ECE 385

Fall 2020

Experiment 3

A Logic Processor

Zhicong Fan/Xin Jin
AB2/Online
Yucheng Liang

Introduction:

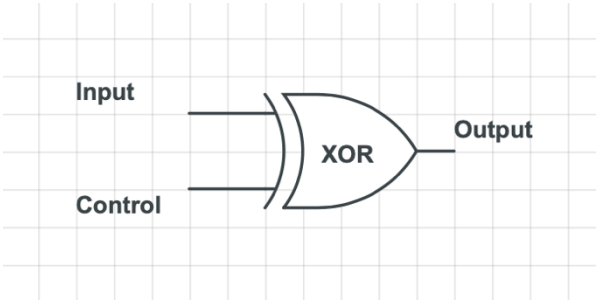
In this lab, we designed a 4-bit serial logic operation processor. There are 12 inputs in this circuit, data input (D0-D3) and control inputs (F0-F2, R0-R1, Load A/B, EXECUTE). With these 12 inputs, the circuit could achieve 8 different operations, and store the outputs in 4 different ways.

Prelab Questions:

Q: 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.

A: We could achieve this by using a single 2 input XOR gate. When control pin is 0, the output maintains the same value as the input. The output will be the invert of the input when the control pin is 1.

XOR		Input	
		0	1
Control	0	0	1
	1	1	0



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

A: A modular design divides the entire circuit into several relatively small parts. So that we could test units one by one. For example, in this lab, if the output from the Computation unit is correct, but the final output is wrong, then the error must be in the routing unit. If we design a modular design, we work on one part at a time which cuts down the development time comparing to implementing a random chip in an entire unit.

Operation of the logic processor:

Load Values into Register A and Register B:

Let's say we would store the value into A first and then B. First, we need to flip the switches (D3-D0) for the value we want to store in Register A. Then, flip Load A switch. After that, the data would be loaded into Register A. Then we just need to do the same operation again except that instead of flipping Load A, we would flip Load B.

Initiate Computation and Routing Operations:

After loading data into register A and B, we need to specify what function we need representing by flipping the switches F0-F2. For example, if we want an XOR operation, we would then flip F2 to 0, F1 to 1, and F0 to 0. Then, to specify the routing, we need to flip switches R0-R1. For example, if we want to store the output in register B and remain A unchanged, we would need to flip R0 to 1, and R1 to 0. Please refer to the tables below for other operations.

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

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

Description and Diagrams:

Description:

For the Register Unit:

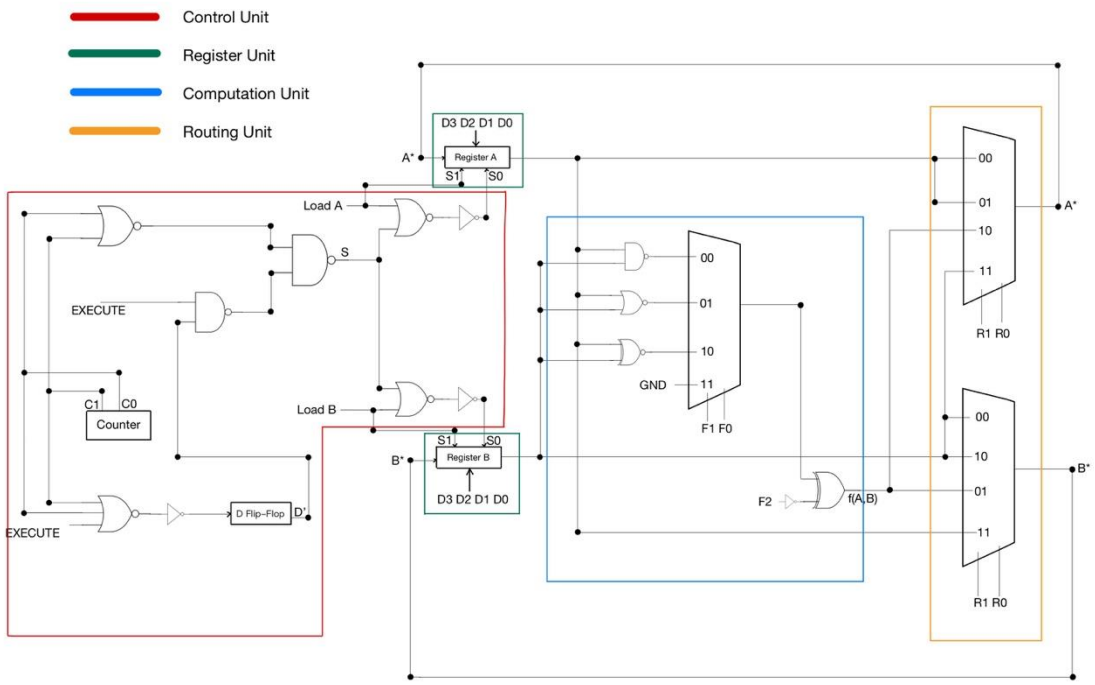
Inputs D0-D3 are directly wired to the parallel inputs of the Shift Registers to load input data into the registers. Then the output of the routing unit is wired in the Shift Right Serial Input pin to load the data routed by the routing unit. To control the mode of the register, S0 S1 calculated by the control unit are wired to pin S0 and pin S1 respectively to specify which register to load the parallel inputs and when to shift to shift right serial input mode.

