# ECE 385

Fall 2020

Experiment # 3

# A Logic Processor

Lian Xinyu & Liu Tianyu

TA: Li Shuren

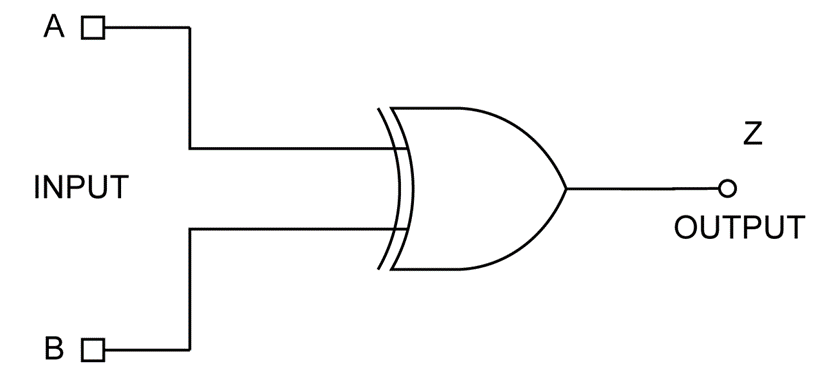
**Introduction**

In this lab we made a simple logic process which can finish some bitwise operations for two 4-bit length number strings. Moreover, we have achieved eight different logic functions for input and designed four routing paths for data process. We also built a control logic to output correct result. In the lab we used counters, multiplexers, and various logic gates to stimulate the real work. The purpose of this lab is to let us get familiar with how a computer and CPU work. The version of our software Quartus is v18.1.

**Prelab**

1. The circuit is (an XOR gate):

In our design, A is the input signal and B is the control signal. That is, if B is 0, Z will output the original value of A; if B is 1, Z will output the invert value of A. Taking A=1 as one example. If A=1, B=0, then we will have Z=A XOR B=1. If A=1, B=1, then we will have Z=A XOR B=0.



1. Answer:

If we use more complex chips, such as flip-flop or MUX to finish this task, we will face a higher risk to meet errors in the whole circuits. In real work, we always need to build a large circuit to fulfill the targets. In this process, we normally divide the circuit design into several modules, such as control module, running module, test module, etc. That is because if we find errors in output for a large circuit, this fact will be not easy for us to find the sources of bugs. However, using modular design, we can test separately for every different module. If all of our modules pass our test and give right output, the probability of correct output for the whole circuit after combining will be very high. And if the design is a group work, modular design can allow us to divide the whole circuit to several part. Everyone can hold one part separately. To save time and increase the productivity, we usually choose modular design.

1. The design of our circuit is attached at the end of this paper.

**Operations of logic processor**

1. Load registers

We need to load data into two registers. First of all, we can use D0-D3 as the data input terminal by adopting the parallel loading mode of 74LS194 chip. For the different string A and B, they are corresponding to the input D0-D3 of different chips, so we need to switch D0-D3 for loading. Meanwhile, we notice that it is possible to use control signal to determine which register I intend to input data. For example, if I intend to load data to register A, we need to switch signal LOAD\_A. That depends on our demand.

1. Initiate a computation and routing operation.

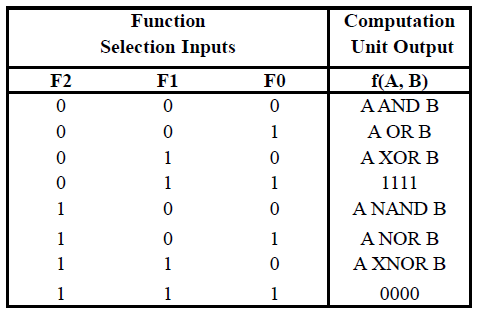
The sequence begins with loading data into the registers. And then, we need to switch F0-F2 to determine one operation result. We also need to switch R0-R1 to determine the way routing back to the registers. Finally, we will set the EXCUTE signal as logic 1 so that simulation process will start.

