥随意改变节拍，乐团的演奏就会混乱，各个乐器之间的协同就会出现问题，最终影响整场表演的和谐。在数字电路中，时钟信号就像是乐团的指挥，它确保所有的数据转移和处理都能在正确的时间以有序的方式进行。

Methodology

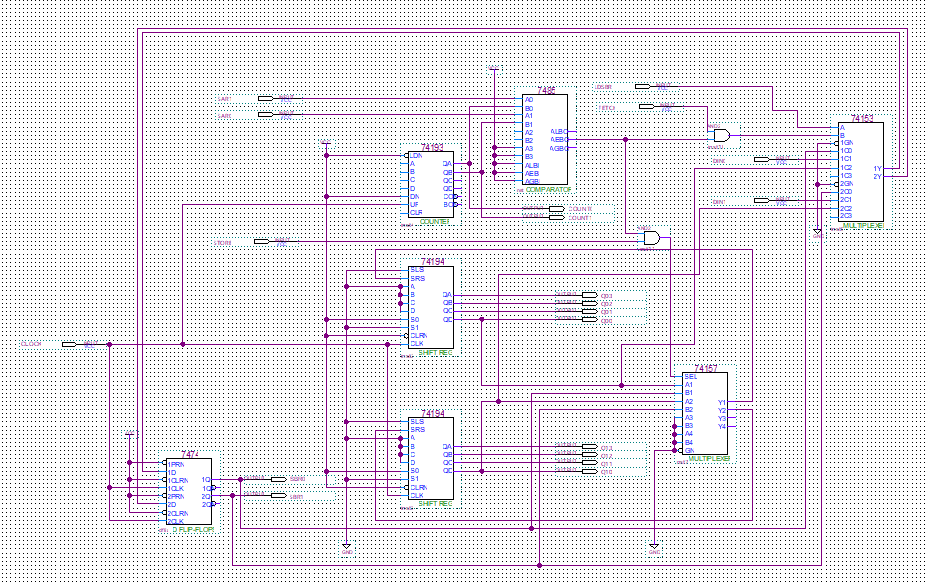
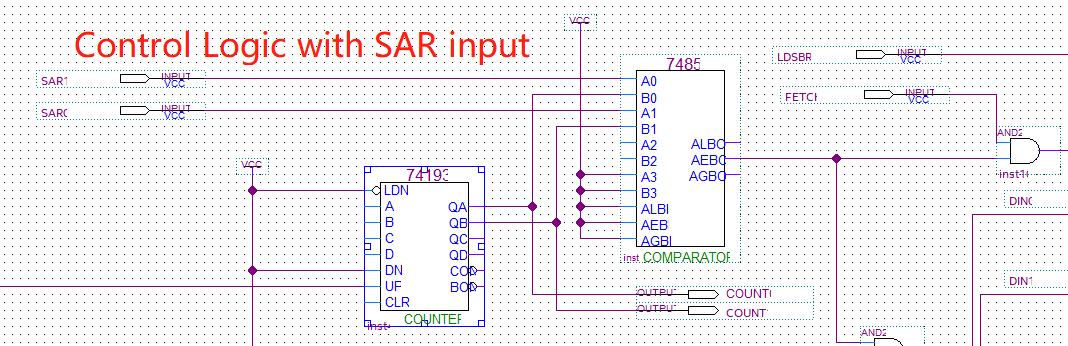


Figure 3: Overview of Our Data Storage Circuit

* 1. Summarize what high-level function your circuit performs. How many words does your memory contain and what is the bit width of each word? The introduction should be approximately 3 ­ 5 sentences.

