ECE 385

Fall 2019

Experiment #3

A Logic Processor

Pouya Akbarzadeh and Patrick Stach

Section: ABC

Lab TA: Vikram Anjur, Yuhong Li

1. Purpose of Circuit

The purpose of this experiment was to design and build a logic operation processor. The circuit should be able to do eight functions. These eight functions are AND, OR, XOR, 1111, NAND, NOR, XNOR, and 0000. The circuit should also have four different ways to route the result. The user should be able to choose between 4 possible outputs for A* and B* respectively: A and B, A and F, F and B, B and A. The following would be achieved by the use of plethora of multiplexers, alongside logic gates. This circuit is a 4-bit serial logic processor.

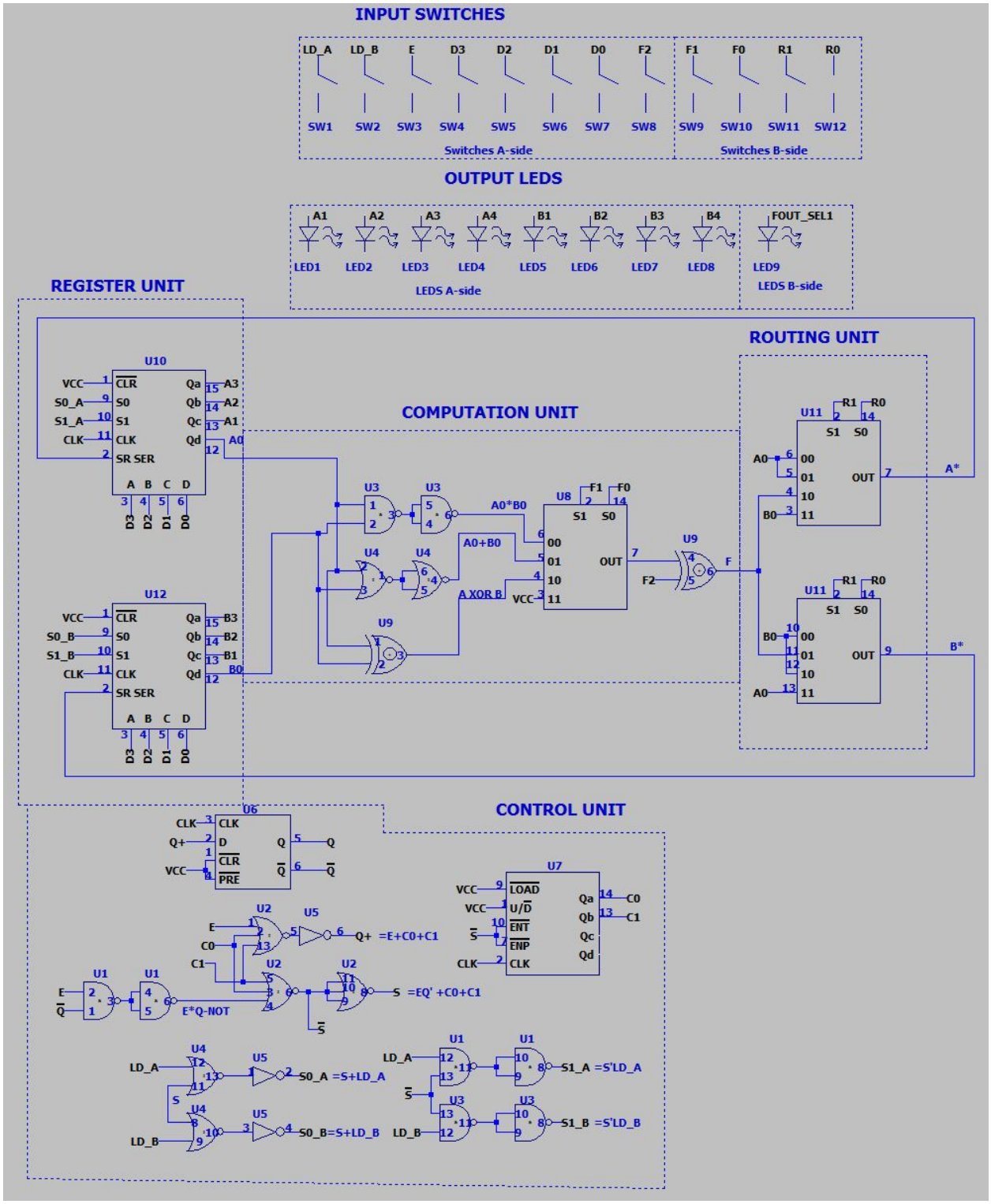
2. Logic Diagram and Written Description of the Circuit

2.1. Overview

See the table below for detailed information regarding the reference designators used in our logic diagram and circuit layout sheet.

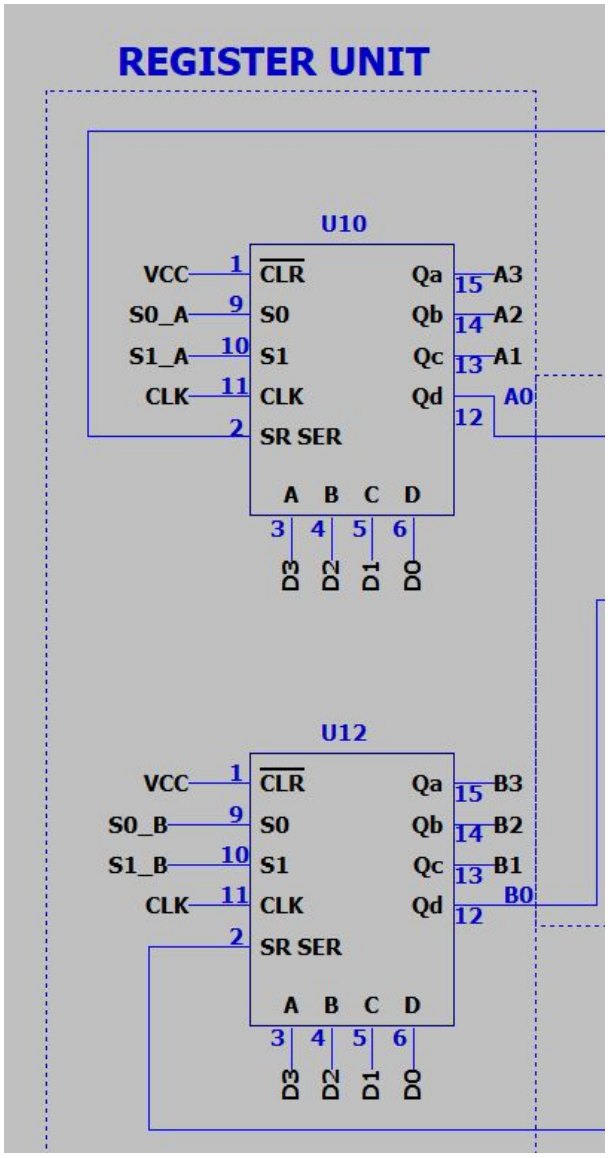
Reference Designator	Type	Description
U1, U3	SN7400	Quad 2-Input NAND
U4	SN7402	Quad 2-Input NOR
U2	SN7427	Triple 3-input NOR
U6	SN7474	Dual D Positive Edge Triggered Flip Flop
U8, U11	SN74153	Dual 4 to 1 Multiplexer
U10	SN74LS194A	4-bit Bidirectional Universal Shift Register
U5	SN7404	Hex Inverter
U7	SN74LS169A	4-bit Synchronous Up-Down Counter
U9	SN7486	Quad 2-input Exclusive-OR

The circuit is broken up into 4 modular units: the shift register unit, the computation/logic unit, the routing unit and the control unit.



