ECE 385 Lab Report

Spring 2023

Lab 3: A Logic Processor

Name: Zhu Hanggang, Hong Jiadong

Student ID: 3200110457, 3200110970

Prof. Chushan Li, Prof. Zuofu Cheng
ZJU-UIUC Institute
April 6, 2023

1 Introduction

In this lab, we build a simple bit-serial logic operation processor that allows eight basic operations like AND,OR, XOR,NAND, etc. It supports operations on two 4-bit numbers and routes the results of the operations in four ways. We also build a finite-state machine to serve as the control unit of the processor.

1.1 Prelab Question A

What we need to do is simply use a 2 to 1 MUX and a NOT gate to implement the logic.

Suppose A determines the output inverts B or not, A = 0 means invert B, A = 1 means not invert B.

A possible implementation is, for the input of MUX, we connect NOT gate and B to the first position, and connect B to the second position.

Here is the implementation sample:

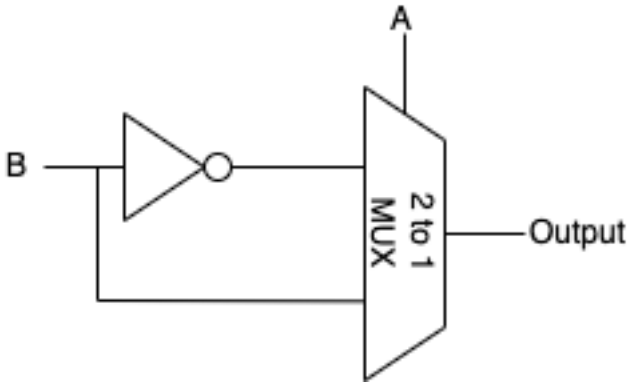


Figure 1: Sample Circuit

1.2 Prelab Question B

The unit test of a partition of the circuit is easier than the overall circuit. As long as you have unit tested the functional circuit blocks separately, we only need to test whether the combinations and the sequence of circuit blocks are right or not. Massing up everything will lead to ghost bugs which are hard to find in the massive design. The modular design is also an easy-to-hard approach that is quite useful in development.

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.

If we what to load the data into Register A or B, we should let Shift signal in the Mealy Machine maintains the low level.

To Load the data D3-D0 into a specific Register, we should flip the corresponding Load switch of the register we want to load into to a high level. For example, if we what to load D3-D0 into Register A, we should set Load A to high and keep E at low, then the data D3-D0 would be loaded into register A, the process of loading B is the same but to change the Load A switch to the Load B switch. Also, we can set both Load A and B to high if we what to load D3-D0 to both Register A and B at the same clock circle.

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

For the computation part, we should flip Execute switch to indicate that the processor should execute the computation, then the computation and the register shifting would be controlled by the Finite State Machine of the Control Unit, which is shown in figure 2.

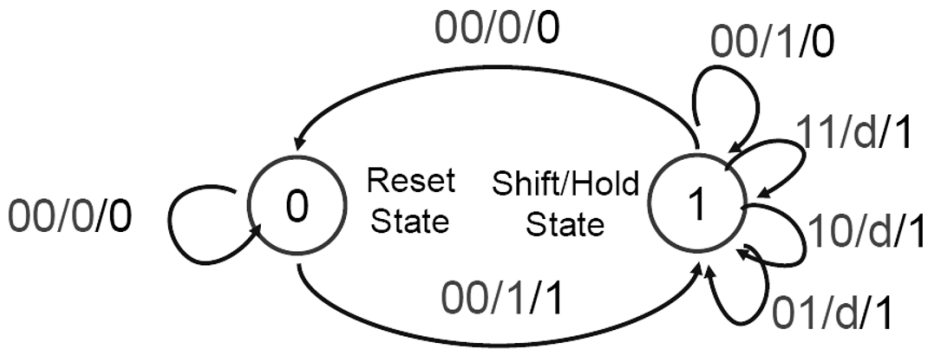


Figure 2: FSM From Manual

Another important control of the computation is the computation type selection controlled by F2-F0. We can choose 8 different types of computations. The detailed modes are shown in table 3a 3b

Basically, for the routing processing, we should flip the Routing Switches R_0 and R_1 to specific signal levels to achieve certain routing requirements. The truth table is shown as follows, F indicates the calculation output of the computation block.