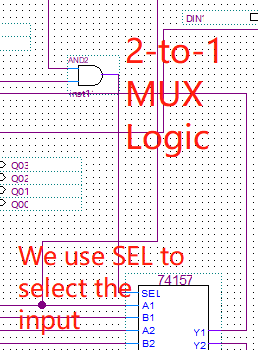


图示, 示意图

描述已自动生成 图片包含 日程表

描述已自动生成

图片包含 图示

描述已自动生成 

STORAGE

address contents

SAR

word 0

word 1

SBR word 2

word 3

word 4

FETCH word 5

word 6

STORE word 7

Figure 1: An Eight-word Storage Unit Using 4-Bit Words

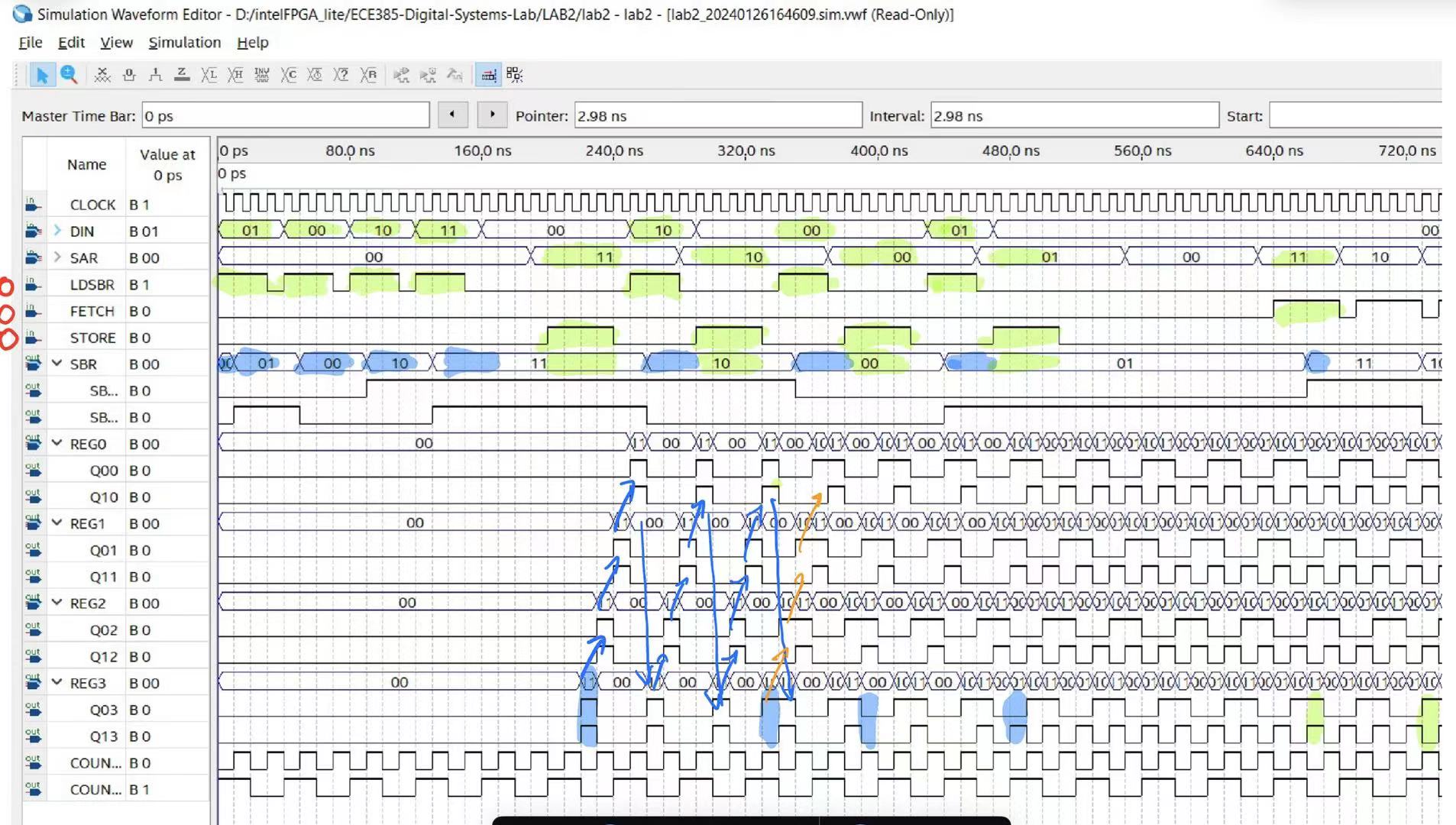
Our Waveform Result:

