ECE 385

Spring 2024

Experiment # 3

Lab3: A Logic Processor

Name: Jie Wang, Shitian Yang

Student ID: 3200112404, 3200112415

Prof. Chushan Li, Prof.
Zuofu Cheng
ZJU-UIUC Institute
Jan. 26, 2024, Friday D-225
TA: Jiebang Xia
Demo: 5/5

1. Introduction

Purpose of Circuit:

In this lab, we are required to design and build a bit-serial logic operation processor using 4-bit shift registers, multiplexers, a counter, and a finite state machine (FSM) for the control unit. Simulated on Quartus 2 Environment, the processor will perform eight different logic functions and route the results in four different ways. The design includes a register unit with two registers (74194), a computation unit for logical operations, a routing unit for signal direction, and a control unit realized by an FSM to manage the register inputs.

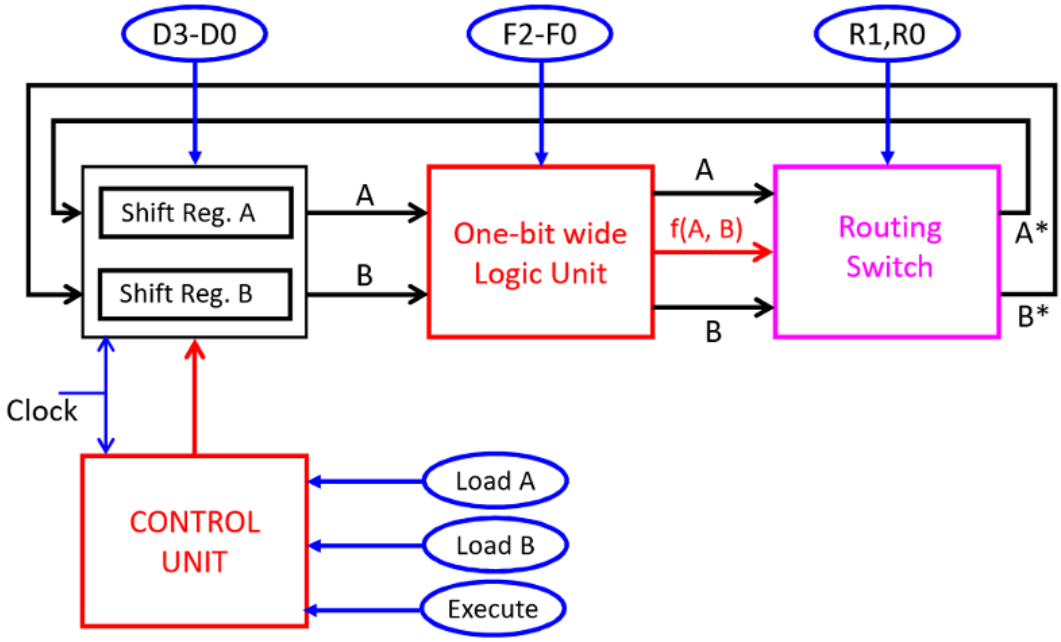


Fig-1: Logic Block Diagram of the circuit

The expected instruction can refer the following data sheet, with:

1. 4 bit data input D[3:0] and Load A&B signal, loading into register A, B separately.
2. Control signal F[2:0] to select the logic operation function.
3. Routing signal R[1:0] to select the router output register.
4. Execute signal EXE[0] to turn on the shifting based processer.

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

Fig-2: Function Look Up Table

2. Prelab Question

1. **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.**
   We can use XOR. The XOR gate acts on the input signal and a control bit. If the control bit is 0, the output is the same as the input signal; if the control bit is 1, the output is the inverted signal.

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

   It can improve testability and reduce development time by allowing designers to focus on individual components rather than the entire system. Each module can be tested independently, making it easier to locate and fix errors. Moreover, modular designs enable parallel development of components, speeding up the overall design process.

3. **Design, document and build the circuit described in Part II. Your circuit should be able to perform correctly all the functions listed.**
   We use two 74194s to build the shift memory, A~D ports to load data, S0~S1 to control the shift, hold or read status. And the we use a 74157 and a 74153M in series as 8-1 MUX and load F2~F0 to choose which one to output. Then use a 74153 (8-2 MUX) to decide four 2-membered pairs by using R0~R1. For control unit, we decide to use a counter and a flip-flop to simulate the state machine. The control unit will read EXECUTE and its outputs to send signal to the S0 port for those two shift memory registers.