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

3.1 Written Description

In this system, we can use Control Unit to generally control timings to execute the calculation and whether to shift or load data in the shift registers. In the computation block, we use a simple operation code F2-F0 to control different calculation types of the Calculation Unit. In the Routing Unit, we also use simple operation code R1-R0 to control the different routing methods.

Function Selection			Function Output
F_2	F_1	F_0	$F(A,B)$
0	0	0	$A \cdot B$
0	0	1	$A + B$
0	1	0	$A \otimes B$
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

(a) Function Table

Routing Selection		Router Output	
R_1	R_0	A^*	B^*
0	0	A	B
0	1	A	F
1	0	F	B
1	1	B	A

(b) Routing Table

Figure 3: Function and Routing

3.2 High-level Block Diagram

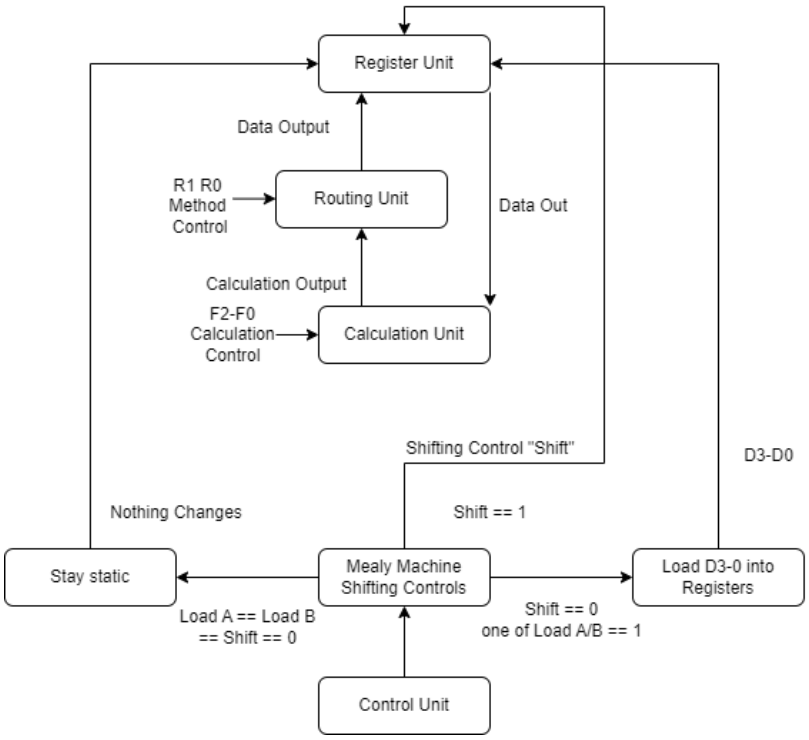


Figure 4: High-level Block Diagram

3.3 State Machine Diagram

We use Mealy Machine implementation in this unit. The reason for our choice is explained in Postlab questions. The State diagram of our design is shown as follows. Since diagram of Mealy machine is already given in lab manual, we slightly modify the Finite State Machine forms into more precise diagrams in the real clock circle behavior of the circuit. The state in the following graph is QC_1C_0, the input is E and the output is S. The diagram is shown in the figure5.