**Description of circuit**

1. Written description

The first part is for the **register unit**. Register unit is composed of two registers (we use chip 74LS194 in this lab). This unit can load data A and B in parallel and update the content of A and B. For the implement details, every 74LS194 chip has D0-D3 used for loading data, S0 and S1 used for determining the carry type, SRS for loading the data and providing a right shift direction and CLK input for clock signal. After the corresponding input signal is connected, the registers are ready to work.

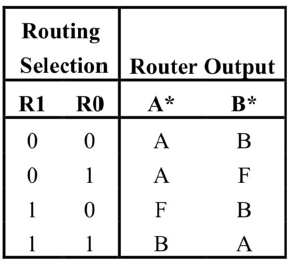
The second part is **computation unit**, this part is known for evaluating f(A, B). This part receive input from the register output signal Q\_D parts (which means the bit that need to be computing of A and B). Depending on the input F0-F2 signal content, the functions performed by the Computing Unit were also different. To implement computing Unit, we used an 8-to-1 MUX (74153 chip). At the same time, we design the implement for computing functions using different gates (OR, XOR, NAND, etc.). Moreover, due to the limitation of specifying the chip list, for some functions, such as for an OR gate, we need to concatenate a NOT gate with a NOR gate to implement. Here are the details of function selection input signals and computing unit output signals for this computing unit (we can get similar content in course materials):



This is the choice of chips for this part:

|  |  |
| --- | --- |
| **IC Description** | **Type** |
| **1-1 NOT** | **7404** |
| **2-1 NAND** | **7400** |
| **2-1 NOR** | **7427** |
| **2-1 XOR** | **7486** |
| **3-1 NAND** | **7410** |
| **3-1 NOR** | **7427** |

The third part is **routing unit**. In routing unit part, we need to pass the result of computing unit output back to the register according to the requirement of the list shown below. In addition, we have signals R1 and R0 to determine the output of routing unit. This unit performs one cycle for a pair of A and B to complete the processing. When a pair A and B finishes the logic processing, the routing unit waits for the signal to be ready for the next computing task.



The final part is **control logic unit**. We believe this part is relatively difficult for us to design. We have LOAD\_A and LOAD\_B to determine the loading of data and EXCUATE signal used to tell the control unit that the select switches and the register contents are ready for execution and that the control unit should begin the computation cycle signal, and then this unit will shift the register unit the required number of times and halts until the next execution is requested. Obviously, we need a mechanism to track the control cycle, which is known as the state machine. In addition to some logic gates, we also need to combine them with flip-flops (7474 chip) to implement the state machine design (i.e. the control logic unit).

1. High level graph

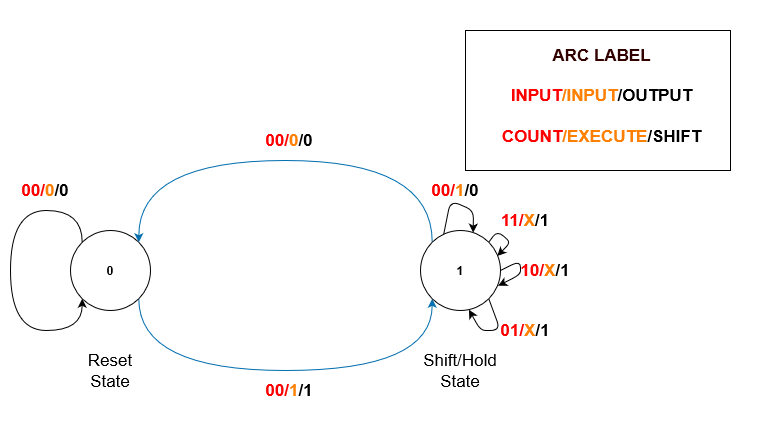
The high level schematic for our design is shown below, and the key factors have been highlighted.

图片包含 日程表

描述已自动生成

1. State machine

We choose Mealy machine to perform as the control unit. The transform graph of different states can be shown as:



From this graph, we name two states as ‘Reset State’ and ‘Shift/Hold State’. The input needed to transfer the state is listed on the diagram. In addition, it is worth noting that the Mealy machine can work for shifting of registers in four clock cycles.