图形用户界面, 应用程序, 表格, Excel

描述已自动生成

BUG:

1. 元器件接错了，VCC 短路到GND，时钟信号无输入不工作
2. 元件选择错误，datasheet理解错误，导致counter输入有问题,更换counter型号之后正常工作
3. Fetch 和store 模式选择与元器件链接时，MUX没有正确的选择

To fetch a word from storage, the unique word address is placed in the Storage Address Register (SAR) and a FETCH signal is sent. The binary string or a content of the specified word appears in the Storage Buffer Register (SBR) a short time later (exactly how much later depends upon the technology used for the storage).

To store a word into storage the unique word address is placed in the SAR, the binary data to be stored is placed in the SBR, and a STORE signal is sent. The binary data in the SBR is stored in the word whose address is specified in the SAR. The previous contents of the word are destroyed by the store operation.

Cathode ray tubes, delay lines, and magnetic cores were once used for storage. In the 1970s, this was replaced by semiconductor RAMs, which are common now. You should be able to construct a storage unit from parallel-in/parallel-out shift registers, multiplexers, counters, and combinational logic.

One storage technique uses serial-in/serial-out (SISO) shift-registers shifting synchronously. A single 1024-bit SISO shift register could be used to provide 1024 words of storage, where each word is a single bit (e.g. a 1024x1 RAM). Note that with a 1024-bit SISO shift register, only the output of the rightmost flip-flop and only the input to the leftmost flip-flop are available. In theory, a shift register can be built at much lower cost compared to a RAM. This is because there are far fewer pins and interconnections in shift register than in RAM. In addition, the storage cell of a shift register could be very simple, a capacitor for example. The shift operation is simply moving charge from one capacitor to the neighboring capacitor. Such shift registers are called Charge Coupled Devices (CCDs). Today, CCDs are primarily used in imaging applications, such as in digital cameras. The charge in a cell slowly decays and therefore must be refreshed before it is lost. For this reason a SISO memory based on CCDs must be continuously shifted to keep the information from being lost.

Words larger than a single bit can be constructed by using more 1024-bit shift-registers clocked synchronously. Typically, 16 such SISO shift registers would be used to construct 1024 words of storage, where each word is 16 bits long. More generally, an n‑bit, m-word shift-register storage consists of n m-bit shift-registers shifting together (see Figure 2).



Figure 2: Configuration of a Shift-register Storage

As mentioned earlier, an alternative to the above storage devices are those devices that are built with "static" logic elements (SRAM). This is a setup where the storage device can retain data so long as a specified supply voltage is maintained. These SRAM chips are readily available from a number of manufacturers with varying features and parameters.

III. PRE-LAB

A. Design, document, and build a 2-bit four-word shift-register storage unit using two 74LS194 shift-registers without using the parallel load or parallel output capability. Of the 74LS194 data (non-control) inputs and outputs, you may use only the serial input and the rightmost (Qd) output. For the purposes of this experiment, imagine that the maximum and minimum clock period is specified as 1 millisecond for the 74LS194. The registers must be shifted on each clock pulse. The clock must run continuously – do **not** gate the clock (this is bad practice in digital design, why?) You **may** use clock enable or inhibit pins on the chips.

Signal Definitions:

LDSBR When LDSBR is high, the SBR is loaded with the data word DIN1, DIN0.

FETCH When FETCH is high, the value in the data word specified by the SAR is read into the SBR.

STORE When STORE is high, the value in the SBR is stored into the word specified by the SAR.

