Experiment 2,
Simple shift register memory
ECE 385

Jordan TAN, Jeffrey HUANG

January 31, 2017

Date Performed: January 31, 2017

1 Objective

The objective of this lab is to create a simple 2 bit 4 word memory unit using TTL chips. The design of this circuit revolved around the use of 4 bit shift registers which are abstracted away with a comparator and counter unit.

2 Description of Circuit

2.1 Operation

2.2 Addressing

Addressing was implemented with a counter and comparator. The counter would increment for every clock signal. Since the shift register moves the value that is evicted from the register to the last index in the register, each address is relative. After four clock cycles, the values of the register is refreshed. When the comparator, loaded with the SAR signal, matches with the current address on the counter then it means that the relative address on the shift register matches with the address that we are trying to use.
Data is only read or written when the store or fetch signals are high and the SAR address matches the current relative address.

2.3 Read

The fetch instruction loaded the value in the SAR index of the shift register and loaded it into SBR. This was how data was retrieved from the memory unit. The SAR index was controlled by the switches on the IO box.
The SBR was connected to two LED's on the IO box and when this signal is high then the LED's will light up to reflect the values stored in that register.

The circuit only reads when the address we are trying to use matches the address on SAR and the fetch signal is high. The value of the select bit on the 3-to-1 mux is set to 0 and the data is loaded from the SAR into the SBR.

2.4 Write

The store instruction took the value in SAR and loaded the value set on the two external switches into the respective memory location in the shift register. The SAR memory location was controlled by two external switches on the IO box. For example, if SAR was set to 11 and the data switches are set to 10 then the value 10 will be loaded into index 11 in the shift registers.

2.5 Implementation

The memory system was designed using two shift registers and two 2-to-1 multiplexers. The multiplexers uses the last element of the shift register and the SBR register as inputs. Using the multiplexer, the shift register wraps around so the data stays intact even after a large amount of clock cycles. On the rising edge of the clock, the shift register shifts all the elements to the right by 1 index and evicts the leftmost element in the register.

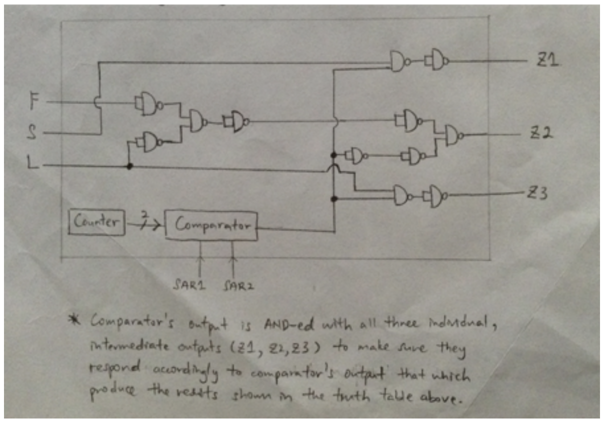
The addresses unit is made with a comparator and counter unit. The shift register has four indices and since it's shifting on every rising clock edge, every address is relative. On every rising clock edge, the counter is incremented by one and if the current count matches the address inputted by the switches, then the logic unit will enable fetching or storing. The SBR register is inputted with a 3-to-1 multiplexer. The multiplexers use the shift register, switches, and SBR outputs as inputs. When Fetch is high, then the multiplexer outputs the shift register. When loadSBR is high then the multiplexer uses the switches input as output. Otherwise, when the circuit does nothing the multiplexer uses the SBR output as input to maintain the state. We define $address = comparator(switches, counter)$. If the Fetch signal is high and $address$ is high, then the circuit will take the leftmost element in the shift register and store it in SBR. If the Store signal is high and $address$ is high, then the circuit will take the values from the switches and store it in the rightmost index in the shift register. The logic unit does this by changing the select bits on the 2-to-1 and 3-to-1 muxes.

The registers are implemented using D flip flops. The SBR is the register for the output of the circuit. Whatever is loaded on the SBR is output to the LEDs. The input to the circuit are a bunch of switches and 3 switches that control the Fetch, Store, and LoadSBR instructions.

3 Design process

3.0.1 Control unit

The control unit was made with 12, 2 input NAND gates

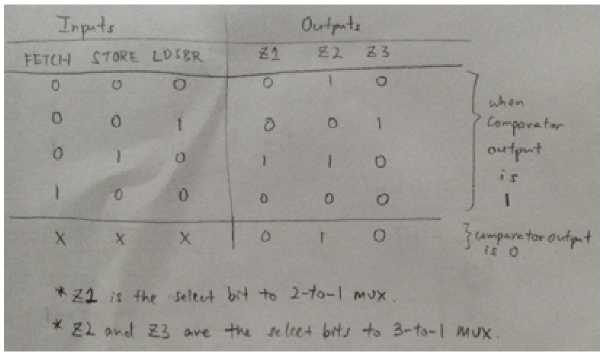


* Comparator's output is AND-ed with all three individual, intermediate outputs (Z1, Z2, Z3) to make sure they respond accordingly to comparator's output that which produce the resetts shown in the truth table above.

Figure 1: Comparator output is AND-ed with all three individual outputs, z_1, z_2, z_3 ot make sure they respond accordingly to comparator's output that which produce the resets shown on the truth table.