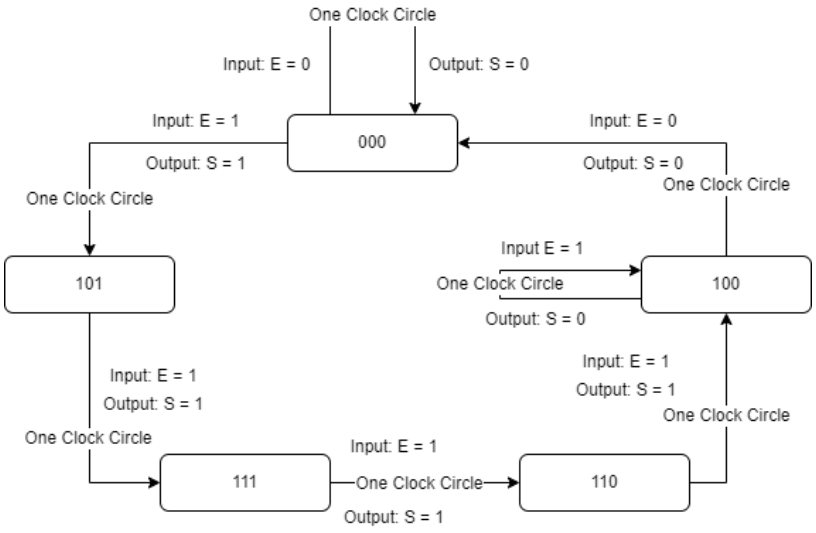


Figure 5: Mealy Machine State Diagram

4 Design steps taken and detailed circuit schematic diagram

4.1 Written procedure of the design steps taken

First, we implemented the calculation module and Shift Register. After using simple logic to test the calculation module and Shift Register, we designed and implemented the Control Unit for the control interface between the Shift Register and the calculation module.

The difficulty in implementing Control Unit is the implementation of FSM. First, we drew the Karnaugh map according to the Manual and designed the logic gate circuit. The K-maps we drew based on the lab manual are shown in figure 6 and figure 7

C_1C_0 \ EQ	00	01	11	10
00	0	0	0	1
01	×	1	1	×
11	×	1	1	×
10	×	1	1	×

(a) K-map for S

C_1C_0 \ EQ	00	01	11	10
00	0	0	1	1
01	×	1	1	×
11	×	1	1	×
10	×	1	1	×

(b) K-map for Q^+

Figure 6: K-map for S and Q^+

C_1C_0 \ EQ	00	01	11	10
00	0	0	0	0
01	×	1	1	×
11	×	0	0	×
10	×	1	1	×

(a) K-map for C_1^+

C_1C_0 \ EQ	00	01	11	10
00	0	0	0	1
01	×	0	0	×
11	×	0	0	×
10	×	1	1	×

(b) K-map for C_0^+

Figure 7: K-map for C_1^+ and C_0^+

From the k maps, We can conclude that

$$S = C_0 + C_1 + EQ'$$

$$Q^+ = C_1 + C_0 + E$$

$$C_1^+ = C_1 \otimes C_0$$

$$C_0^+ = EQ' + C_1C_0'$$

Particularly, one important Control Unit component is the Load A/B (shown in figure 8 As the lab manual shows, Load A/B should load D3-0 into shift-register to Register A/B when the Execute is not working and the corresponding Load A/B is high. We choose to use three MUX to implement the design. As a matter of the fact, there's a simper way to design this Load A/B part, as we don't need MUXs for inputs Load A and B. Load A and B can be directly connected to the mux that determines input to the registers. The reason why we do this is that this design helps us understand the circuit at initial attempt, but we later find it that it can be optimized.

