ECE 385

Fall 2019

Data Storage

Pouya Akbarzadeh and Patrick Stach
Section: ABC
Lab TA: Vikram Anjur, Yuhong Li

Purpose of Circuit
In this experiment we designed and built a circuit which was a 2-bit, 4-word shift-register storage unit. The labs purpose was to utilize multiplexers, flip-flops, and shift registers. This lab is useful in context of architecture because the processor needs to store and shift data all the time. This simple 2bit, 4-word shift-register storage unit gives us a glimpse of what to expect.

Written Description of Circuit
The circuit built in this lab used various chips to read, write, and save data. The user input is read and processed through the control logic. The control logic has some user input to drive the circuit to work in the intended way. Some of these inputs are directly from the user and some have gone through some logic. These inputs include Store, Fetch, and LDSBR which are directly from the user. Another input is from output of the comparator which uses SAR and the Counter. The detailed logic can be seen under "Logic Diagrams" section. The control logic had controlled a dual 2-to-1 and dual 3-to-1 multiplexers. The multiplexers had a 1-bit and a 2-bit select respectively. Please note that the reason we have dual multiplexer setup is due to the fact that we are working with 2-bits.

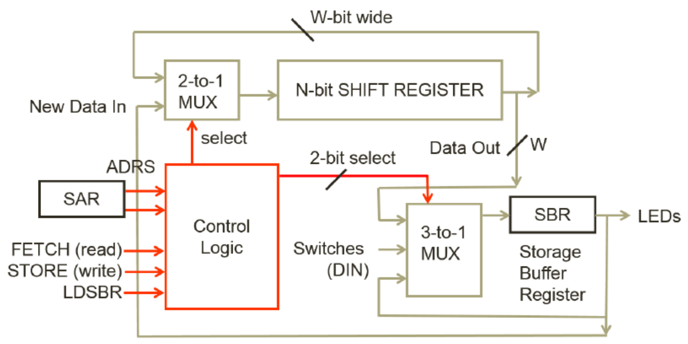
Shifting focus to the dual 2-to-1 multiplexers that is driven by the control logic we see that it has two inputs. The dual 2-to-1 multiplexer controlled whether the data used will be the old data loaded from the SBR or new data that was input by the user. The select bit was controlled by Control Logic. The data output from the 2-to-1 multiplexers are put into the shift register. The n-bit shift registers simply right shifted the data on the rising edge of the clock. Please note that this shift was a circular right shift. The data output was fed back to the 2-to-1 multiplexer and to the dual 3-to-1 multiplexer. The dual 3-to-1 multiplexer gets feedback from the SBR, n-bit shift registers and user input from the switched (DIN1 and DIN0). There was no 3-to-1 multiplexer so instead a 4-to-1 multiplexer was used (SN74153).

The storage buffer registers previously mentioned as SBR the very last component before information is passed to the LEDs. Also take not that every feedback has to go through the SBR. The SBR and SAR, storage address register, work hand in hand. The word address, chosen by the user, is put into the SAR and the binary data is stored in the SBR when the user enables the Store switch. Please note that the pervious content stored would be deleted after the store operation takes place.

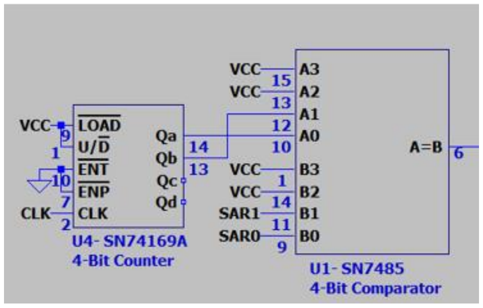
This design and implementation should be the best because the logic has been simplified using De-Morgan's law alongside basic logic operation rules to ensure most simple wiring. However, using this many chips caused the wiring to still be messy. There was no way to make this circuit clean and easy to debug. However, that is the case with most implementation. Our implementation did not gate the clock, gating the clock could have caused a lot of issues, and performed perfectly with both high and low frequencies during the demo.