**Design steps taken and detailed circuit schematic diagram**

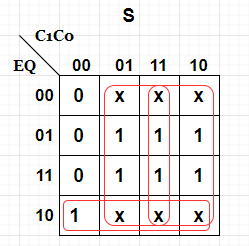
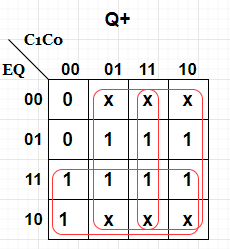
1. Control logic is the most difficult part, so we start to design it at first. To implement the functions of Mealy machine in circuit level, we first need to change the truth table of E-Q/C1/C0/S/Q+/C1+/C0+ into different K-maps so that we can get the logic expression for the updating of signals. (Note: E means EXECUTE, S means register shift. For **this cycle**, C1 and C0 are the output of the counter, and Q is the output of the flip-flop. For **next cycle**, C1+/C0+ are the output of the counter, and Q+ are the output of the flip-flop.). In addition, we can use C1/C0/ C1+/C0+ to build the counter, according to the transform graph for the truth table.

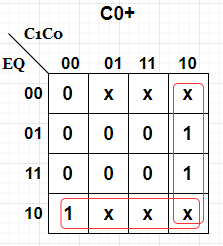
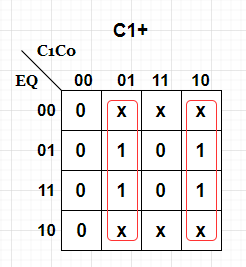
The truth table is:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **E** | **Q** | **C1** | **C0** | **S** | **Q+** | **C1+** | **C0+** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | X | X | X | X |
| 0 | 0 | 1 | 0 | X | X | X | X |
| 0 | 0 | 1 | 1 | X | X | X | X |
| 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 | X | X | X | X |
| 1 | 0 | 1 | 0 | X | X | X | X |
| 1 | 0 | 1 | 1 | X | X | X | X |
| 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 |

Note that we need to use K-maps to derive the logic expression for S/Q+/C1+/C0+, and C1/C0/C1+/C0+ can be used to build a counter using logic circuit. Here are the details.

The K-maps are:





For the logic expression of Q+, it is: .

For the logic expression of S, it is.

For the logic expression of C1+, .

For the logic expression of C0+, .

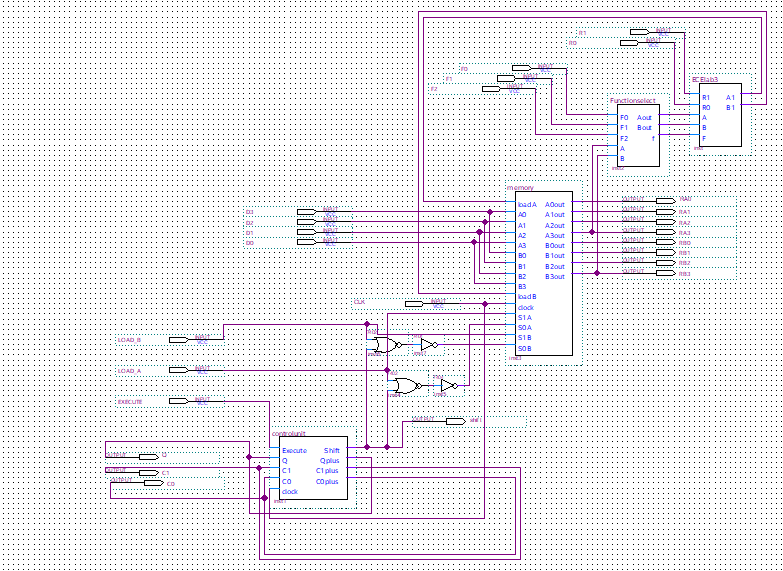
After we find the logic expression for S (same as LD for the counter) and Q+**,** we can continue building the circuit.