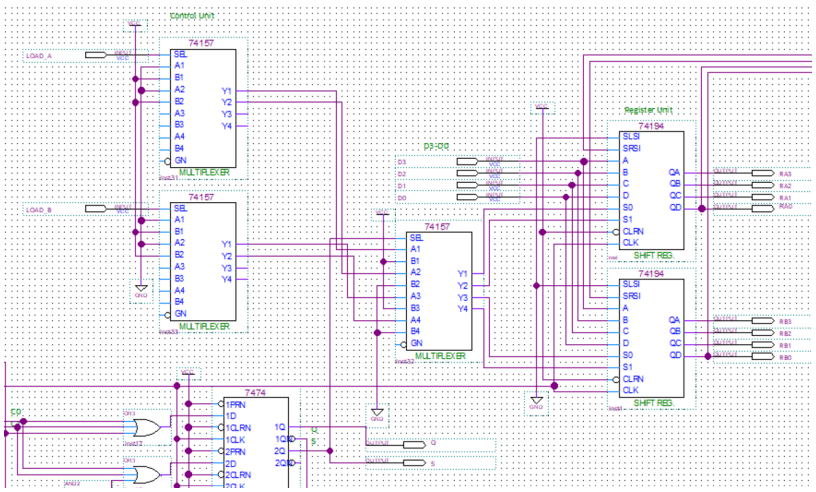
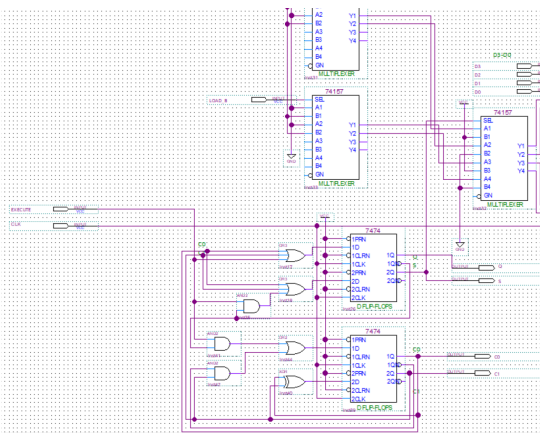
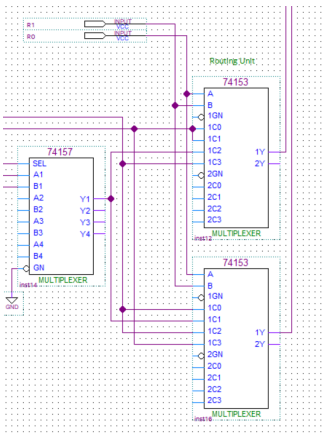


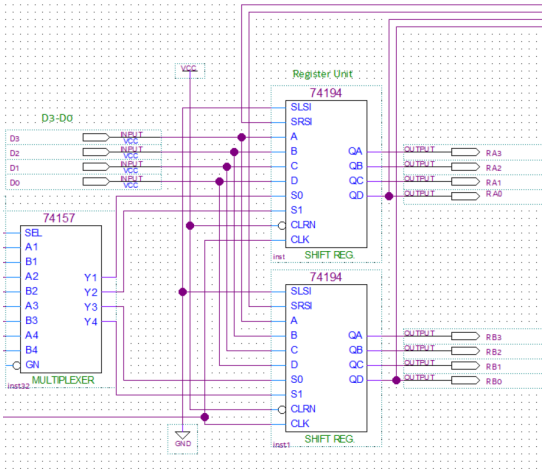
Figure 8: Load A/B Diagram



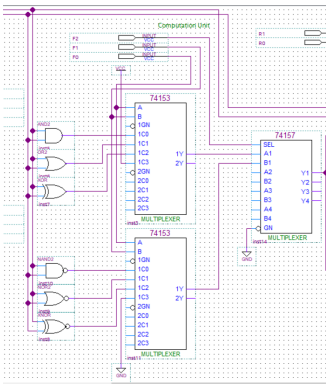
(a) Control Unit



(b) Routing Unit



(c) Register Unit



(d) Computation Unit

Figure 9: Circuit Schematic

4.2 Detailed Circuit Schematic

The Control unit is a Mealy Machine, it consists of AND, OR and flip-flops and follows the logic as suggested by the logical expression. The detailed circuit is shown in figure 9a