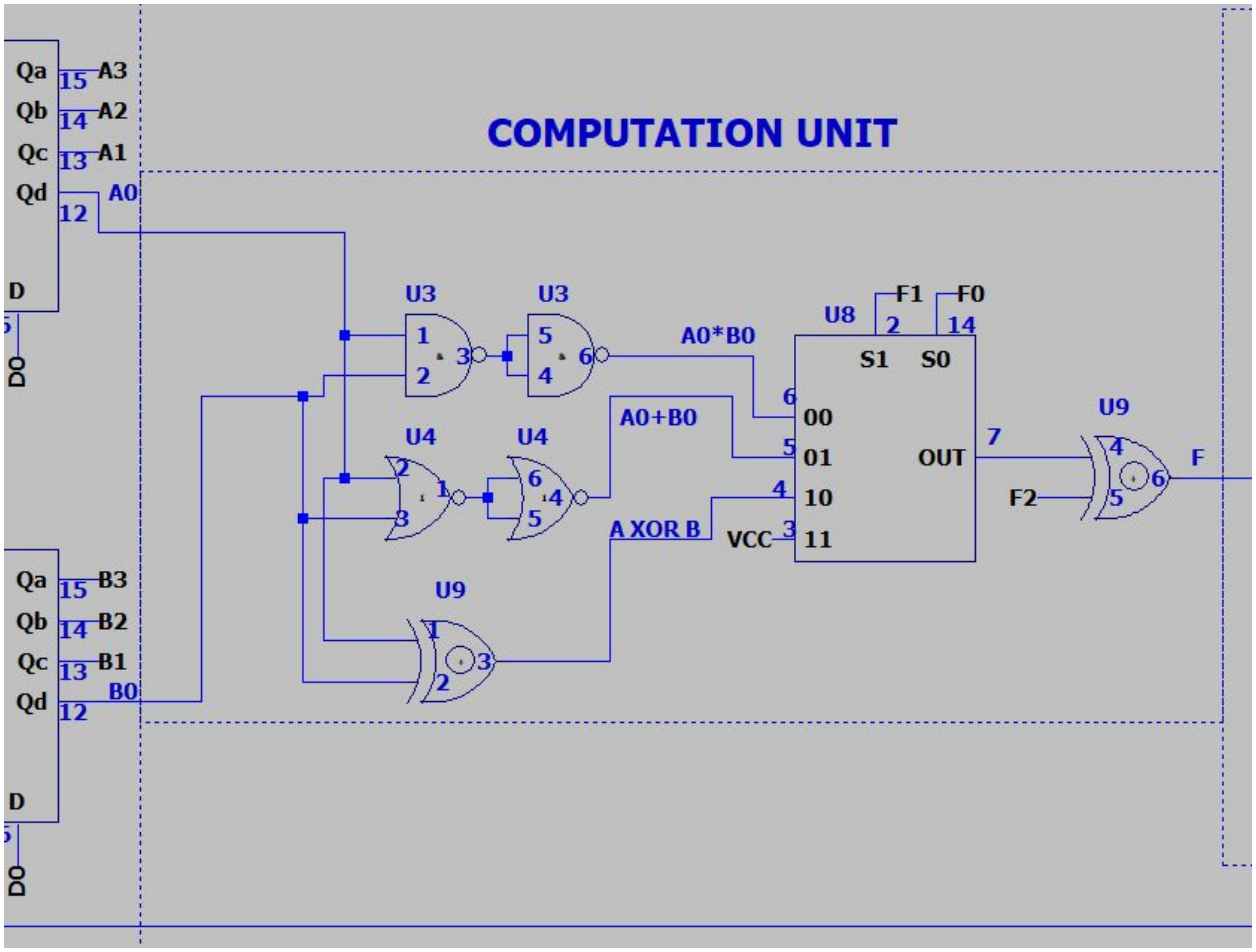
2.2. Register Unit

The shift register unit consists of two 4-bit shift register. A and B are 4 bit words (A3A2A1A0; B3B2B1B0), so we use one shift register to store A and the other shift register to store B. The LSB of both registers connects to the computation unit since computations are processed one bit at a time. The 4 data switches, D3, D2, D1, D0 are connected to the parallel load to both registers. The SR SER takes the new output from the routing unit, A* and B* and will be placed into the MSB (A3, B3 respectively) during a right shift. The shift register will change between 3 modes: shift right, parallel load and hold- these modes will be chosen based on the select bits from the control logic: for register A we have S1_A, S0_A; for register B we have S1_B, S1_A.



2.3. Computation Unit

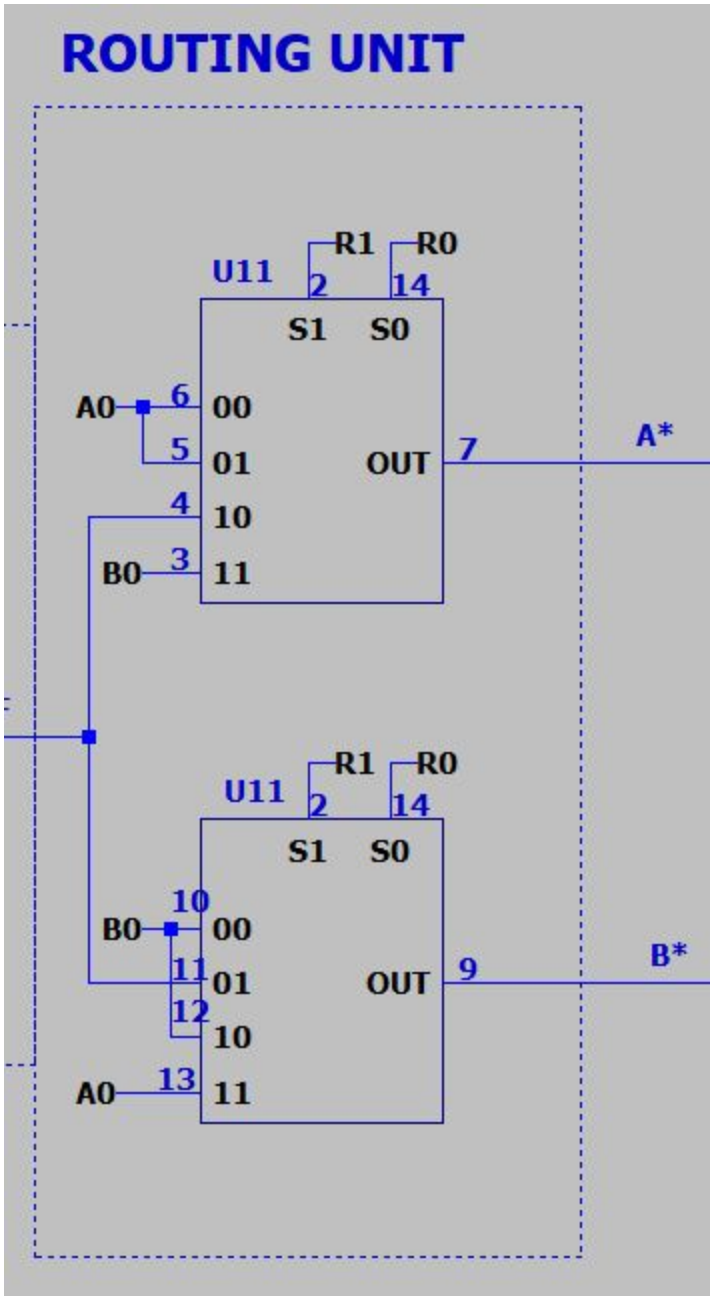
The computation unit takes the LSB from both registers A0, B0 and performs 4 different operations using logic gates that go into a 4:1 MUX- with inputs 00 is AND, 01 is OR, 10 is XOR and 11 is “1”. The F1 and F0 switches go into the S1 and S0 select bits of the MUX and choose between those 4 inputs. Then that output goes into a 2 input XOR gate, with the other input being F2. F2 will invert the output if F2 is 1 (ie: AND becomes NAND)



2.4. Routing Unit

The routing unit consists of two 4:1 MUXes (one for A*, one for B*) with the same select bit switches, R1 and R0, that will choose the A* and B* to go into the shift register. For the A routing MUX, select bits of 00 choose A, 01 choose A, 10 choose F (the output of the arithmetic