For the Computation Unit:

In our circuit, NAND, NOR, XOR and several inverters are used to implement the 8 functions. From 00-11 in the 4-to-1 MUX controlled by F0-F1, we have: NAND, NOR, XNOR (XOR and an inverter) and a wire connected to the ground (0). We noticed that the other four functions were simply the invert of those four. So, we used F2 as the control and implemented a simple 2 to 1 invert circuit described in pre-lab. F2 goes through an inverter and we use an XOR to control the final output of the Computation Unit. For example, if we want an XNOR operation (F2-F0 = 110) on Registers A = 0 and Register B = 1, the 4-to-1 MUX would output 0. Then this 0 signal XOR with the inverted F2' = 0, would give us the final output 0 which is correct.

For the Routing Unit:
R1 and R0 are used in the routing unit. From 00-11, we have: A B, A F, F B, B A, which means whether we should store the result F, or the same value of either Register A or Register B to the registers. For example, if we choose, 01(A F) mode, Register A would still store the original value while the Register B would store the result computed by the computation unit.

For the Control Unit:
EXECUTE, Load A, Load B are the fundamental inputs of our logics, and we used a counter to help keep track of the current state. We will used C1 and C0 from the counter and as well as EXECUTE to determine the current state, whether it is in the middle of computation or it is held at the finish state, etc. The state is represented by an output Q, which will be passed to a D flip-flop. Q will also be the control pin of the counter to determine whether counter should keep counting for the computation or it should halt once a computation is finished. Moreover, we will use C1, C0, E, and D (output of D flip flop of Q) to produce output S, which will be used to control the register unit with Load A and Load B.



State Machine Diagram:

In this lab, we used Mealy machine instead of Moore Machine since it would contain less states. Here is the diagram:

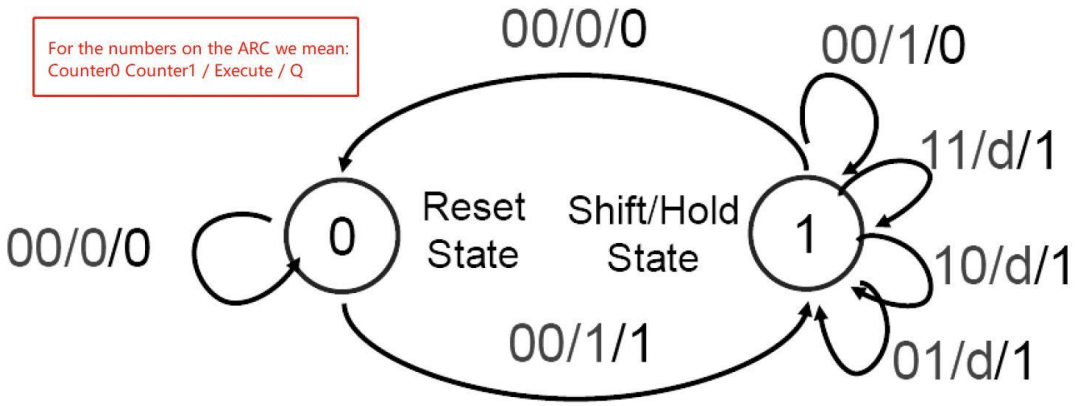


Figure 2: State Diagram

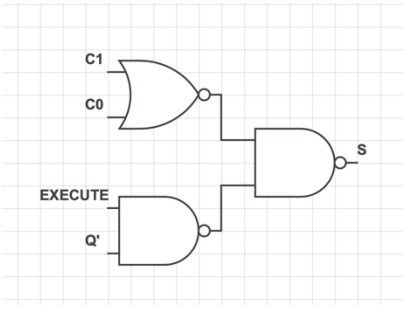
Design Steps:

