ECE 385
Lab 3
A Logic Processor

Kang, Xingjian
Li, Ziying

March 6, 2024

1 Introduction

1.1 high-level function

This is a bit-serial logic processor that can switch to 8 functions: AND, OR, XOR, NAND, NOR, XNOR, CLR, SET, and SWAP by 4-bit shift registers, several multiplexers, and some type of counter.

1.2 pre-lab questions

A. xor gate that takes the outputs of the logic gates and not signal as inputs.

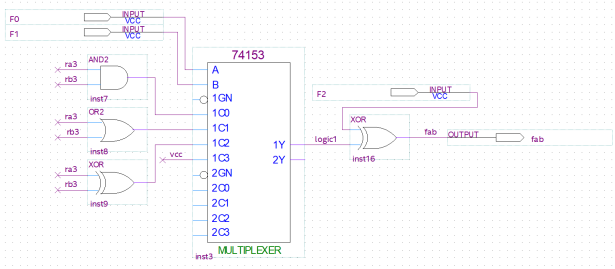


Figure 1: Enter Caption

B. We can connect inputs and outputs to only one modular and debug, which is much easier to fix a bug in the whole big system. We can examine limited components to find the bug faster.

2 Operation of the logic processor

2.1 Describe the sequence of switches the user must flip to load data into the A and B registers

Keep "EXECUTE" at 0. Use "D3,D2,D1,D0" to input the binary number, then flip LOAD A or LOAD B to load the number into Register A or B.

2.2 Describe the sequence of switches the user must flip to initiate a computation and routing operation

Keep "EXECUTE" at 0, flip "F2,F1,F0" for logic and "R1,R0" for routing, then flip "EXECUTE" to 1 for a short time and then flip it back to 0. After 4 shifts, the result will be in register A and B.

3 Written description, block diagram and state machine diagram of logic processor

3.1 Written description

3.1.1 Control Unit

This unit is to decide when to tell the shift registers to shift right. This unit has three input signals and a clock signal that aids in shift register synchronization, which are identified as Load A, Load B, and Execution. Only when the Execution signal goes from 0 to 1 will the control unit and the entire circuit come to life. Additionally, the Load A and Load B will decide which register should load the data D3–D0 from the register unit's input port and if the shift unit should load it at all. The output of the control unit will act as select signals for teh register unit.

3.1.2 Register Unit

This unit consists of two 4-bit shift registers. D3–D0, a 4-bit signal, will be entered into these registers. Other input signals will be from the routing unit. The control unit will produce two select signals, S0 and S1, which determine whether shift registers should shift the data or retain it. The rightmost data bits from registers A and B will be loaded into the computing unit.

3.1.3 Computation Unit

This unit receives the data and computes the results depending on the operation selected by the given signals F2-F0. Based on the selection inputs, the calculation unit will select one operation out of a total of eight to perform and compute the results, F, with A and B. The outputs, A, B, and F, will be the routing unit's inputs.

3.1.4 Routing Unit

This unit is to determine which two of the three inputs from the computation unit should be put back into the register unit. The routing unit will accept the data from the computation unit, and a unique signal consisting of two bits, R1 and R0. The output of the routing unit will be loaded back to the register unit.