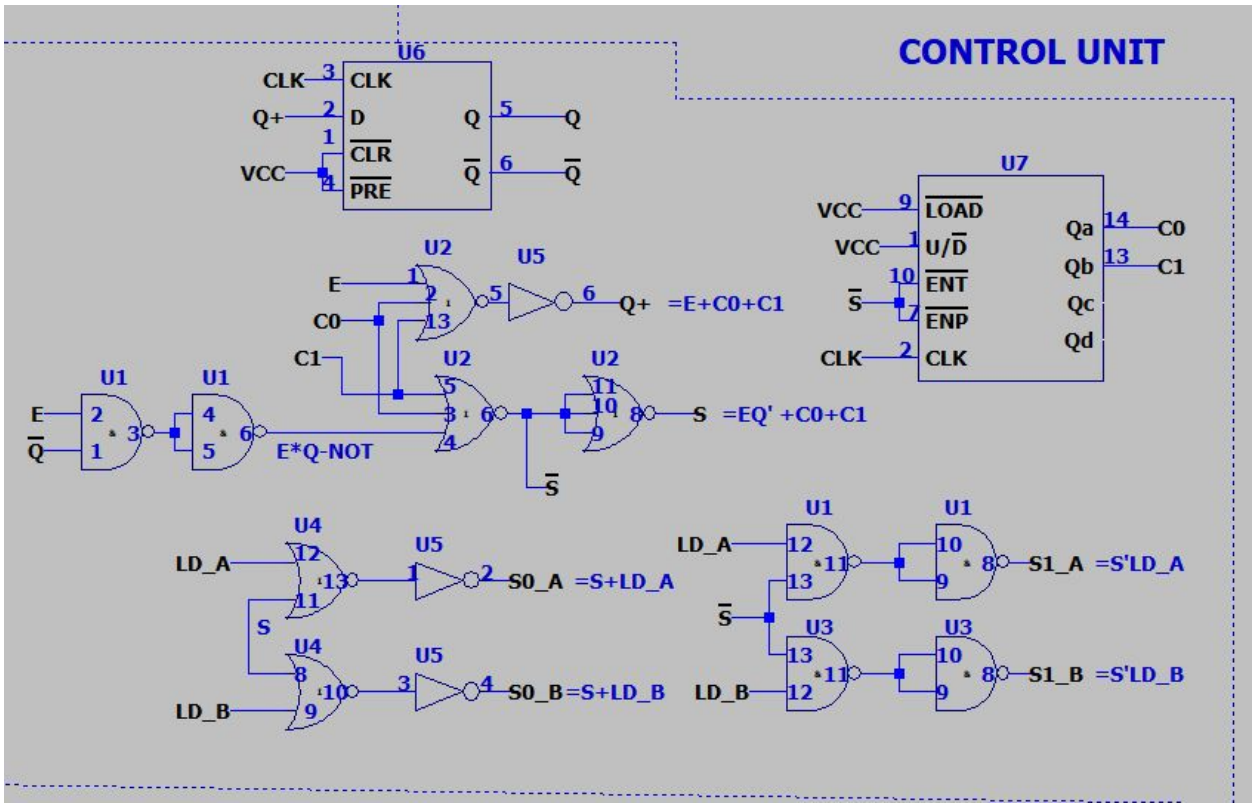
unit), 11 choose B. For B routing MUX, select bits of 00 choose B, 01 choose F, 10 choose B and 11 choose A.



2.5. Control Unit

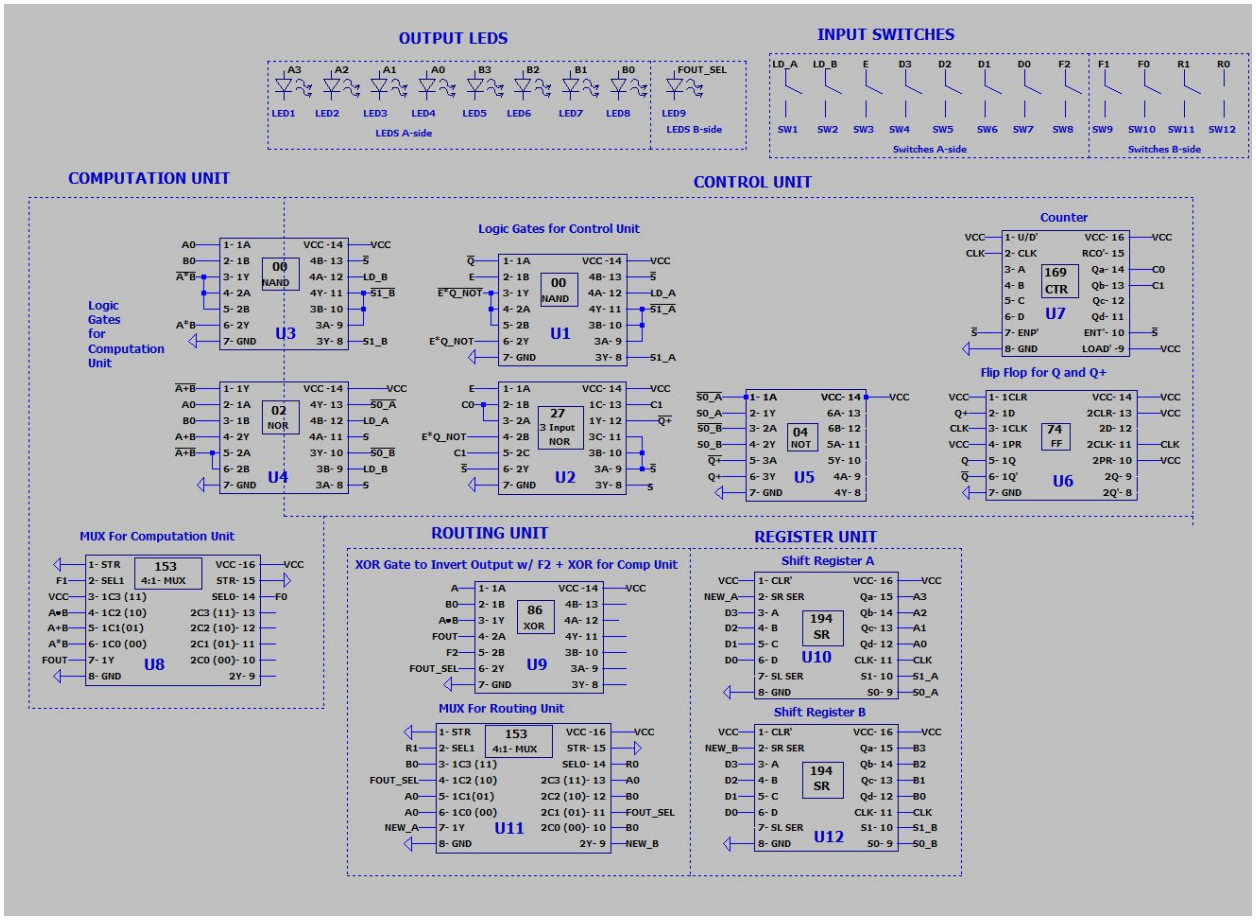
The control unit is responsible for sending select bits to the shift registers, multiplexers, and storing Q and Q+ simultaneously. The select bits are the outputs of a combination of logic gates. S is the select bit for whether the shift register should keep shifting (1= SHIFT) which is used to

enable and disable the counting of the counter. S1_A and S0_A are the select bits for shift register A to operate in a specific mode (SHIFT RIGHT, LOAD, HOLD), same with S1_B and S0_B for shift register A. The counter is used to keep track of the current state so that we can know when we shifted 4 times. The flip flop is used to be able to store Q and the next state of Q (Q+) at the same time.