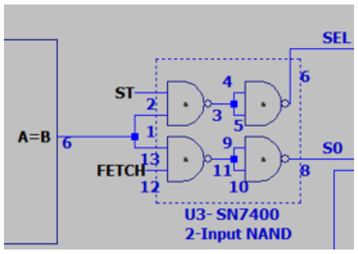
Some changes that had to be implemented from the pre-lab/design phase are as follows. The first change was simplification of the logic. Our first approach was much more complicated and harder to debug. The final versions can be found under the "Logic Diagrams" sections. The changes and simplifications were made by creating K-maps and gate translations. A problem that was presented initially with the circuit was incorrect grounding of certain pins. Some pins must have a 0 input, and some with high,1, input. The incorrect grounding of certain pins caused a lot of issues. The first obvious issues was that the output was incorrect. The second issue was that it made debugging harder because it led us to believe that the issue was caused by our logic not incorrect wiring. This issue was fixed after reading the data sheets provided by TI. This error was made on the SN7485. Instead of having 3 pins on high they were on low and thus did not give us the desired output.

High Level Block Diagram and Logic Diagrams



Control Logic





Comparator for SAR and Counter- Comparator will output 1 when SAR matches current counter output

Select BIT Logic-

SEL will be 1 when the comparator output is 1 and ST is 1. This mean we will only store new data when the user indicates we want to with the ST switch and only when the address selected matches the address from the counter.

S1 will be 1 completely dictated by LDSBR switch pressed (1), and S0 will be 1 when comparator output and FETCH switch are 1

State Diagrams and Tables

4:1 MUX
S1= LDSBR
S0= Fetch*Comp
(Comp is comparator output, when SAR matches binary counter)

S1_S0= 00 => Select from SBR
S1_S0= 01 => Select from Shift Register
S1_S0= 10 => Select from DIN switches

LDSBR	Fetch	FETCH	S1	S0
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	1
1	1	0	1	0
1	1	1	1	0
1	1	0	1	0
1	1	1	1	1

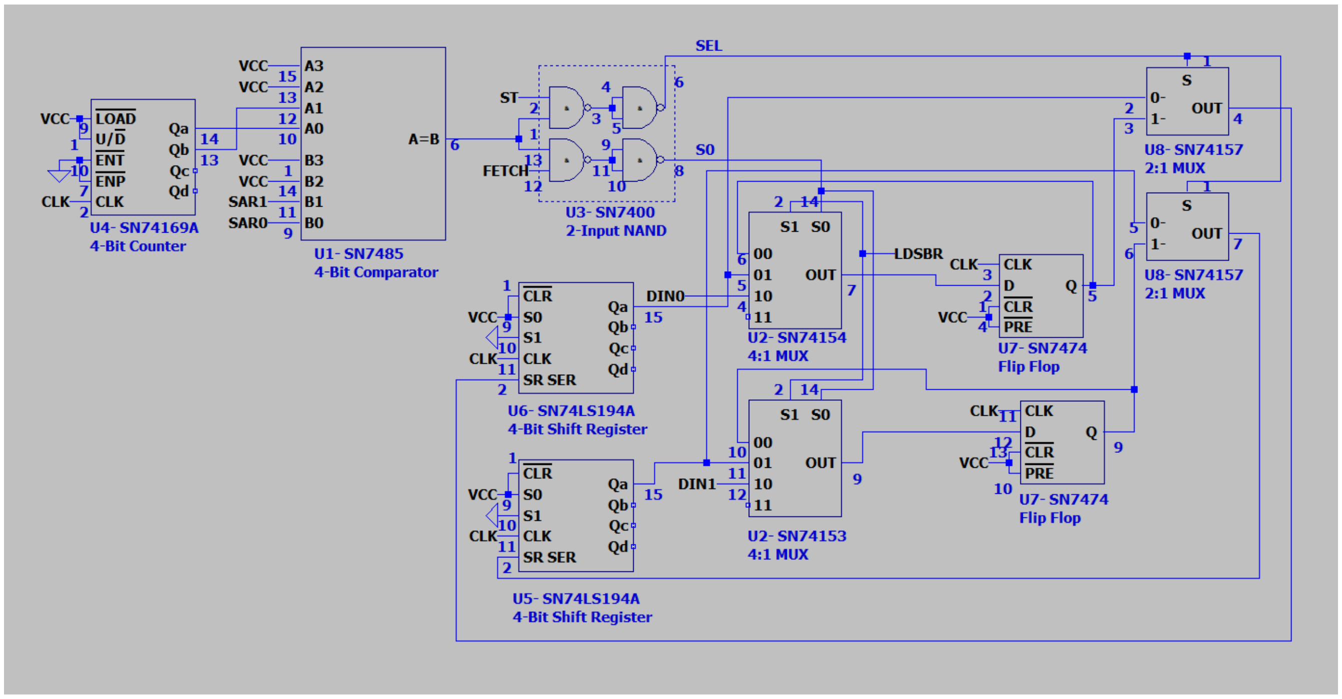
2:1 MUX
SEL= ST*COMP
SEL=0 indicates we will load old data into register (circular shift right)
SEL=1 indicates we will load new data from SBR

