ECE 385

EXPERIMENT #2

Data Storage

I. OBJECTIVE

In this experiment, you will design and construct a simple 2-bit, four-word shift-register storage unit.

II. INTRODUCTION

Conceptually, random access memory (RAM) is a storage device arranged as a set of binary words that can be individually identified and accessed using unique addresses (see Figure 1).

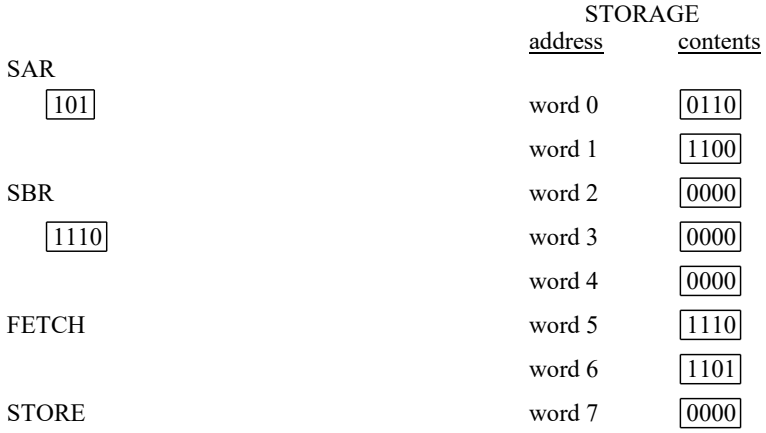

Figure 1: An Eight-word Storage Unit Using 4-Bit Words

2.2

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).

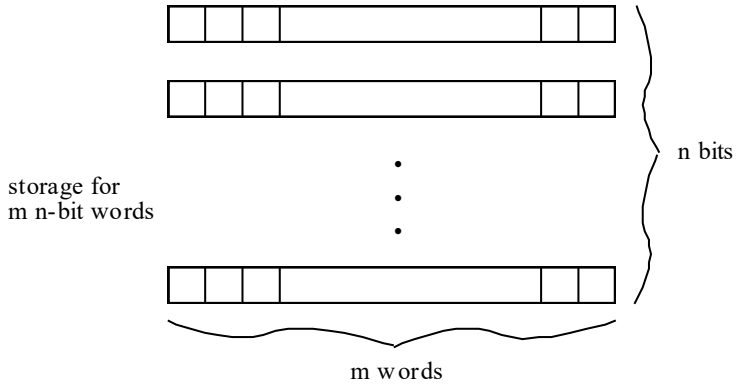

Figure 2: Configuration of a Shift-register Storage

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~~ 4-bit shift 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~~shift register chip. 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. In addition, depending on the shift register you end up using, you may need to wire the parallel outputs to the parallel inputs, which is allowed.~~–~~

Formatted: No underline

Signal Definitions:

2.4

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 switches. FETCH, STORE and LDSBR should also be obtained from switches. Display SBR1 and SBR0 on LEDs. 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 ~~have to~~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 ~~over and over again~~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~~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. ~~Notice that all of the registers must be continuously shifting, including the SBR. Thus, it is easy to see that t~~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), ~~again depending on the current operation.~~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 ~~properties~~ 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

2.6

"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 ~~has to~~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).

Formatted: Font: Bold

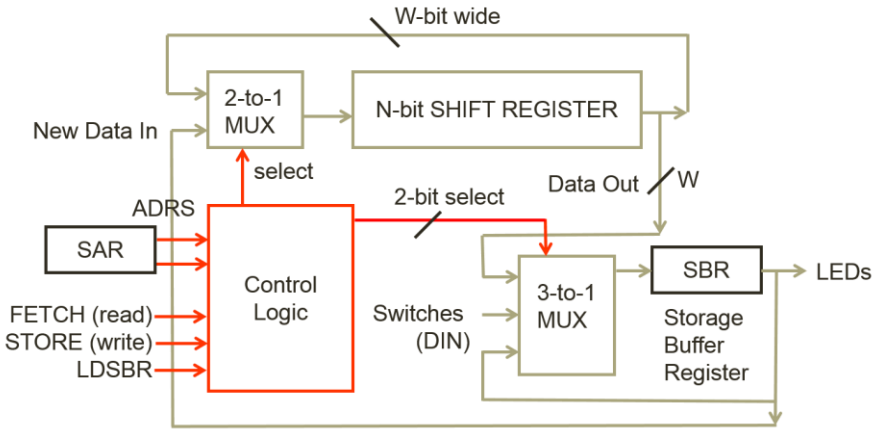

Figure 3: Block diagram of the shift-register storage unit.

~~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 at the moment when the internal data address from the 2-bit counter~~

~~matches the SAR. This random cell Z will now be associated with the address Y. Later on, when you wish to fetch from address Y, you wait for the internal counter to match the SAR again, and 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 in order to control the MUXs, the 'select' signals generated by the control logic has to 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).~~

HINT: Use the ~~Pulse Generator~~either a debounced switch or the FPGA (we will provide a bitstream) to provide a basic clock. Continuously clock the shift-register and use a counter that keeps track of which word is currently available. Use combinational circuitry (or ~~74LS85~~a comparator) 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 ~~in order to~~to strep through your circuit (why?).

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).

2.8

IV. LAB

Finish testing and demonstrating your circuit to your TA. Correct the design if necessary and document your changes for the lab report.

Follow the Lab 2 demo information when debugging is completed.

V. POST-LAB

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

Write a report, you may follow the provided outline below, or make sure your own report outline includes at least the items enumerated below. Note that this is a group report, so only one partner will need to hand-in the report (decide who uploads the report and make sure they do it before the due date!)

1. Introduction
    a. 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
    a. Describe how the addressing is implemented. When does the circuit commit a read or write from/to the input switches and output register respectively?

Formatted: Bulleted + Level: 1 + Aligned at: 1" + Indent at: 1.5"

Formatted: Indent: Left: 0"

Formatted: Underline

Formatted: Indent: Hanging: 0.25"

   b. 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.
   c. 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
   a. High-level description
      i. Describe in words what components are necessary to perform the operations described in the written description of the memory circuit operation.
   b. 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
   a. Provide an intuitive written description of your control unit.
   b. Include a block diagram of your control unit. It is acceptable to turn in either:
      i. A single high-level block diagram, which contains the sub-components inside the control unit.
      ii. 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
   a. If you used k-maps or truth tables during design, include them here. (If you didn't need them, you don't need to include them).
   b. You do not need a state diagram for this lab.
   c. Written description of the design considerations taken (did you consider multiple implementations of the same circuit and the tradeoffs of each?)
   d. Detailed Circuit Schematic
      ~~6.~~i. 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. This should be generated with Fritzing, but

Formatted

Formatted: Font: Not Bold

Formatted: Font: Not Bold

you must make sure the schematic is well organized and legible, e.g. no 'airwires'. ~~Use net labels to break your schematic into functional blocks.~~

6. Component Layout Sheet / Photograph of your breadboard design
    a. For the online version of the course, it is required that you use Fritzing for the component view, which will be generated from the breadboard view in Fritzing. You will likely require multiple breadboards. Make sure your breadboard reflects your actual implementation and should look like an illustrated view of your breadboard photo.
    ~~6.~~ Your lab report should include photograph of your circuit as built on the breadboard. Individual wires need not be clearly visible as your component layout suffice for documenting the wiring.
7. Description of all bugs encountered, and corrective measures taken
8. Conclusion
    a. Summarize the lab in a few sentences
    ~~b.~~ 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.

Formatted: Font: Not Bold

Formatted