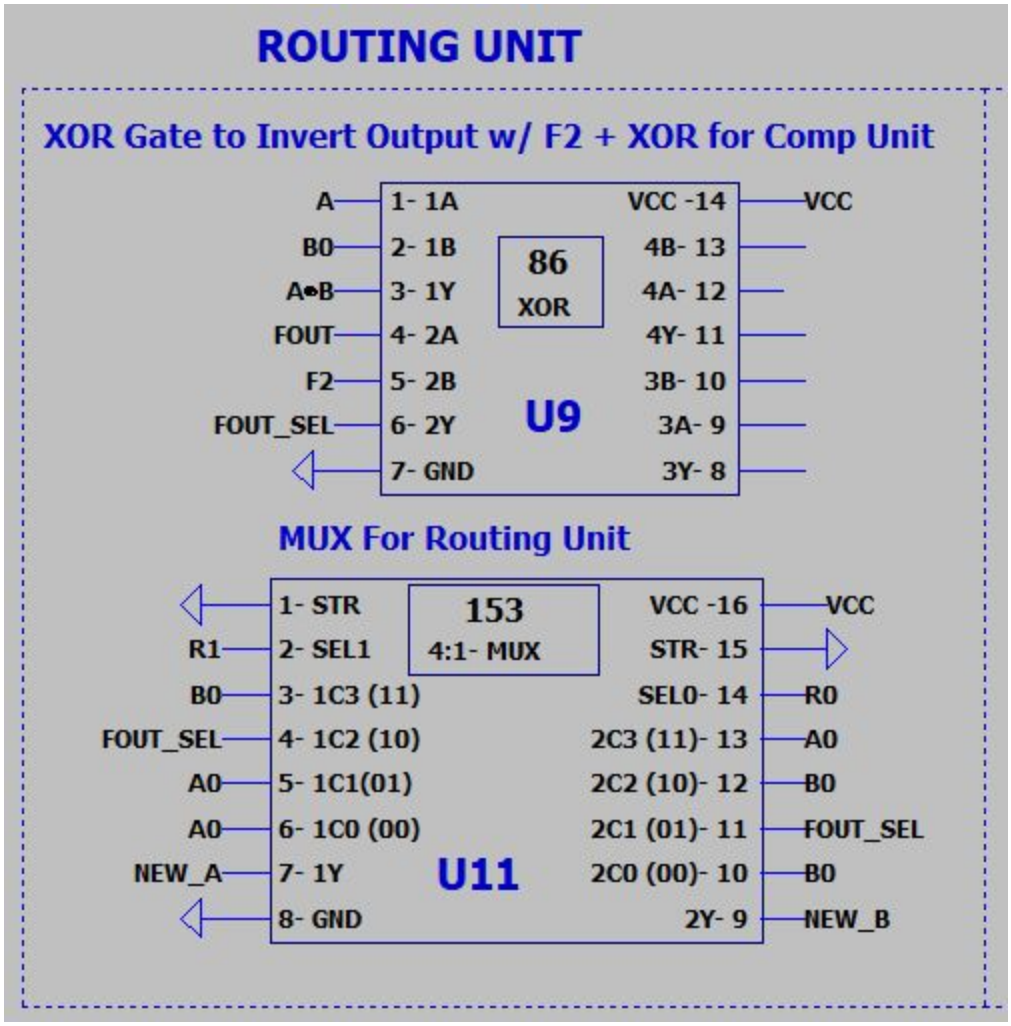


3. Circuit Layout Sheet

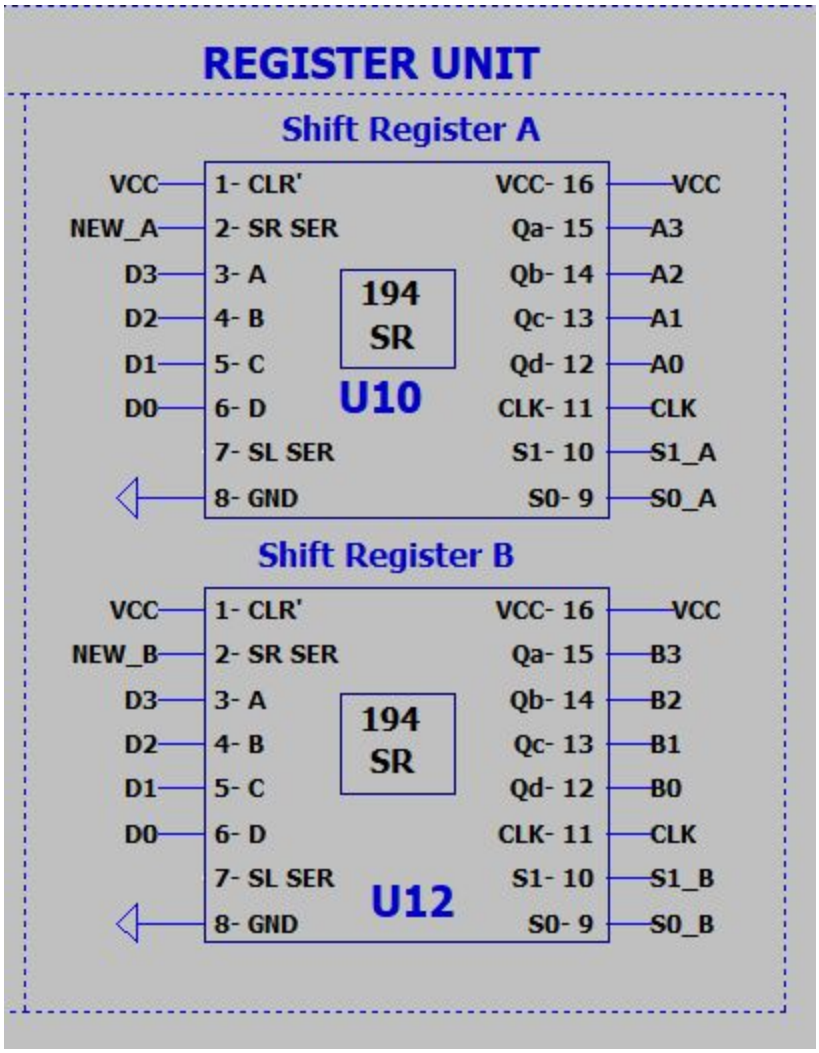
3.1. Full Layout



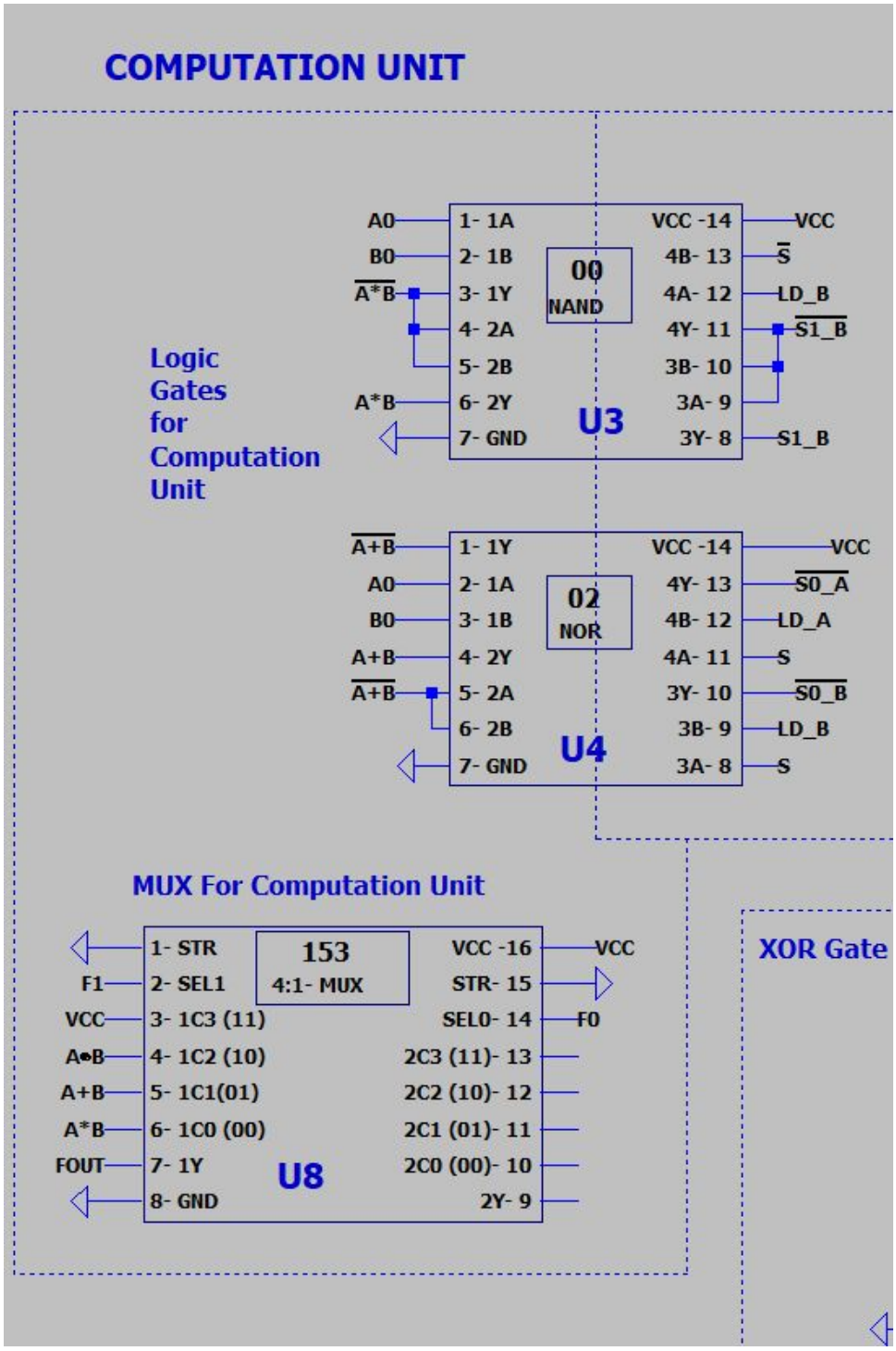
3.2. Routing Unit



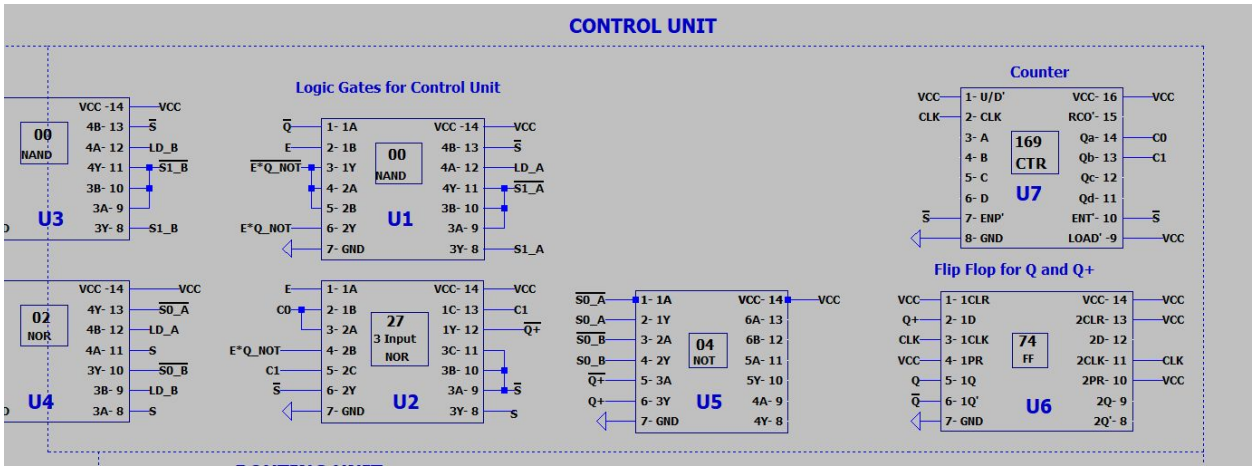
3.3. Register Unit



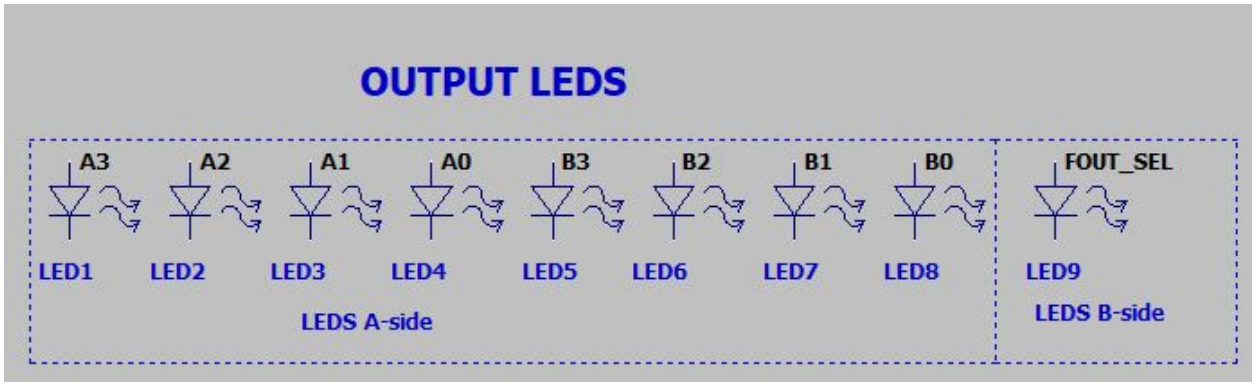
3.4. Computation Unit

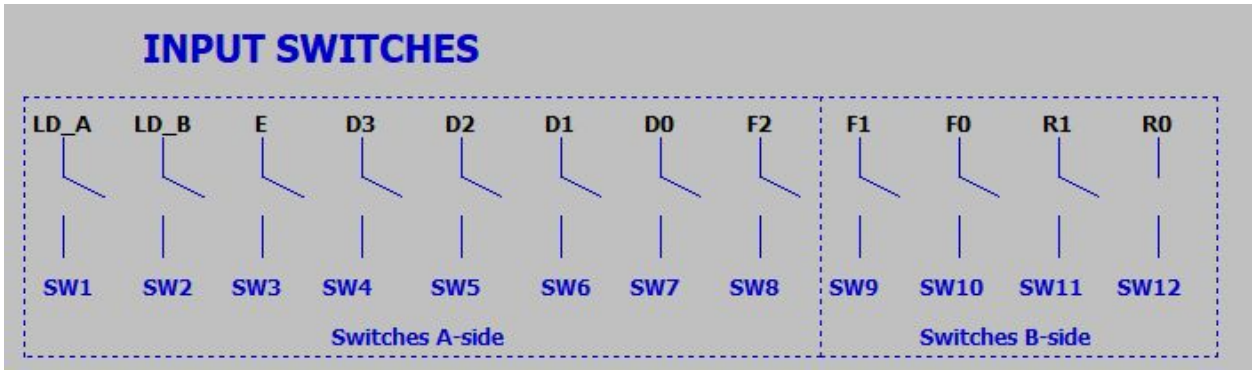


3.5. Control Unit



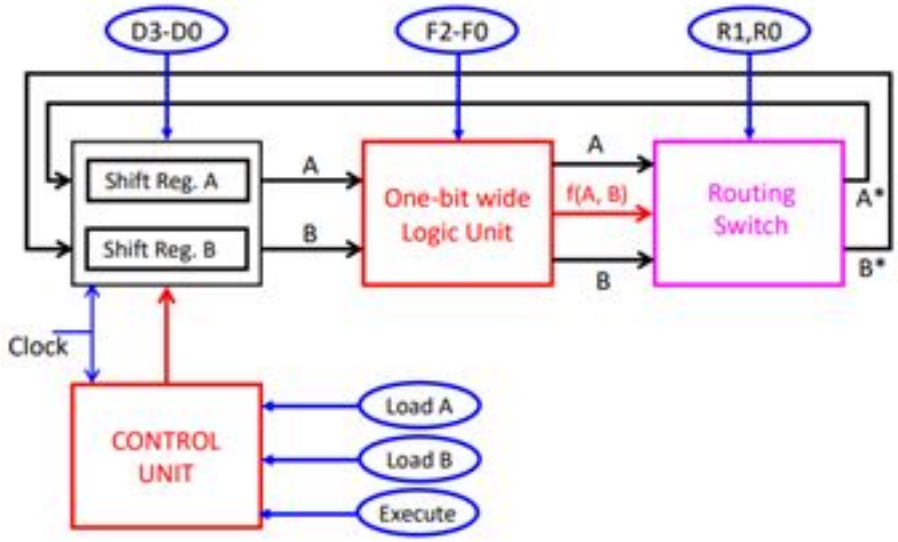
3.6. I/O Board



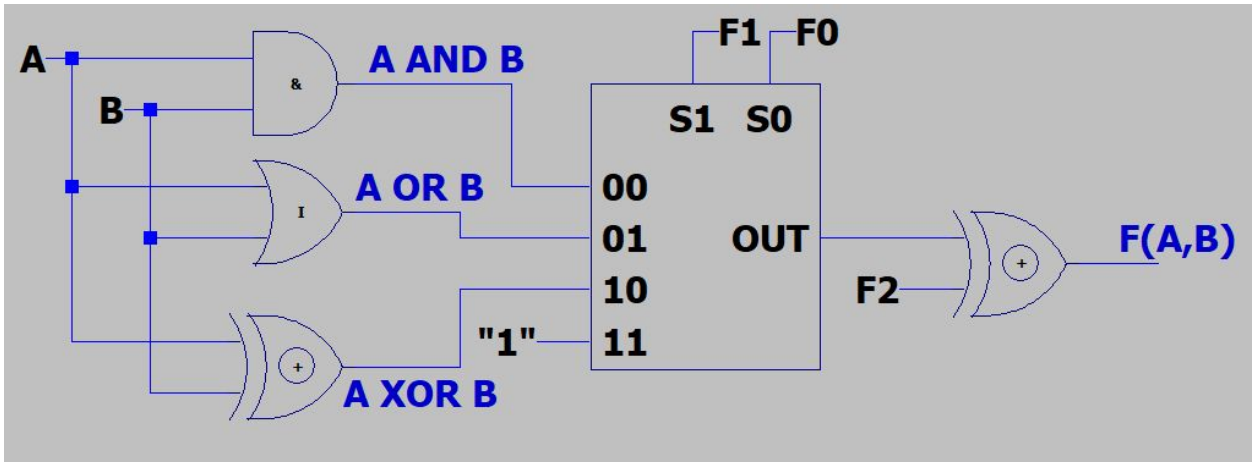


4. High Level Block Diagram

4.1. Overview

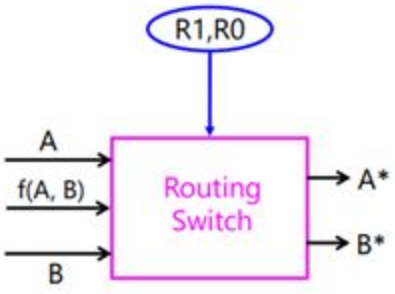