3.2 high-level block diagram

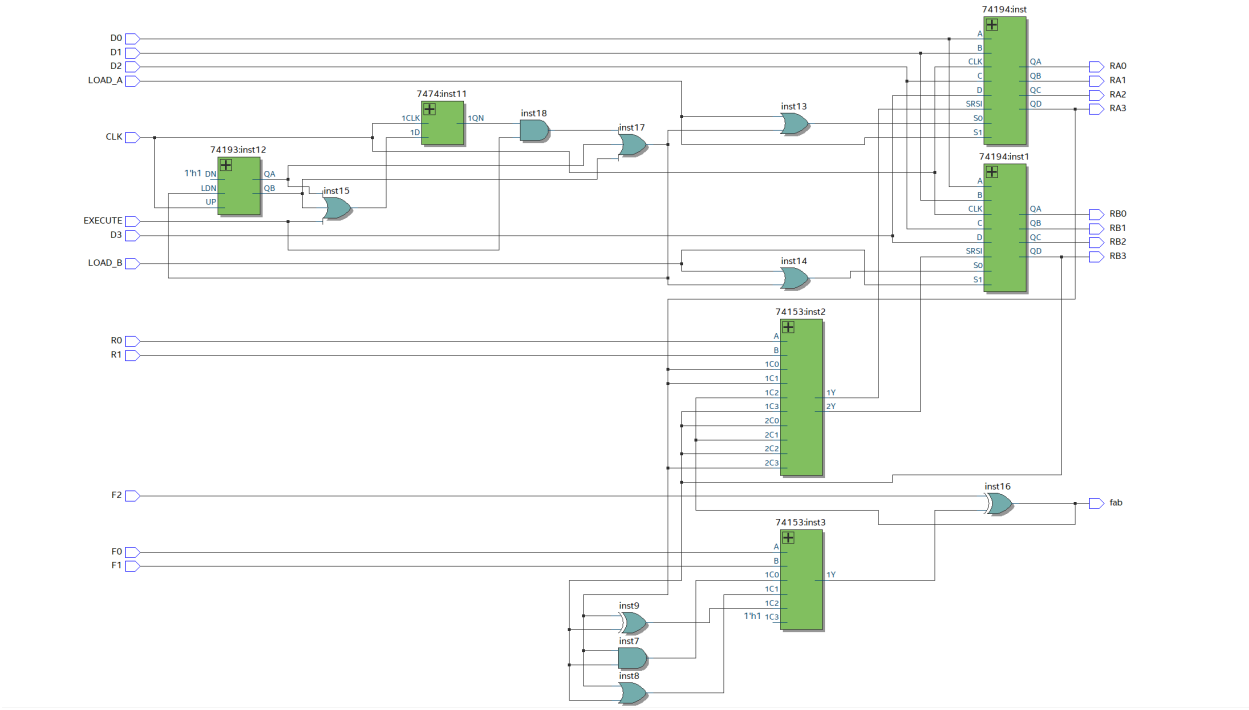


Figure 2: RTL diagram

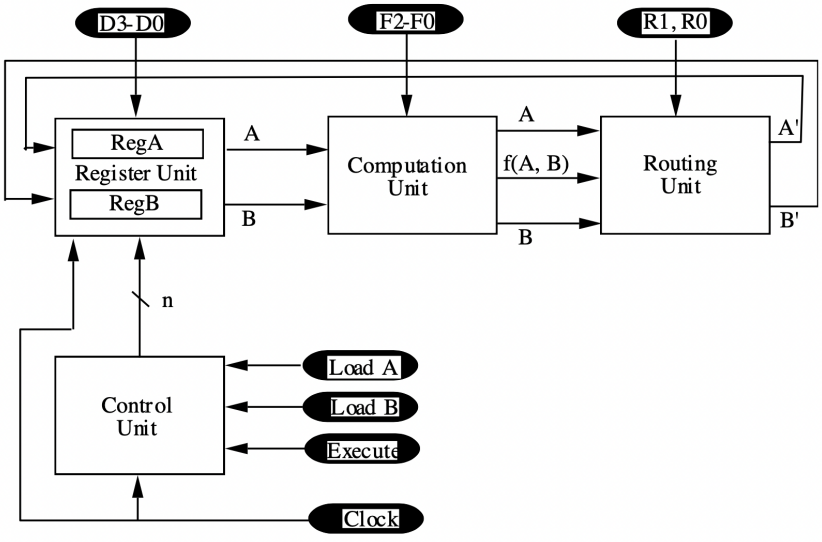


Figure 3: Block diagram

3.3 State Machine Diagram

i. The type of machine we use

We used a Mealy machine

$Q^+ = C_0 + C_1 + E$

$S(shift) = C_0 + C_1 + E\overline{Q}$
$count = E + S$

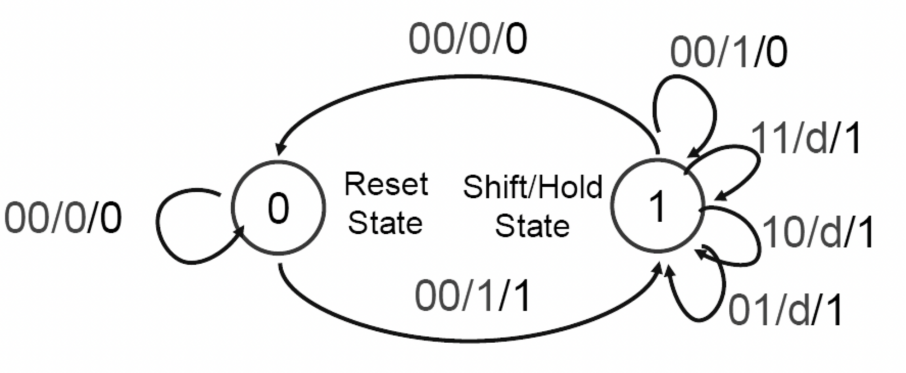


Figure 4: State Machine diagram

ii. Label each state

For "Reset" state, it simply wants for the triggering of "Shift/Hold" state. For "Shift/Hold" state, it allows for 4 times of shifting in serial, and it would stop shifting after 4 times of shifting have been done whenever Execute bit is still 1. If Execute bit is still 1, it will remain at "Shift/Hold" state; otherwise, it will switch to "Reset" state.

iii. Label each arc

The values on the arcs represent count/execute/shift. "count" is responsible for monitoring how many times it should shift; "execute" is responsible for whether to start a new round of shifting or not; "shift" is responsible for whether to perform (right) shifting a bit or not.

4 Design steps taken and detailed circuit schematic diagram

4.1 Written procedure of the design steps taken

4.1.1 K-maps

EQ \ C1C0	00	01	11	10
00	0	X	X	X
01	0	1	1	1
11	0	1	1	1
10	1	X	X	X

(a) K-map for S

EQ \ C1C0	00	01	11	10
00	0	X	X	X
01	0	1	1	1
11	1	1	1	1
10	1	X	X	X

(b) K-map for Q+

EQ \ C1C0	00	01	11	10
00	0	X	X	X
01	0	1	0	1
11	0	1	0	1
10	0	X	X	X

(c) K-map for C1+

EQ \ C1C0	00	01	11	10
00	0	X	X	X
01	0	0	0	1
11	0	0	0	1
10	1	X	X	X

(d) K-map for C0+

4.1.2 Truth table

TABLE 1: Control unit state transition table using the Mealy state machine

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

Figure 5: Truth table for control unit

4.1.3 Written description of the design considerations taken (did you consider multiple implementations of the same circuit and the tradeoffs of each?)

For the computation unit, I considered using a multiplexer for the not mode and then optimized to an xor gate. For the routing unit, I considered two separate circuits for registers A and B, but then found the two could be combined together.