SBR1, SBR0 The data word in the SBR; either the most recently fetched data word or a data word loaded from switches (note that when none of the LDSBR/FETCH/STORE switches is set, SBR should maintain the data in it)

SAR1, SAR0 The address, in the SAR, of a word in the storage

DIN1, DIN0 Data word to be loaded into SBR for storing into storage

Use flip-flops for the SBR. DIN1, DIN0, SAR1, and SAR0 should be obtained from INPUT ports in schematic. FETCH, STORE and LDSBR should also be obtained from INPUT ports in schematic. Display SBR1, SBR0, DATA Qx0-3 in memory on OUTPUT. You may also wish to include the display of other signals in your design for convenience when debugging your circuit. You can assume that only one of the FETCH/STORE/LDSBR switches will be set at any given time. Do not combine the FETCH/STORE switches!

To design the shift-register storage unit, we first need to look at the required specs. The most crucial requirement is for the shift registers to shift continuously, while using the serial input and output to store and fetch the data. We can break down our circuit operation into four operations: *load*, *read*, *write*, and *do nothing*. Let’s first imagine the scenario where the circuit is turned on, but we are neither loading, reading nor writing. This is the most common state of the circuit, where no action is taken from the user – *do nothing*. Our requirements dictate that the shift registers must continuously shift, where any potentially stored data will be shifted out of the registers and into the void. To prevent losing any data, we will need to redirect the data shifting out of the registers back by connecting the serial output of the registers to their serial input, where the stored data will now be looping continuously in the shift registers. However, during a *write* operation, we do want to replace the old data with new data. To serve both purposes, a 2-to-1 multiplexer (MUX) can be placed at the serial input of the shift registers, taking either the new data or the old data depending on the current operation.

The rest of the circuit operation hinges upon the SBR, which serves two purposes: loading new data from DIN during a *load* operation and reading from the shift register during a *read* operation. Note that the SBR must also behave as a register (that is, be able to synchronously maintain its previous contents). There are two ways to approach this. In the first, a simple D-flip flop may be used to implement the SBR. The input of the SBR takes in these three different choices by using a 3-to-1 MUX (or a 4-to-1 MUX with one input ignored), where one of the inputs is used to loop back when the SBR needs to maintain previous data. Alternatively, you may use a register chip which has a load enable to implement the SBR. This allows you to use a 2-to-1 MUX here instead.

But what is the ‘current operation’ at any given moment? Surely the desired operation is dictated by the user using the switches, but the inputs alone is not sufficient to tell each part of the circuit what to do. For example, if you would like to read from a specific address in the shift registers, you would first set the SAR to the specific address then you would hit the FETCH switch. But since the shift registers are constantly shifting data in and out of their serial ports, when exactly do you load the data into the SBR? How do you exactly tell what input the MUX should choose from? To solve the various problems associated with controlling and timing, it is generally not a good idea to use the inputs to directly control the various circuit components. Rather, it is almost always desired to have a centralized control logic that takes in all the inputs, process the request, and sends out various signals to control the circuit components. Figure 3 shows a general block diagram for the proposed circuit design. The most common form of a control logic is a state machine, which we will discuss in the next experiment. In this experiment, we will improvise a simpler control logic based on the requirements of our specific circuit.

First, notice that our shift registers are four word long, that is, each data will take exactly four shifts/clock cycles to loop back to its original location. We can exploit this property by employing a 2-bit counter (four distinct values) to keep track of the internal data address, then use a comparator to match the internal address with the SAR. Note that since the register is always shifting, it is meaningless to indicate "absolute" storage addresses. Rather, all addresses are "relative." If you wish to store data X in address Y, you can write the data into a random cell Z when the internal data address from the 2-bit counter matches the SAR. This (previously arbitrary) cell Z will now be associated with the address Y. Later, when we wish to fetch from address Y, we wait for the internal counter to match the SAR again - that is when cell Z once again becomes available for reading or writing. Another interpretation that might be useful is that the counter always keeps track of the address associated with data to be shifted out from serial output/into serial input of the shift register array at the **up-coming clock edge**. Note that to control the MUXs, the ‘select’ signals generated by the control logic must take into account of the input switches and the comparator output (to indicate if we are currently looking at the correct address for reading/writing).