The Routing Unit contains three MUX, the design is shown in figure 9b

For the Register part, we use two 74194 chips to implement our design. The implementation is quite simple, the detailed circuit is shown in figure 9c

For the Calculation Unit implementation, it contains the corresponding computational gates and Two 74153 MUX, the detailed circuit design is shown in figure 9d

5 Description of all bugs encountered, and corrective measures taken

The realization of the whole circuit is relatively smooth. The only thing that has twists and turns is understanding the specific operation of control logic, especially fully understanding what changes FSM will make in the actual circuit. We struggled with the relationship between S state and E for a long time and failed to understand the relationship between Load A/B and E, which led us to spend a long time consulting TA and re-reading the experimental manual in the later stage of debugging.

First of all, for the pin problem of the 74194 chip, we used the wrong link sequence at the beginning of the output, resulting in a large deviation in our waveform simulation. After repeated inspection, we found the problem of pin sequence.

Secondly, we have a problem with our understanding of E. At first, we thought that only when E is at a high level can load play a role. Finally, we found that if E is at a high level, S will always be at a high level. In this case, the priority of load and S is not mentioned at all. After consulting TA, we found that our understanding is biased.

6 Conclusion

In this lab, we design and build a bit-serial logic operation processor. We can load values into two 4-bit shift registers, and do some operations (NAND, NOR, XNOR, SET0, etc.) on them. We can also determine which register will the output be stored to through a routing unit. We use a Mealy machine to design the control unit, and a counter is used to reduce the complexity of the control unit circuit.

7 Post-Lab Questions

1. **Difficulties you had in debugging the circuit. How does the modular approach help you isolate design and wiring faults?**

   In the process of building this circuit, we did not fully understand the implementation of the circuit control logic at the beginning, especially the use of Load. In the beginning, we thought that Load only works when Execute starts to work. Finally, according to FSM's State analysis, we found that Load actually works when Execute is at a low level. Finally, we understood that Execute starts to calculate rather than start to work on a static circuit.

2. **What are the tradeoffs of a Mealy machine vs a Moore machine?**

   The output of Moore machine depends only on its current state, which indicates that Moore machine is more intuitive. It's easier to design but may require more circuit elements. The output of Mealy machine depends on its current state and input. So generally Mealy machine requires less number of states, indicating a simpler circuit implementation. But it's more difficult than Moore machine. In this lab, Moore machine will have a large truth table and lots of gates. But Mealy machine only has two states. So we choose Mealy machine in this lab.