1. Secondly, we need to design the register unit. For the design of register unit, we need to connect S from the control unit to the input port S0 and connect LOAD signal to input port S1. These two ports decide whether to shift the register. We also need a CLK signal to work as clock signal. For the loading of data (A and B), we can use D0-D3 as parallel input. We also need to set the SRL port of register with the result of the computing unit part. After we connect all the ports, the design of the register unit is completed.
2. For the computing unit, we can use a 8-to-1 MUX to receive F0-F2 as control signal. In addition, for the data input part, we will compute the output from the register unit at first, then send them to the MUX. The details of computing unit are described in the former table. After we connect all the wires, our design is completed.
3. Finally, for the routing unit, we need two 4-to-1 MUXs to finish the design. For each MUX, it needs to receive input from the computing unit and register unit. Moreover, we have F0-F1 working as control signal to determine the output of routing unit. The output of routing unit should connect with the input port (SRL) of two registers. The details of routing unit are described in the former table. After we connect all the wires, the whole circuit is completed.

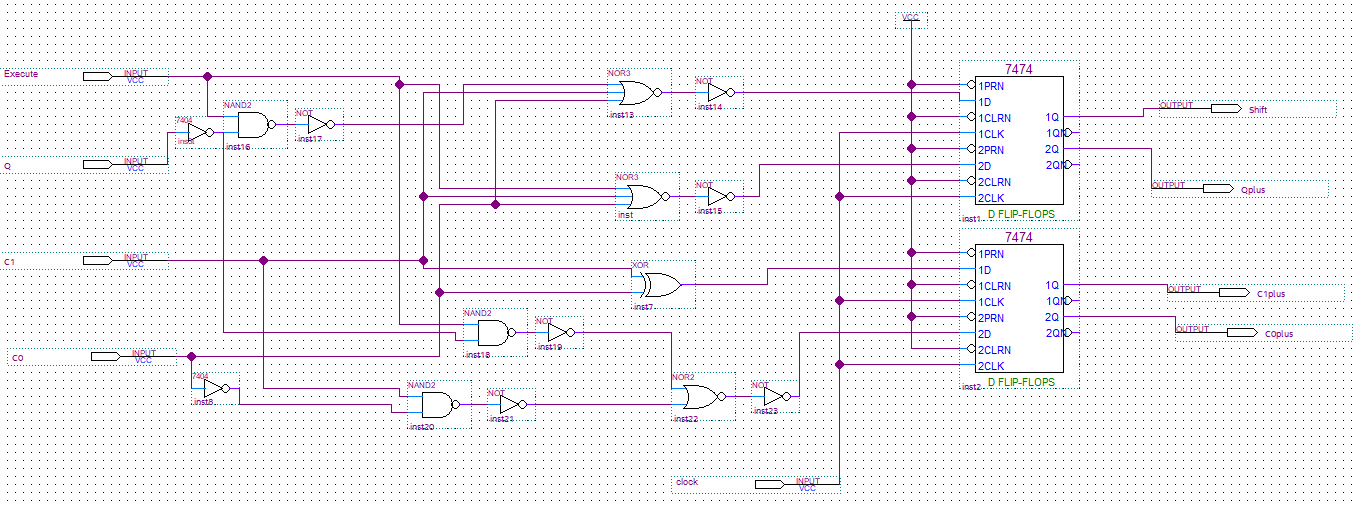
**Circuit Schematic**

Here is the circuit schematic for our design.