4.2 Detailed Circuit Schematic

4.2.1 A gate level schematic of our circuit

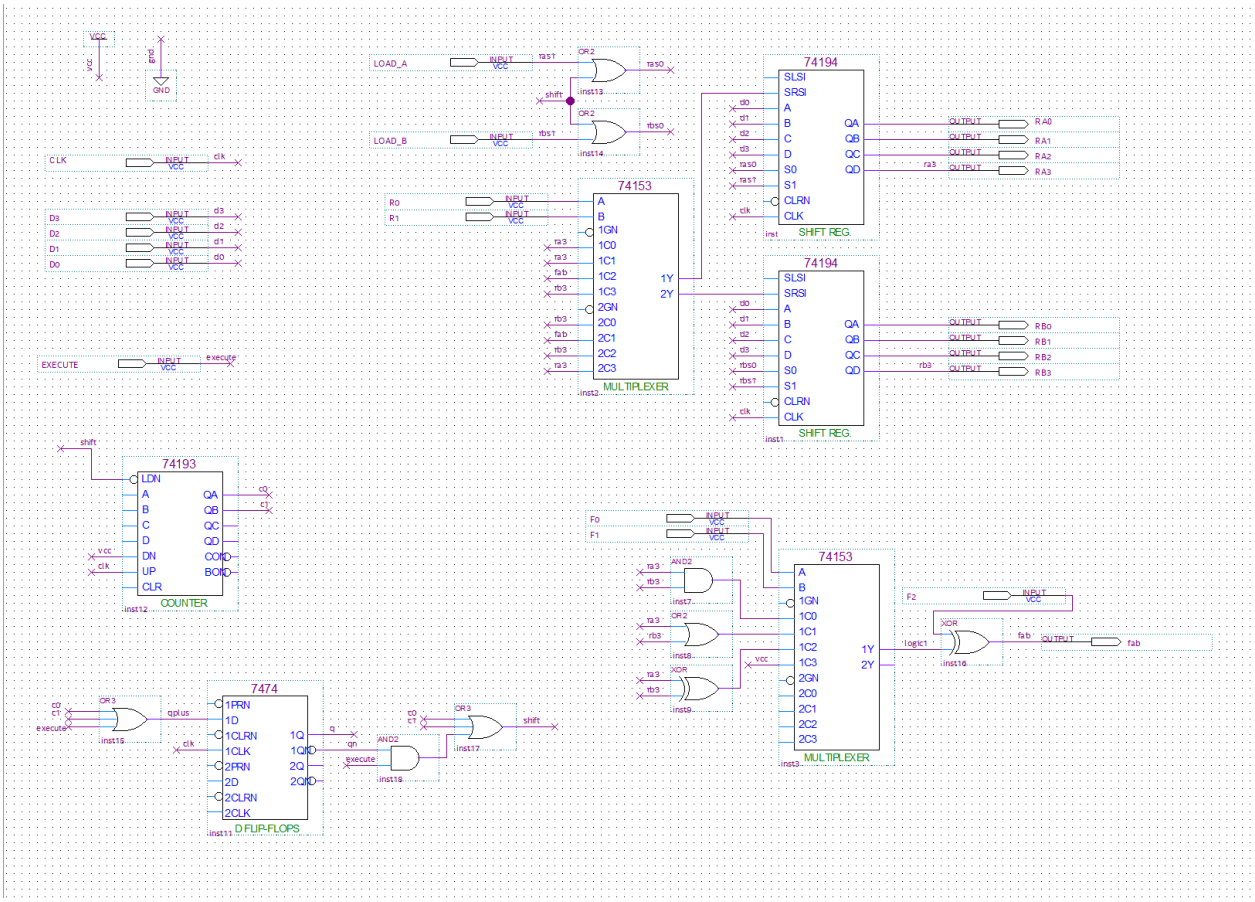
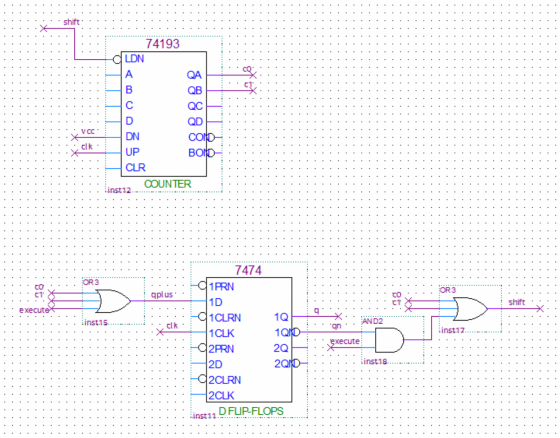
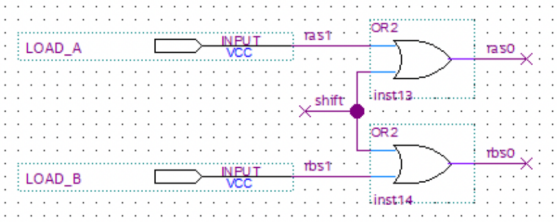