4.2. Computational Unit



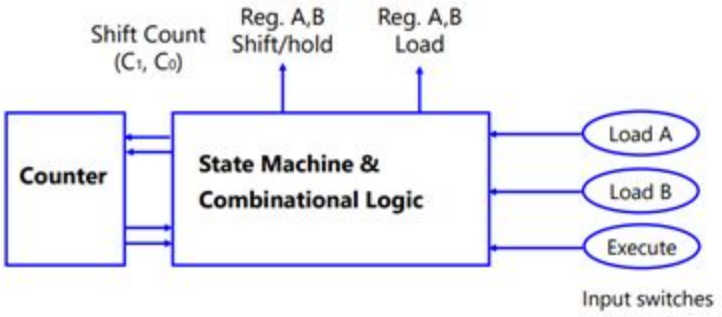
Function Selection Inputs			Computation Unit Output
F2	F1	F0	f(A, B)
0	0	0	A AND B
0	0	1	A OR B
0	1	0	A XOR B
0	1	1	1111
1	0	0	A NAND B
1	0	1	A NOR B
1	1	0	A XNOR B
1	1	1	0000

4.3. Routing Unit



Routing Selection		Router Output	
R1	R0	A*	B*
0	0	A	B
0	1	A	F
1	0	F	B
1	1	B	A

4.4. Control Unit



5. State Diagrams and Tables

5.1. Q+ (Next State of Q)

Q+

E,Q	C0,C1 = 00	01	11	10
00	0	X	X	X
01	0	1	1	1
11	1	1	1	1
10	1	X	X	X

Q+ = E + C0 + C1

5.2. S (Shift Bit)

S

E,Q \ C0,C1	00	01	11	10
00	0	X	X	X
01	0	1	1	1
11	0	1	1	1