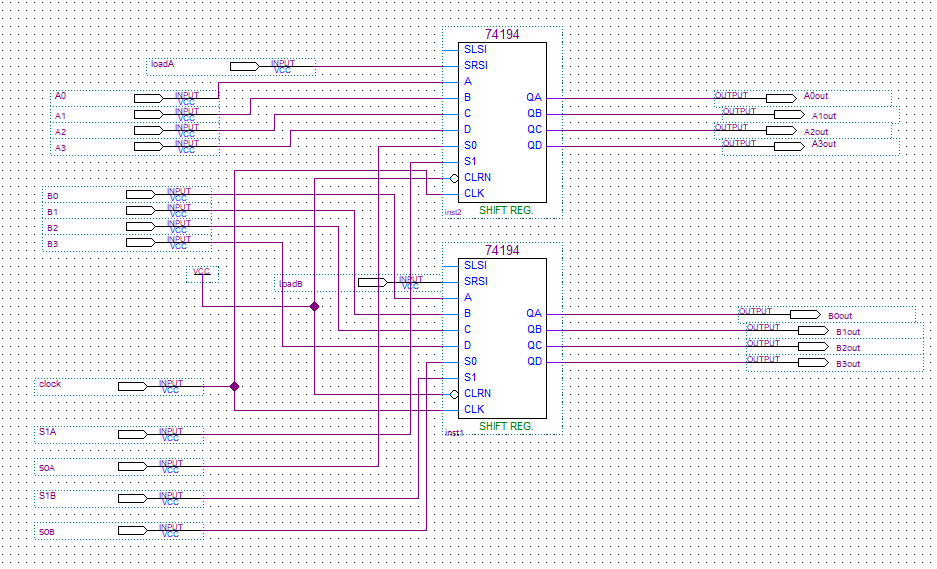
whole circuit:



control unit:



memory part:

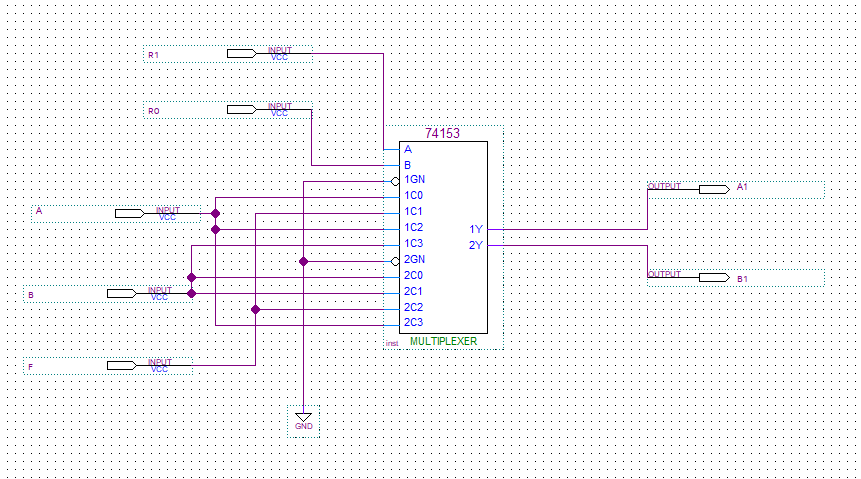


computation part:

图示, 示意图

描述已自动生成

routing part:



**Post-lab**

1. Bugs & Debugs

Thanks to Pro. Li, we learnt a lot in the lectures and we can finish the lab smoothly. In this lab, we only meet one bug that when we connect the line in Quartus, we combine one line with two other lines, which will cause the error that one signal will have two different input. After we delete the wrong line, our circuit worked well.

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

We choose Mealy machine in this lab not only because of the truth table for the control unit provided for us is written in the form of Mealy machine, but we notice that Mealy machine has an advantage that Moore machine cannot match: fewer states for the same level control requirement.