Since the circuit involves the state Mealy state machine, we need to design the circuit very carefully in order to replicate the exact transition function. The transition of the states is given by the table below:

Exec. Switch ('E')	Q	C1	C0	Reg. Shift ('S')	Q^+	$C1^+$	$C0^+$
0	0	0	0	0	0	0	0
0	0	0	1	d	d	d	D
0	0	1	0	d	d	d	D
0	0	1	1	d	d	d	D
0	1	0	0	0	0	0	0
0	1	0	1	1	1	1	0
0	1	1	0	1	1	1	1
0	1	1	1	1	1	0	0
1	0	0	0	1	1	0	1
1	0	0	1	d	d	d	D
1	0	1	0	d	d	d	D
1	0	1	1	d	d	d	D
1	1	0	0	0	1	0	0
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	1	1	0	0

The transition function to the next state “Q+” and the output “S” to the register unit depend on input EXECUTE, current state Q, and as well as the counter C1 and C0. Based on the table above, we can create the K map below for output “S”. Notice that the output S is also the control pin to the counter.

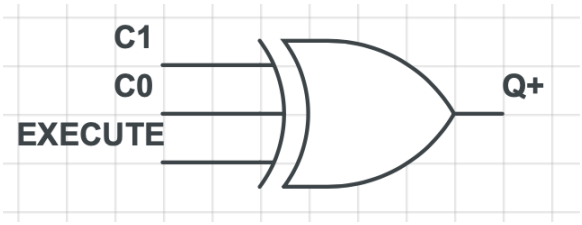
“S”	C1C0				
EQ		00	01	11	10
	00	0	d	d	d
	01	0	1	1	1
	11	0	1	1	1
	10	1	d	d	d



S = EQ’ + C1 + C0 = ((EQ’)’(C1+C0)’)’

Similarly, we can also create the K map for the next state Q+.

Q+	C1C0				
EQ		00	01	11	10
	00	0	d	d	d
	01	0	1	1	1
	11	1	1	1	1
	10	1	d	d	d

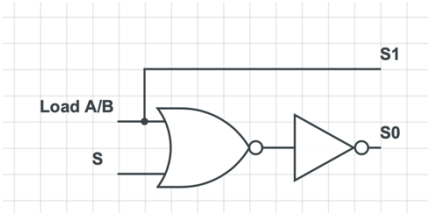


Q+ = E + C1 + C0

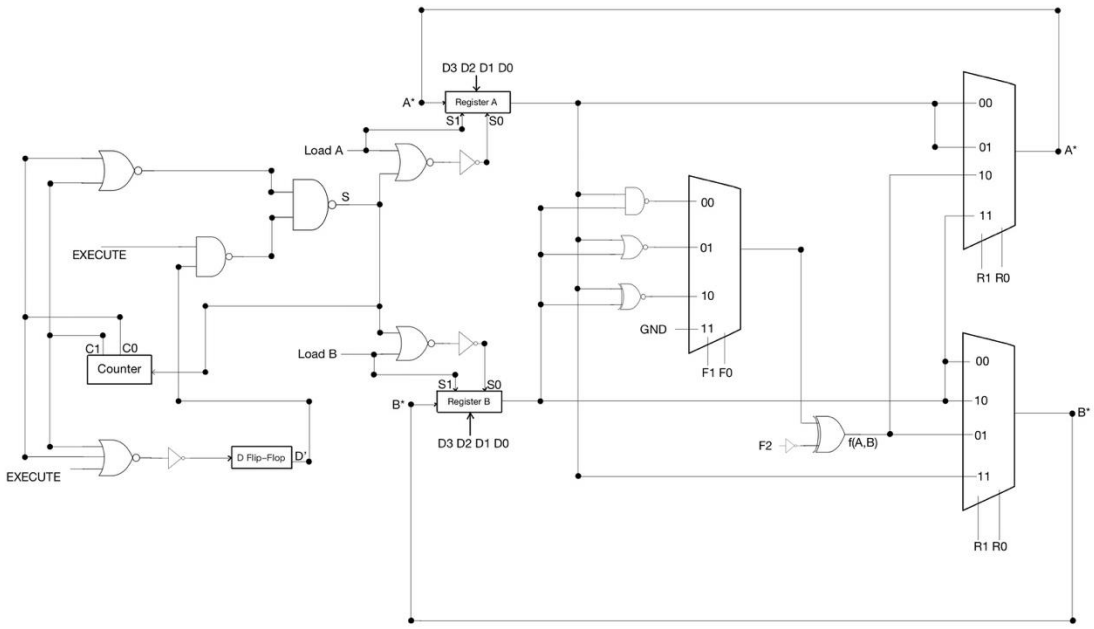
We also need to design the control to registers, which determines the mode of them.