8 Appendix: Simulation Result

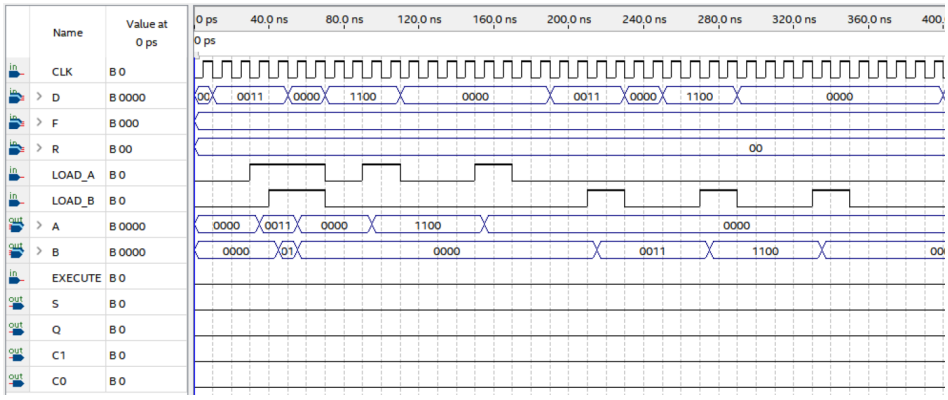


Figure 10: 0-400ns Wave Form

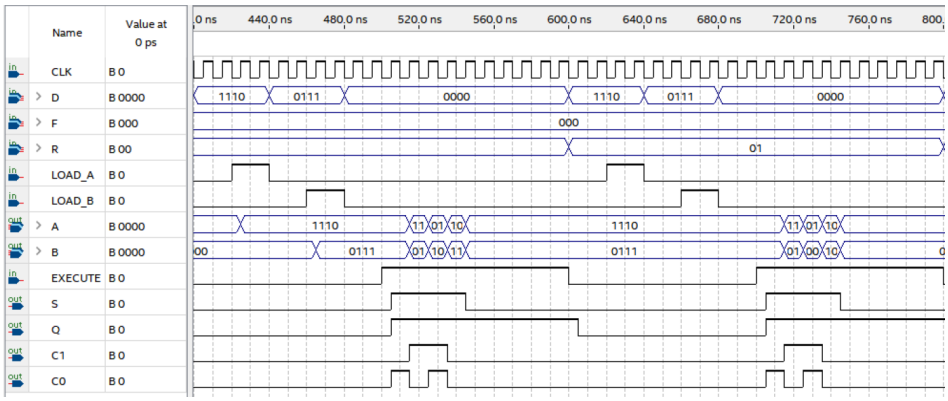


Figure 11: 400-800ns Wave Form

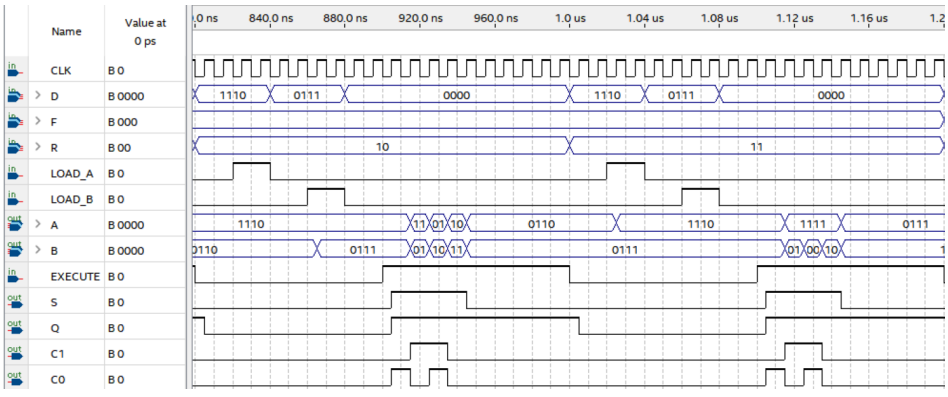


Figure 12: 800-1200ns Wave Form

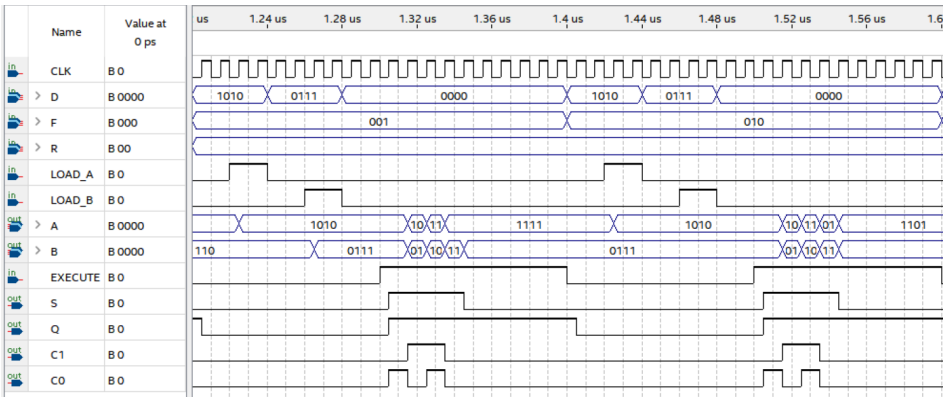