10	1	X	X	X

S = C1 + C0 + EQ'

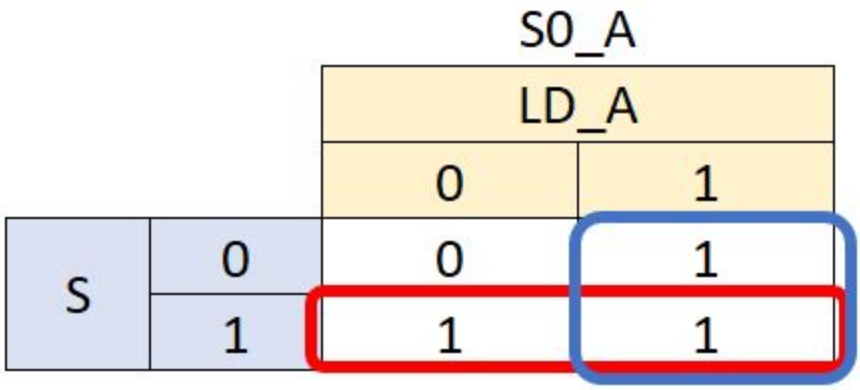
5.3. S1_A (Register A's S1 Select Bit)

S1_A

S \ LD_A	0	1
0	0	1
1	0	0

S1_A = S'LD_A

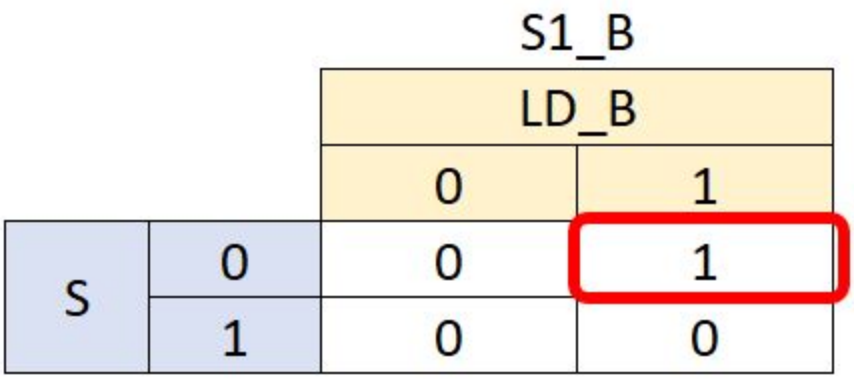
5.4. S0_A (Register A's S0 Select Bit)



		S0_A	
		LD_A	
		0	1
S	0	0	1
	1	1	1

S0_A = S + LD_A

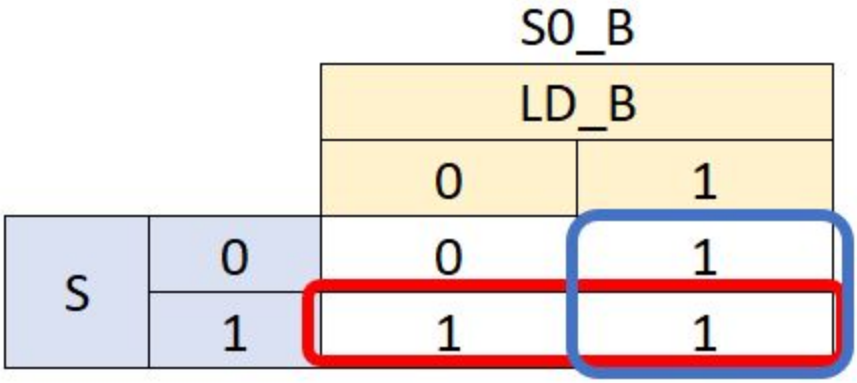
5.5. S1_B (Register B's S1 Select Bit)



		S1_B	
		LD_B	
		0	1
S	0	0	1
	1	0	0

S1_B = S'LD_B

5.6. S0_B (Register B's S0 Select Bit)



S0_A = S + LD_B

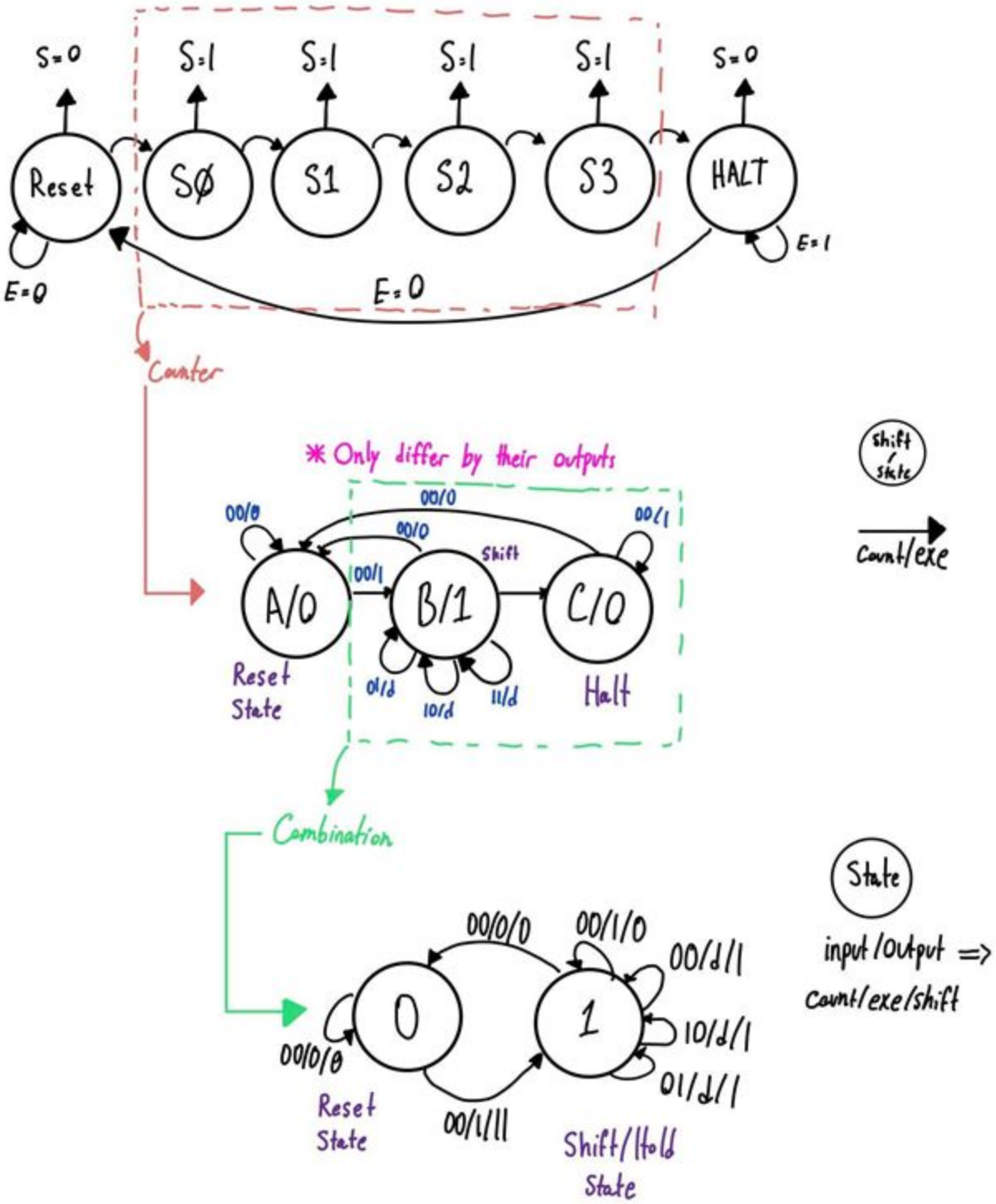
***SN74LS194A Shift Register Modes of Operation**