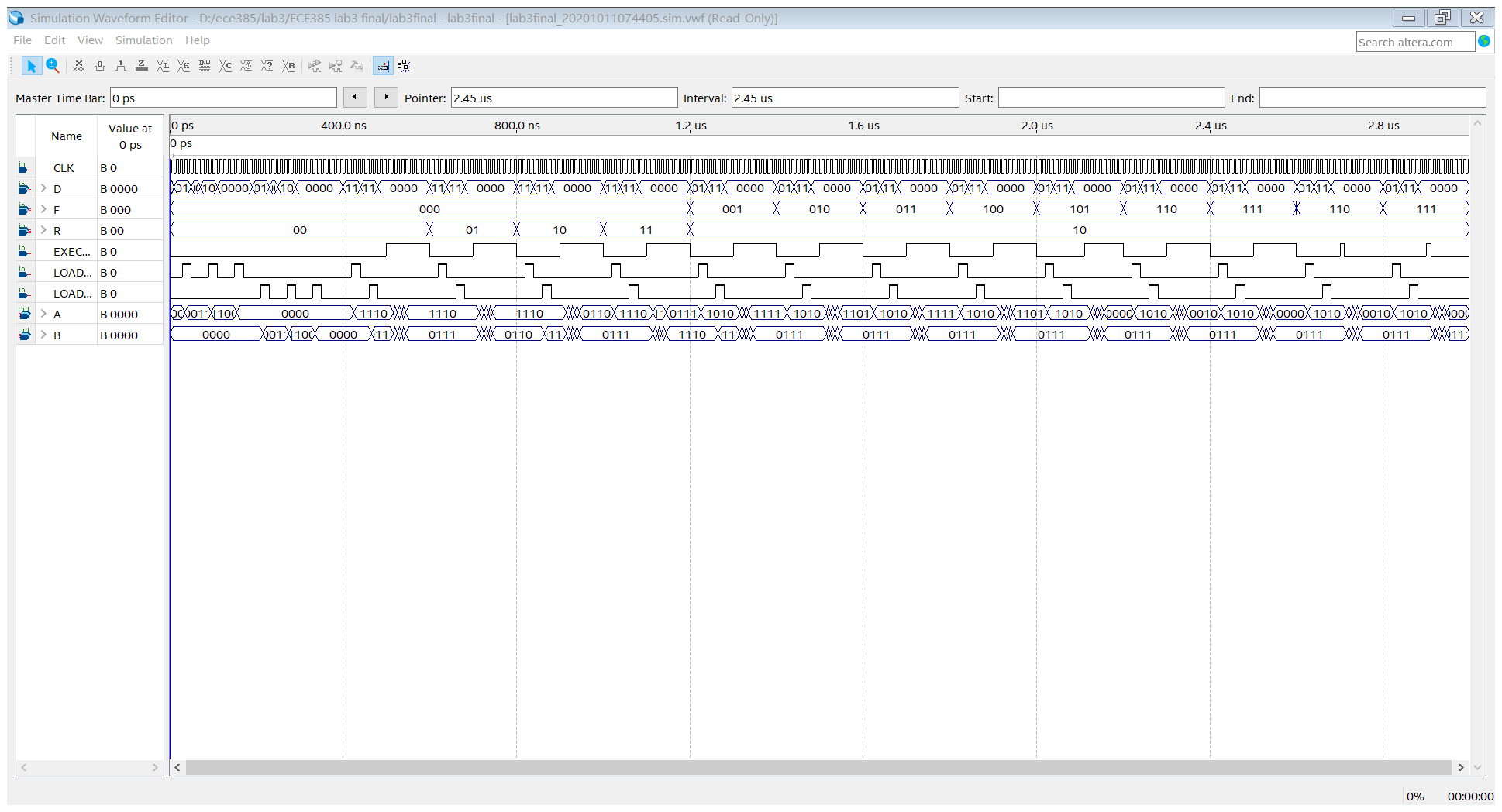
Since the outputs of the Mealy machine depends on a combination of the current state and the current inputs, we can use fewer states for the implement of Mealy machine than the Moore machine, so using Mealy machine will simplify the structure of our circuit.

Nevertheless, sometimes Moore machine will perform better than Mealy. Firstly, the architecture of Moore machine is easier for us to understand. In addition, Moore machine provides us more states so that we will spend less time to find bugs in our circuit because of more test choices. According to one blog [1], the output of Moore machine is synchronized with the clock, and the jitter of the output signal can be eliminated to a certain extent when Moore machine receives complex input signals.

**Conclusion**

In this experiment, we designed a bit-serial logic processor, which can perform eight different logical computation on two 4-digit binary numbers. We divide the whole circuit to several parts, register unit using two 4-bit shift registers, computation unit using combinational logic, routing unit using two 4-1 MUXs and control unit using mealy machines. Through this lab, we learned the importance of having a modular design especially when creating larger circuits. We also had a refresher on Mealy and Moore machines through this lab, which will help us in future labs when we have to design more state machines in System Verilog language.

**Simulation result**

****

Reference

[1] https://blog.csdn.net/a931103123/article/details/41625707