S1	Load A/B		
S		0	1
	0	0	1
	1	0	1

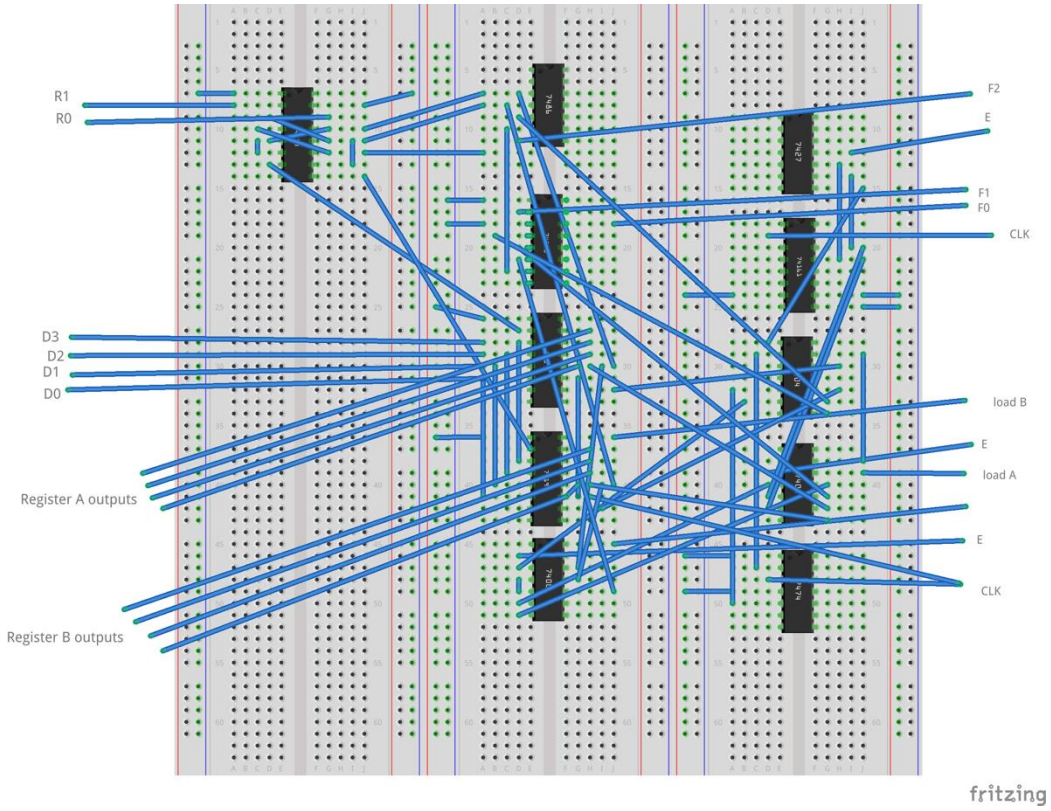
S0	Load A/B		
S		0	1
	0	0	1
	1	1	1



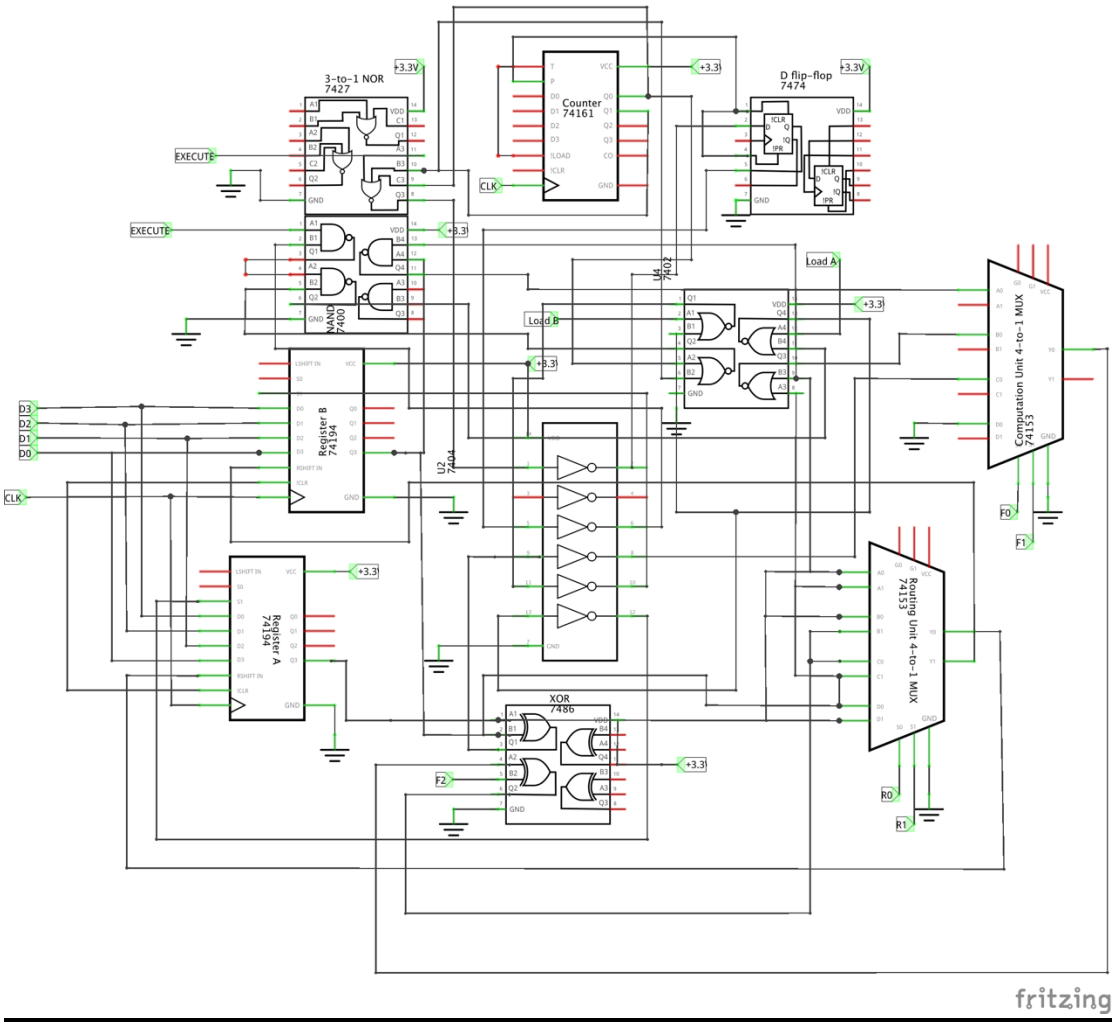
S1 = Load A/B. S0 = S + Load A/B



Breadboard View:



Notice the ground and the VDD are not connoted to the chip and bread board for the sake of clean looking.



Bugs Encountered:

1. We mistakenly apply the result of an XOR gate. The output of 00 is not 1 but a 0. After we change the logic, the computation unit gives us the correct output.
2. The control pin used to control the counter is hard to find. Instead of the output of Execute, Counter0 and Counter1 or the output of flip flop Q, the actual control pin is S the output of Q, execute, Counter0 and Counter1. We drew a truth table to help us decide the logic gates of S.
3. The inputs sequence of a 2-to-1 mux was reversed, instead of LOADA/B and then Ground, I connected the pin as Ground and then LOADA/B.

Conclusion:

In this lab, we implemented a 4-bit processor which can achieve 8 operations and store the outputs in 4 ways. The circuit is composed on four parts: control unit, computation unit, register unit, and routing unit. In addition, this lab used Mealy state machine to control the flow of the operations in control unit.

Post-Lab Question:

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

Since the output of the Moore state machine only depends on the current state, whereas the output of the Mealy state machine depends on both the current state and the inputs, the Moore state machine usually requires more states than Mealy state machine in order to achieve the same functionality. In result, the Moore state machine would require more hardware and react slower compare to the Mealy state machine. But in return, the Moore state machine is able to provide synchronous output while the Mealy state machine cannot. Also, the Moore machine is easier to design whereas the Mealy state machine would need more sophisticated considerations.