S1_S0 = 11 -> Parallel Load

S1_S0= 01 -> Shift Right

S1_S0= 00 -> Hold Values

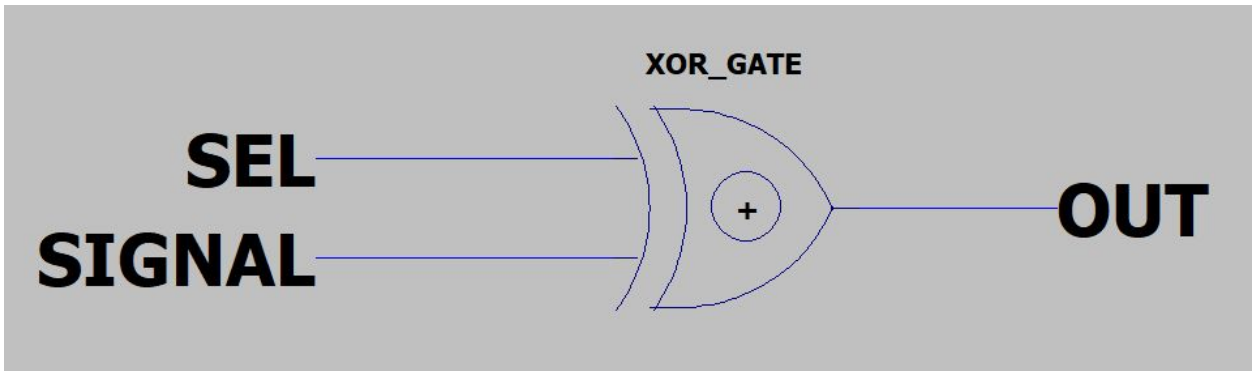
5.7. State Drawing



6. Lab Questions

A. Describe the simplest (two-input one-output) circuit that can optionally invert a signal (i.e., one input determines if the output is equal to the other input or equal to the other input inverted). Sketch your circuit.

The simplest circuit that can optionally invert a signal is an XOR gate. 1 input to the XOR gate will be the original signal (SIGNAL). The other one will be a select bit (SEL). if SEL is 1 the output will be the opposite of what the original signal was. For example: if SEL is 1 and SIGNAL is 1, the output will be 0, thus inverting it.



SEL	SIGNAL	OUT
0	0	0
0	1	1
1	0	1
1	1	0

Truth Table for Circuit

SEL	SIGNAL	OUT
0	0	0
0	1	1
1	0	1
1	1	0

SEL=0 makes OUT same as SIGNAL

SEL	SIGNAL	OUT
0	0	0
0	1	1
1	0	1
1	1	0

SEL=1 makes OUT opposite of SIGNAL

B. Explain how a modular design such as that presented above improves testability and cuts down development time.

Modular design is a good idea in terms of product development and production because it increases testability and increases reusability. It increases testability because it allows to test one specific part of the circuit at a time.

For example, when we were having issues at first with outputting the correct operation based on the function select, we focused only on the computation unit and were able to narrow down the problem. Had the routing unit been combined with the routing unit, for example, it would have been much more difficult to diagnose whether the operations were wrong, the selecting of function was wrong, or the selection of A* and B* were wrong.

Modular design is also very helpful for reusability. This lab wanted a specific combination of routing switches to result in a specific output. However, what if we wanted to reuse this circuit but with a modified combination of routing switches and outputs? With the modularity, we can easily reconfigure a MUX to have different inputs which would result in different outputs with those same

switches. Had everything been wired all in one, making modifications to the routing part of the circuit would inherently modify the computational part of the circuit too.

C. Design, document and build the circuit described in Part II. Your circuit should be able to perform correctly all the functions listed. You may use either 7495A or 74LS194A chips for your shift registers. You will want to study each of the chips carefully before deciding on one or the other. Be sure to make your design as efficient as possible (there is more than one way to design this circuit). A square wave from the Pulse Generator should be used as the basic system clock. Load A, Load B, Execute, D3-D0, R1, R0, and F2-F0 should be inputs from the switches. The control unit must be designed to perform the desired function once and only once each time the execute switch is flipped on. Results of the operation should be obtained even if the execute switch is flipped off in the middle of the computation cycle. You may only assume that the execute switch will remain high for at least one full clock period. Display the contents of Register A and Register B on LEDs. You may also want to include an LED that indicates when the computation cycle is complete for debugging purposes. 3.8

SEE LOGIC DIAGRAM

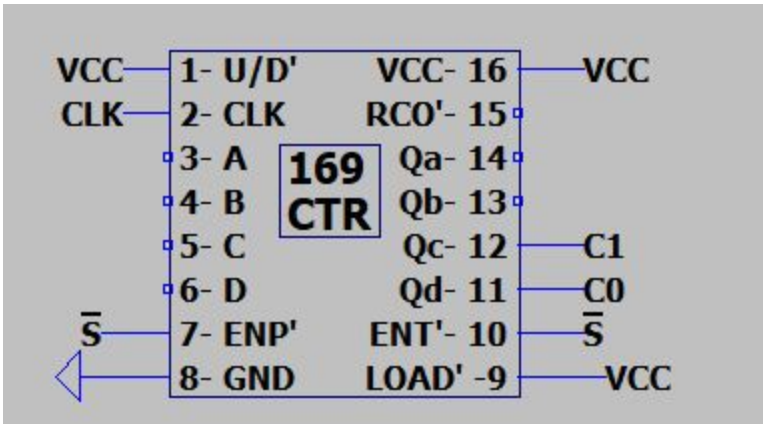
D. Work with your partner to wire-up the circuit.

DONE

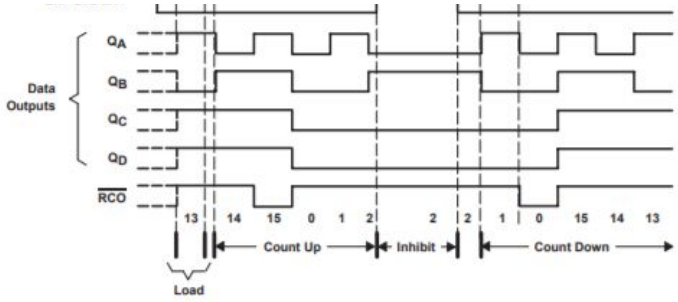
Document changes to your design and correct your Pre-Lab write-up, explaining any difficulties you had in debugging your circuit. Outline how the modular approach proposed in the pre-lab help you isolate design and wiring faults, be specific and give examples from your actual lab experience.