ST	COMP	SEL
0	0	0
0	1	0
1	0	0
1	1	1

Switch Mapping

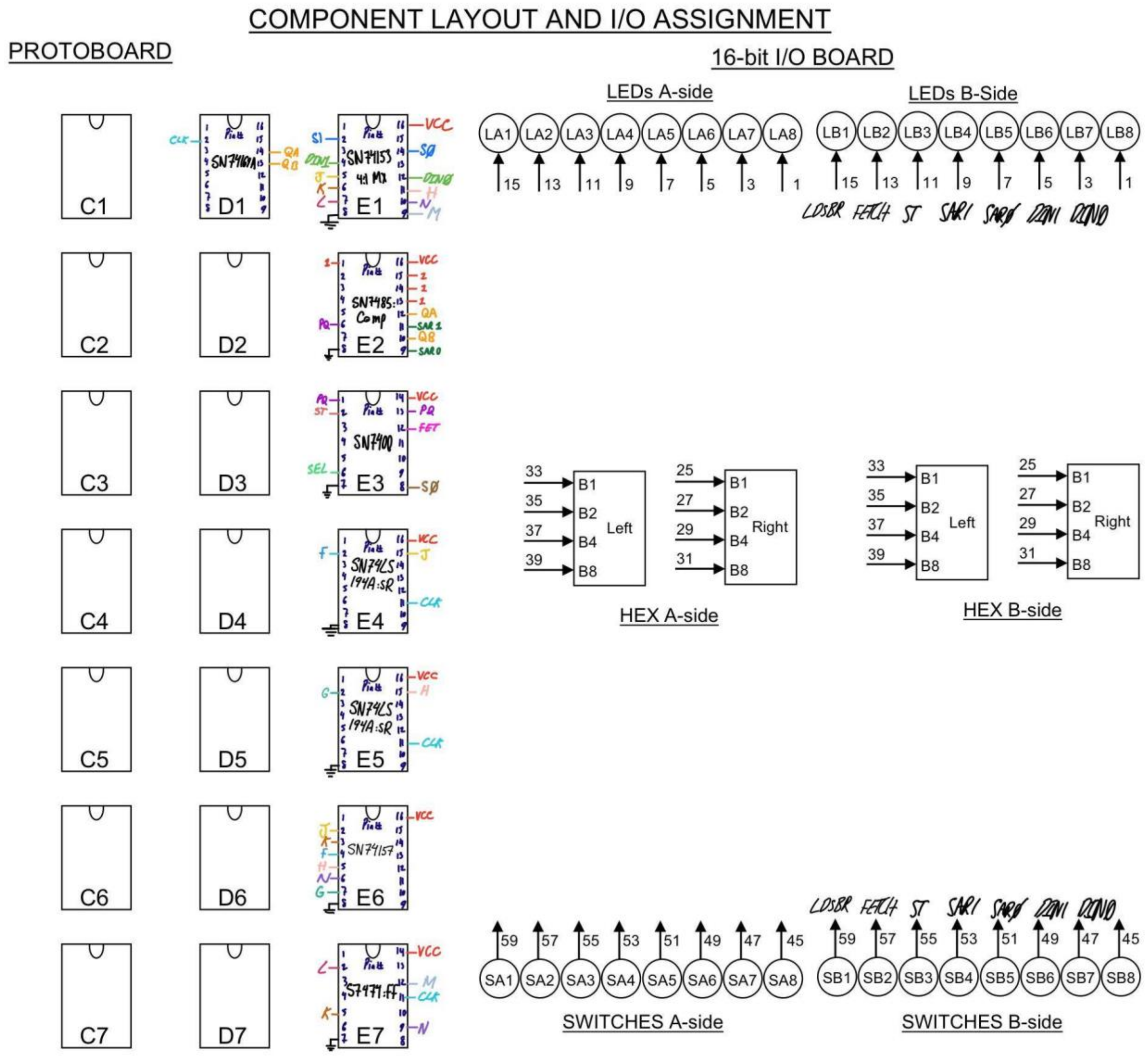
1	2	3	4	5	6	7
LDSBR	Fetch	Store	SAR1	SAR0	DIN1	DIN0