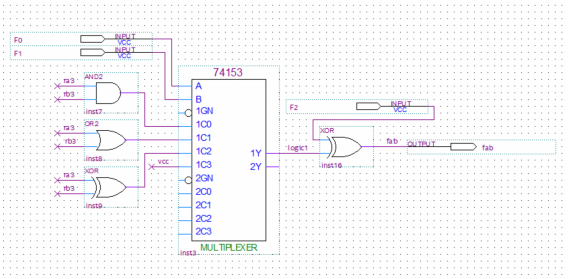
Figure 6: Enter Caption



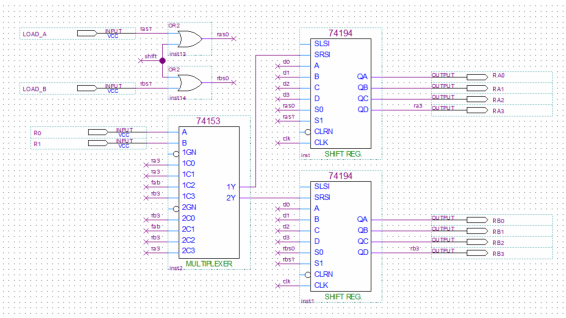
(a) Control Unit (part 1)



(b) Control Unit (part 2)



(c) Computation Unit



(d) Routing and Shift Unit

1. Inputs: D3, D2, D1, D0, F2, F1, F0, R1, R0, LOAD_A, LOAD_B, EXECUTE, CLK
2. Outputs: RA3, RA2, RA1, RA0, RB3, RB2, RB1, RB0
3. Intermediate logic: c1, c0, q, qplus, shift, ras0, ras1, rbs0, rbs1
4. Mode pins: vcc, gnd

5 Description of all bugs encountered, and corrective measures taken

Bug 1: The contents in shift registers A and B did not appear as expected. measure: find that the multiplexer for the routing unit is connected badly, check the table for R1, R0, reconnect it

6 Answer to all post-lab questions

6.1 Document changes to our design and correct our Pre-Lab write-up, explaining any difficulties you had in debugging our circuit.

Change 1: At first I just used a multiplexer for changing the logic to not mode, when I reviewed the class, I found an optimized design with xor gate and used it. Difficulty 1: I did not notice the sentence "After the four shifts, the state machine will state in (SQ+C1+C0+='0100') if the 'Execute' switch remains high,", so I felt confused about whether should I execute multiple times. Difficulty 2: I did not know the order of input and output for registers. Whether it should be 0123 or 3210.

6.2 How modular approach proposed in the pre-lab help our isolate design and wiring faults

Three separate modules that can be developed independently of one another without compromising the functionality of the others are the control unit, computation unit, and routing unit. We first begined the wiring of the control unit, and we verified the output of the unit to be correct. Similarly, we finished the rest two modules and tested them individually. After finishing these three modules, we wired them all together and tested the whole system. Debugging for the three relatively isolated modules reduces testing time when compared to building the entire circuit and discovering wiring errors.

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

The Moore Machine and the Mealy Machine are essentially the two methods used to construct an FSM. Initially we planned to use a Moore machine with three states, but this would have resulted in a state machine with 24 entries in the truth table, so we thought about using two states representing "Shift" and "Halt". We thought of combining the two states representing "Shift" and "Halt" into a single state and using the Mealy machine, which would reduce the number of entries in the state machine's truth table to 16. It seems that the Moore machine model is more straightforward to comprehend and easier for us to develop from the ground up. However, by examining the logic of this Bit-Serial Logic Processor, the design would be much simpler with a Meal machine.

7 Conclusion

In this lab, we build a Bit-Serial Logic Processor. Prior to constructing the processor, it is necessary to design the following four components: the Control Unit, Shift Register Unit, Computation Unit, and Routing Unit. The input to the processor will initially enter the Control Unit, and the external data supplied to the processor will be stored in the Shift Register Unit before being input to the Computation Unit and returned to the Shift Register Unit via the Routing Unit.

8 Our simulation results: waveform generated by the standard testing input, and should be captured in every 400ns