We ran into a few issues when building our circuit, and the modular design helped us troubleshoot our issues because we were able to test things one by one.

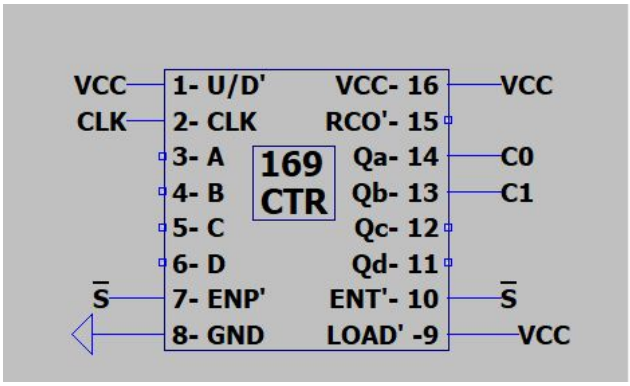
The first issue was with the way we wired up the counter.



Originally, we thought that Qd was the LSB and that Qa was the MSB for the SN74LS169A counter because that's the notation used for the SN74LS194A shift register. However, when troubleshooting the counter, we realized that the counter LSB was actually Qa

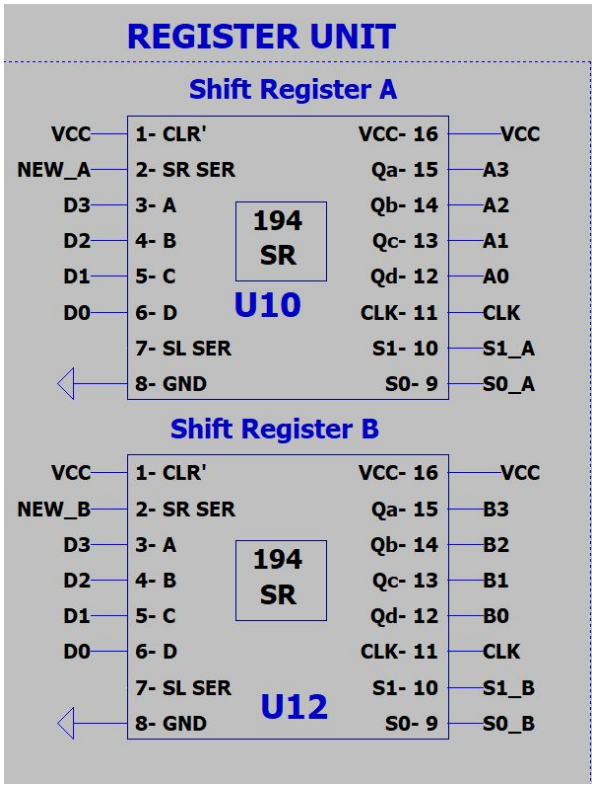


From this small snippet of the datasheet, it is obvious that we flipped the order of the bits, and that would have changed our logic significantly.



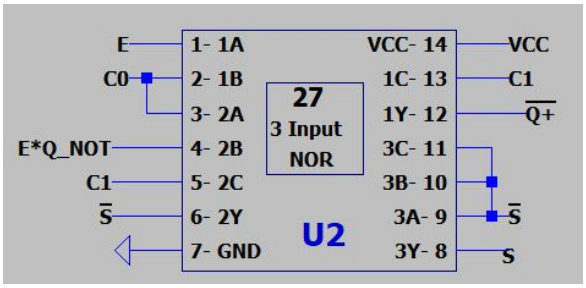
Fixed counter layout

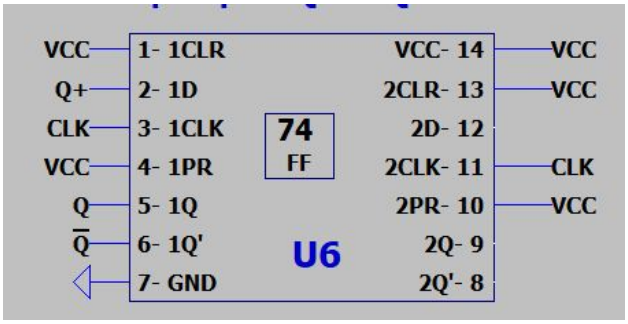
Another issue was that we used the same select bits for register A and Register B- our reasoning was that we didn't want one to be shifting while the other one wasn't to prevent mismatch of the bits. However, until testing that part of the circuit we didn't realize that the logic would have to be different because otherwise there would be no way to Load A or Load B individually. We then realized that if one register was loading and the other was just holding the values, there would be no mismatched buts.



In our fixed design we have S1_B, S1_B and S1_A, S0_A instead of a shared S1, S0

Another issue was that our combinational logic for Q+ was not working when testing the circuit. With the combinational logic for Q+ being wrong it directly affected other combination logic for select bits and furthermore meant whatever was being controlled by those select bits. It wasn't until we went back to the circuit layout sheet that we didn't invert (Q+)' when using the 3 input NOR gate, and we accidentally connected that to where Q+ should have been connected.





Pin 12 of U2 was connected to Pin 2 of U6

Although we spotted these issues and attempted to correct them, we accidentally shorted a connection that caused one of the common rails (5V, Ground) to smoke and the wires connecting them to melt. Immediately after without changing any connections our circuit was behaving completely differently. The counter wasn't working and the registers weren't shifting properly, in addition to any other problem we didn't notice. We suspect that some component got damaged since even after not changing any connections we had different behavior, the problem was that it likely damaged at least one, if not two+ chips, and possibly could have damaged the breadboard. The amount of time it would have taken to unit test every single chip and possibly rebuild it on a different breadboard made us decide that we weren't going to be able to finish building it for demo. We are very confident about our design though, especially due to the modularity because we were able to test individual parts and could confirm they were working, and we feel we fixed all the issues that in an ideal world if we rewired everything perfectly it would have worked fine.

Discuss the design process of your state machine, what are the tradeoffs of a Mealy machine vs a Moore machine?

In our design process, we had to decide between using a Mealy machine vs a Moore machine.

The thing that separates the Mealy and Moore model is the way that the output is generated. For the Moore model, the output is a function is only based on the current state. For the Mealy model, the output is a function not only based on the current state, but also based on the inputs.

A Mealy machine immediately stood out as being the better solution for our requirement of being able to shift 4 times and stop, being able to have a function from current state and inputs would make it a lot easier to implement vs only being able to have a function from current state.

Another reason for using the Mealy machine was the accuracy of the outputs. In a Mealy machine, the outputs change on the rising edge of the clock. This allows for the logic to propagate through the circuit and stabilize by the next rising edge of the clock. In a Moore

machine, the output changes right as the logic is done and doesn't have to wait for the rising edge of a clock. However, due to different propagation delays of different parts of the circuit, some parts might change before others and would result in the output to have false values for a short period of time. That was something we did not want to risk when storing data in registers and performing shift operations, we can easily misplace or lose a bit.

By using a Mealy machine, we were able to synchronously design our circuit by using flip flops, counters, and shift registers that triggered on the rising edge of a clock.

7. Conclusion

In conclusion, while it was disappointing that we could not able to finish building our circuit and demo it, we learned a lot about the design process and know what we could have done better next time.

One of our pitfalls was that we prioritized finishing the easier parts of the circuit first, specifically the computation unit and routing unit. While we were able to get those done and working, it didn't really mean much in the grand scheme of things because the behavior of the circuit was crucially determined by the control logic.

We should have started with the control logic because it is more complex and harder to troubleshoot, and we could have honed in on fixing those issues earlier rather than running into them later with a bunch of other chips and wires connected. We accidentally ended up shorting a part of the circuit when we were building the control logic, right around when we were trying to get our counter to work. We suspect we connected some sort of pins on a chip together that ended up tying ground to VCC when we were trying to wire the configuration and enable pins.

Had we started with the control logic first and had this happen then, we only would had to rebuild the control logic part and not worry about other parts of the circuit being damaged.

Also, another reason for one of our wiring errors was because we started with the component layout first and not the logic/circuit diagram. With the component layout, keeping track of so many pins, it was easy to misread something and connect it to the wrong place. In our case, we didn't invert Q+' and connected that to where Q+ should have been. While Q+ looks very close to Q+', if we actually drew out the logic symbols we would have very clearly noticed the missing inverter gate when tracing the wires to where they were supposed to go.

Through our trials and errors, we not only learned how to go about the design process better, but we also learned more about the operation of the circuit based on how it would behave had certain inputs were wrong.