Figure 13: 1200-1600ns Wave Form

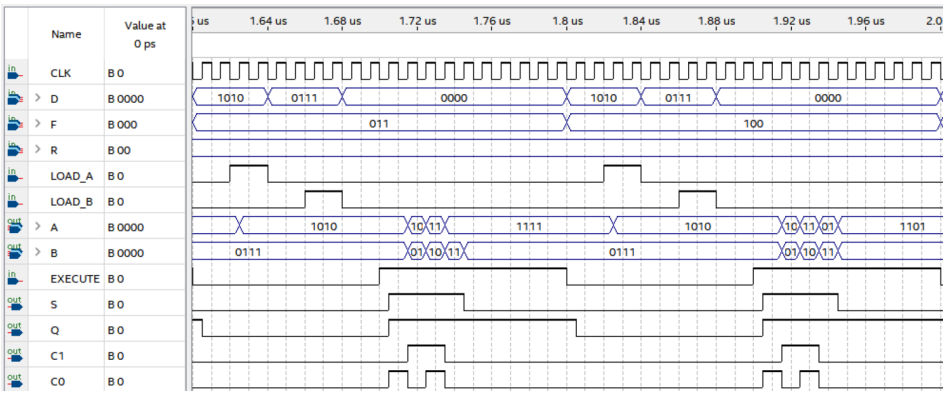


Figure 14: 1600-2000ns Wave Form

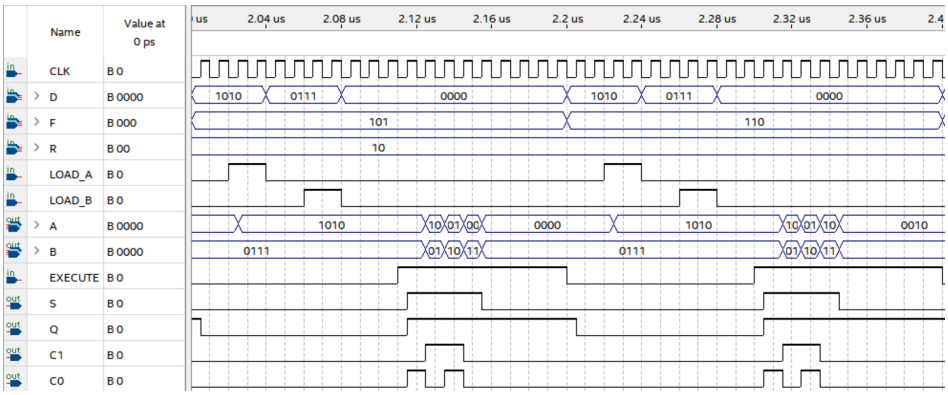


Figure 15: 2000-2400ns Wave Form

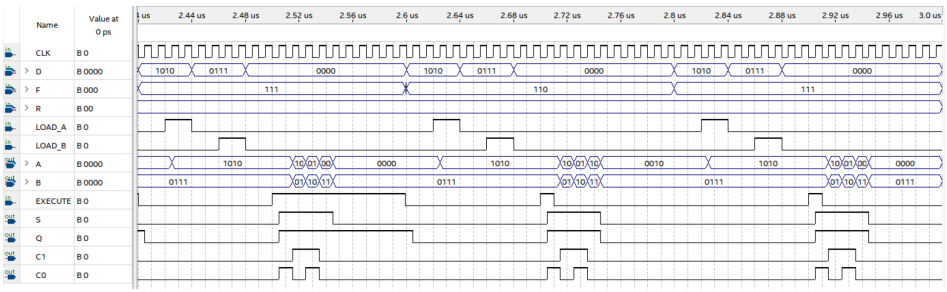


Figure 16: 2400-3000ns Wave Form