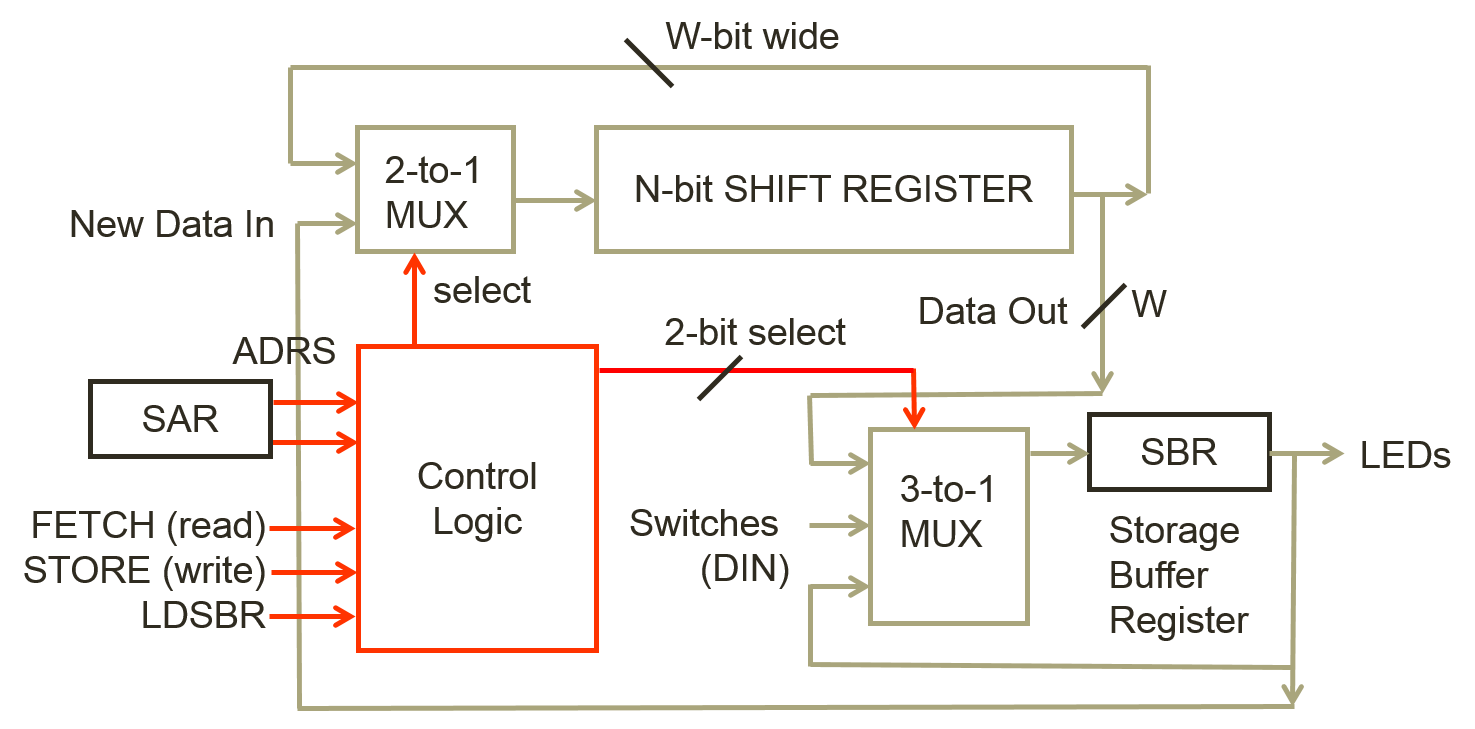


Figure 3: Block diagram of the shift-register storage unit.