3.0.2 The Truth Table

The control unit was designed around the fetch, store, and loadSBR inputs and output as the mux select bits. 2.

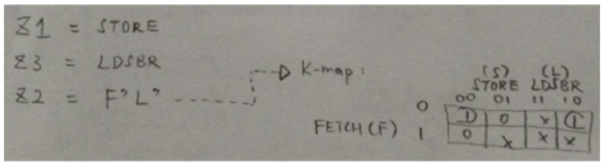


Inputs			Outputs			
FETCH	STORE	LDSBR	Z1	Z2	Z3	
0	0	0	0	1	0	when Comparator output is 1
0	0	1	0	0	1	
0	1	0	1	1	0	
1	0	0	0	0	0	
X	X	X	0	1	0	comparator output is 0.

* Z1 is the select bit to 2-to-1 MUX.

* Z2 and Z3 are the select bits to 3-to-1 MUX.

Figure 2: The truth table for the Control Unit.



$Z1$ = STORE

$Z3$ = LDSBR

$Z2$ = $F'L'$ → K-map:

		(S)		(L)
		STORE		LDSBR
	00	01	11	10
0	1	0	X	1
FETCH (F) 1	0	X	X	X

Figure 3: The truth table in a KMap.

3.0.3 Mux Layout

The Mux select bits came from the control unit as outputs.

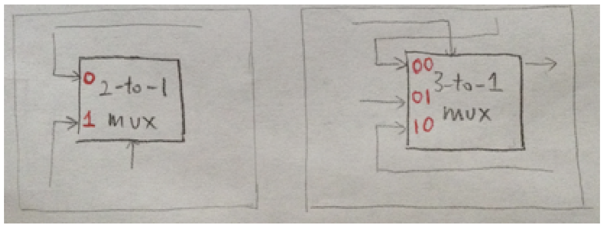


Figure 4: The mux labeling.

3.1 Datapath

The datapath of the circuit

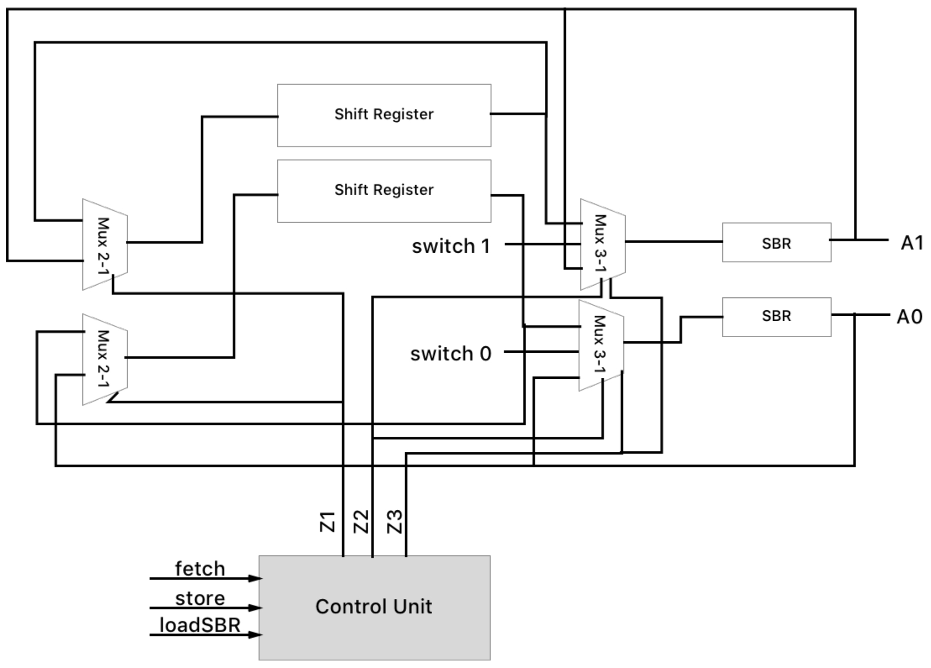


Figure 5: The Datapath.

3.2 Component layout

We separated the circuit into two main sections, the control unit, and the datapath. The control unit resided at the top of the breadboard, closer to the IO board controls. The datapath was on the bottom of the breadboard. Since the design called for two of each unit, one unit for each bit, we put all the units next to each other. The two shift registers were next to each other, luckily most of the chips had both components on the same chip. This saved a ton of space.

The control logic very densely packed on the top of the breadboard and there were three input wires that stuck out of the right side of the control logic. These were wired to three switches on the IO board. The output bits of the control logic were on the same side of the board but a different color wire. They were inserted into the datapath and controlled the muxes.

Most of the datapath was messy. There were alot of wires going everywhere and in the end it became a rat's nest. This contributed to the difficulty of the lab. If we organized the wires, moved the chips, and planned ahead the lab would have been easier to debug and complete.

4 Conclusions

We ran into issues debugging the circuit out of lab. Since the chips weren't working after reading the datasheets very carefully we suspected that the chips were broken. It turned out the switch that we used as a clock switched between 1.1V and 4.3V. The circuit interpreted this signal as a constant high. Because of this the shift register and flip flops never changed when we were debugging. It wasn't until we hooked it up to a multimeter did we realize that the switch was broken, not the circuit. At this point we had already torn down our circuit and didn't have enough time to rebuild it in the lab.