Circuit Schematic



Single chip with multiple parts (i.e dual 4:1 MUX) are noted by same reference number (i.e U1) but different pin numbers

*Component Layout**(Including User Input and Circuit Output)*



Answers to Pre-Lab Questions

A. *Your pre-lab writeup should contain a written description of your circuit operation, a block diagram, operation of the controller, a logic diagram and layout documentation. 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*

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 2.7 switchbox. Only the clock input needs to be de-bounced to strep through your circuit (why?).*

The switch need to be debounced because real switches are not perfect. As the name suggests there is this "bounce" that needs to be taken care of. This bounce happens because the contacts within the switch vibrate before settling in the new position. This bounce causes the output to rapidly fluctuate between high and low for a short time period, which would cause any sort of operations triggered on the rising edge of the clock to happen multiple times with one press. There are many different ways to approach this issue. One of those is a simple circuit which is presented in the general guide. An easier fix to that is the use of the switch-box available in lab.

Only the clock input needs to be debounced because the clock input goes to components such as flip flops, shift registers, etc that operate on the rising edge.

Answers to Lab Questions

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?

We have a binary counter to keep track of the current address that represents the rightmost bit, the bit we are accessing. We cannot just write data in whenever because we don't want to overwrite any data in the incorrect address. We need to wait not only when the user specifies to write, but also when the address indicated by the switches matches the address indicated by the binary counter. These write operations also occur only on the rising edge of the CLK because of the synchronous operation of the circuit.

The MUX chooses its inputs based on the select bits. Our control logic generates select bits the 4:1 MUX, and the state that where you load data into SBR is when S1= 1 and S0=0 (10 on the MUX)

Answers to Post-Lab Questions

1) *Your post-lab writeup (notes) should contain a corrected version of your prelab writeup and an explanation of any remaining problems in the operation of the circuits. This will aid the writing of your lab report.*
   Changes explained in the last paragraph of circuit description.

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?

   A shift register can simplify the complexity any sort of operations that take place because operations will only need to work on one bit at a time (LSB). However, we need to wait through various clock cycles to reach the desired state (in our case time negligible in high frequency clocks). With SRAM of the same size, we would be able to access memory quicker since we don't need to wait for a clock cycle, but the complexity to access specific bits would increase because we would need to run wires to each flip flop instead of simply the least significant bit.

   We made sure that the counters and shift register chips were all synchronous because in order for the data bits to be shifted at the same time (2 bit words, both bits shifted at the same time), we need all the actions to happen on the rising edge of the shared clock. If we did not use synchronous chips, we run the risk of data being corrupted by not being shifted simultaneously

Conclusion

This lab we designed and built a circuit which was a 2-bit, 4-word shift-register storage unit. The labs purpose was to utilize multiplexers, flip-flops, and shift registers. This lab is useful in context of architecture because the processor needs to store and shift data all the time. The design process allowed us to choose between multiple chips and design what we thought was going to perform the best in both high and low frequencies. This lab also had an emphasize on correct implementation of a control logic which drove multiple multiplexers.