**Your pre-lab writeup should contain a written description of your circuit operation, a block diagram, operation of the controller, a logic diagram.** Note that these materials only need to be turned in as part of your lab report, but you should keep in mind to generate them during the design process (rather than later).

HINT: Use the Pulse Generator to provide a basic clock. Continuously clock the shift-register and a counter that keeps track of which word is currently available. Use combinational circuitry (or 74LS85) to check for a match between the available word and the SAR.

B. Meet with your lab partner and wire up and test your design before coming to the lab. Use either the mini-switchbox circuit that you built at the end of Lab 1 (detailed in the General Guide) or attend an open lab session to test your circuit with the real switchbox. Only the clock input needs to be de-bounced to strep through your circuit (why?).

**Demo Points Breakdown: (We will evaluate your simulation results generated by the standard input. The standard input is defined by Waveform2.vwf)**

1.0 point: When LDSBR is high, the data in DIN is loaded from the switches into SBR

1.0 point: When STORE is high, the contents of the SBR are stored into the location specified in SAR

3.0 points: When FETCH is high, the data word specified by the SAR is read into the SBR

Note: you may get 1 point of partial credit from these 3 for demonstrating that your shift register can shift right on each pulse of the clock. If you get the entire assignment working, you do not need to demonstrate this independently (since you may need to rewire a shift register to demo this).

V. POST-LAB

(1) Problem we met while developing the data storage.

1) Your post-lab writeup (notes) should contain a corrected version of your pre-lab writeup and an explanation of any remaining problems in the operation of the circuits. This will aid the writing of your lab report.

2) Discuss with your lab partner and answer at least the following questions in your lab report:

* What are the performance implications of your shift register memory as compared to a standard SRAM of the same size?
* What are the implications of the different counters and shift register chips, what was your reasoning in choosing the parts you did?

VI. REPORT

1. Introduction
   1. Summarize what high-level function your circuit performs. How many words does your memory contain and what is the bit width of each word? The introduction should be approximately 3 ­ 5 sentences.
2. Operation of the memory circuit
   1. Describe how the addressing is implemented. When does the circuit commit a read or write from/to the input switches and output register respectively?
   2. Describe how a write operation is performed on the memory. Describe what switches you flip and in what order. Describe intuitively how the data flows through the circuit at each clock cycle.
   3. Describe how a read operation is performed on the memory. Describe what switches you flip and in what order. Describe intuitively how the data flows through the circuit at each clock cycle.
3. Written description and block diagram of memory circuit implementation
   1. High-level description
      1. Describe in words what components are necessary to perform the operations described in the written description of the memory circuit operation.
   2. Include a high-level block diagram like figure 3 in the lab manual. This diagram should contain components on the granularity of registers, muxes, and blocks and include few/no individual gates.
4. Control Unit
   1. Provide an intuitive written description of your control unit.
   2. Include a block diagram of your control unit. It is acceptable to turn in either:
      1. A single high-level block diagram, which contains the sub­components inside the control unit.
      2. Two block diagrams, one like figure 3 in the lab manual and one containing only the inside of the control unit.
5. Design steps taken and detailed circuit schematic
   1. If you used k­maps or truth tables during design, include them here. (If you didn’t need them, you don’t need to include them).
   2. You do not need a state diagram for this lab.
   3. Written description of the design considerations taken (did you consider multiple implementations of the same circuit and the tradeoffs of each?)
   4. Detailed Circuit Schematic
      1. This diagram should show all components used and their interconnections down to the logic gate level. Large schematics can be broken down into a hierarchy (for example, in the main diagram, the control logic can be shown as a box with inputs and outputs and a separate detailed diagram of the control logic can be placed below the main diagram). You may omit gate level representations of complex chips which you used (counters, etc.) and instead replace them with a block and all connections.
6. Description of all bugs encountered, and corrective measures taken
7. Conclusion
   1. Summarize the lab in a few sentences
   2. Answers to all the post-lab questions, these may either be in a separate section or dispersed into the appropriate portions of the lab report.