3. Operation of the Logic Processor

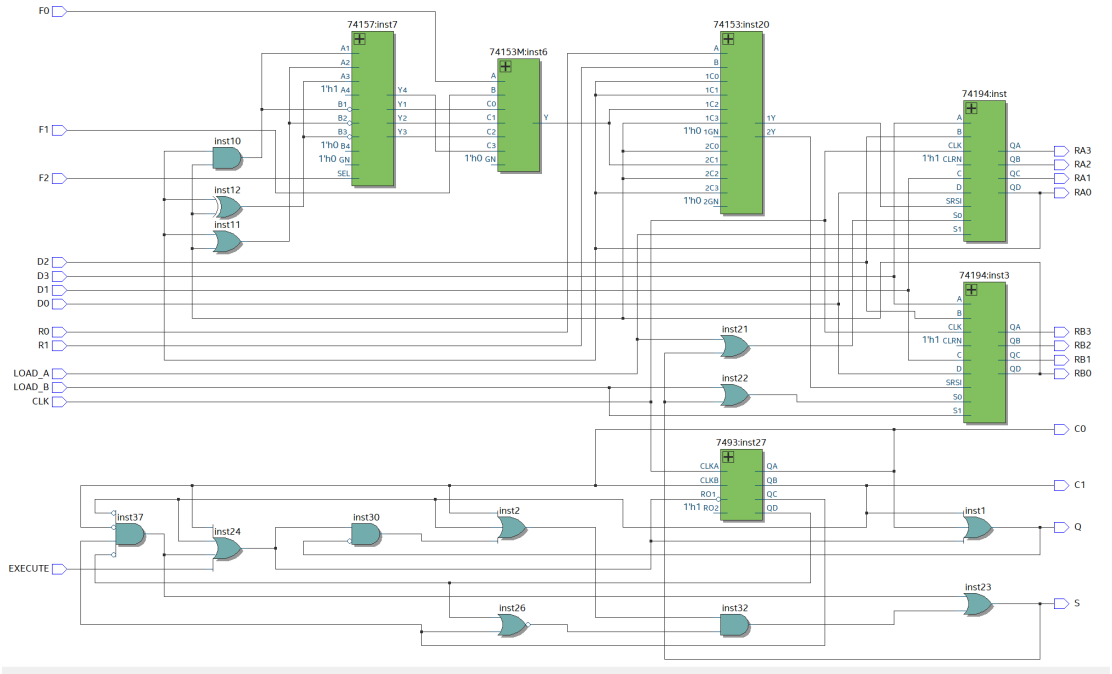


Fig-3: RTL Viewer of Our Circuit

As demonstrated in Fig-3, The data flow in our logic processor can be considered as below:

A. Loading Data into Registers:

1. load by D: when **load_A** or **load_B** is on, the registers will load D3-D0 to the registers.
2. load by compute: in each shift step, a computing result will sent to the SRSI and registers will use shift mod to shift memory this bit

B. Initiating Computation and Routing:

1. To init computation, we must input **F2-F0** (function choosing bits, to decide to use which function to compute) and **R1-R0** (memory choosing bits, to decide to use which register to store the result). It should be noticed that **R1R0=00 or 11** will not memory the computation result but just keep or exchange A and B memory.

2. To start computing, **EXECUTE** is on. For design, in case of garbage **EXECUTE** signal, once **EXECUTE** is on, the computation and shift function will not stop until finish the computation cycle (here is 4 time step for 4 bit register). Then shift registers will shift the memory bits using the clock signal. And the last shift bit will be sent to the function module(controlled by F2-F0) to be computed. Then two of register last bits and the one result bit will be chosen by R1R0 to be sent back to the register and be restored in the first place.

4. Processor Description and Diagrams

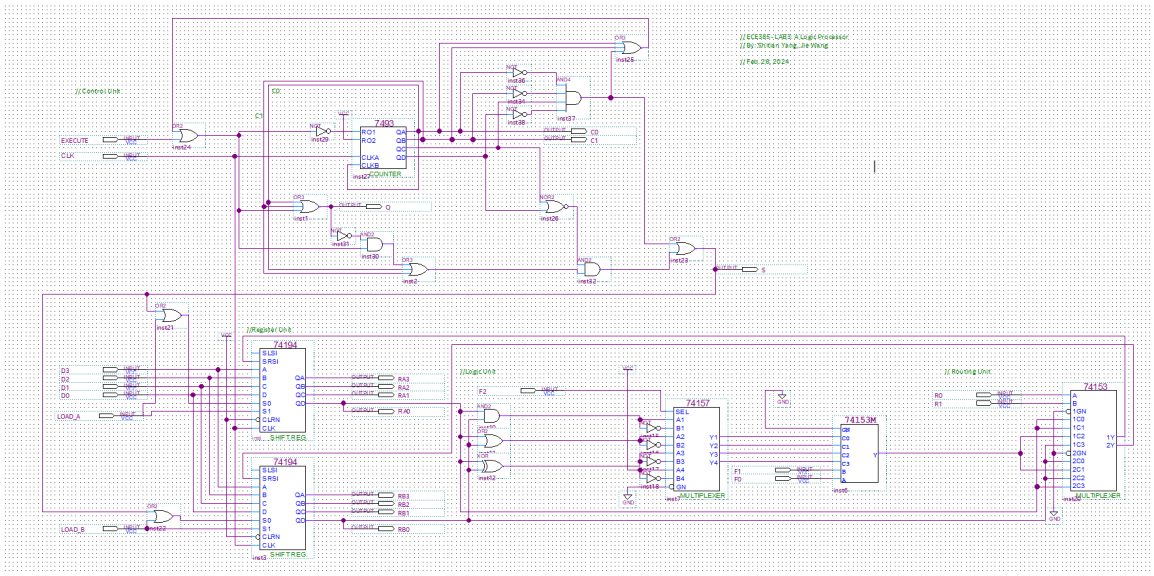


Fig-4: Overview of the circuit

The followings are the details of our circuit: We follow the system diagram given in the slide, as attached in Fig-1. The most important and difficult part may be the Control Unit, while the other three unit are fairly close to the design.

Register File

As referred to the lab manual, we selected **74194** as the storage register. Following the datasheet, we connect control unit to the S0 pin, with output to the logic unit.

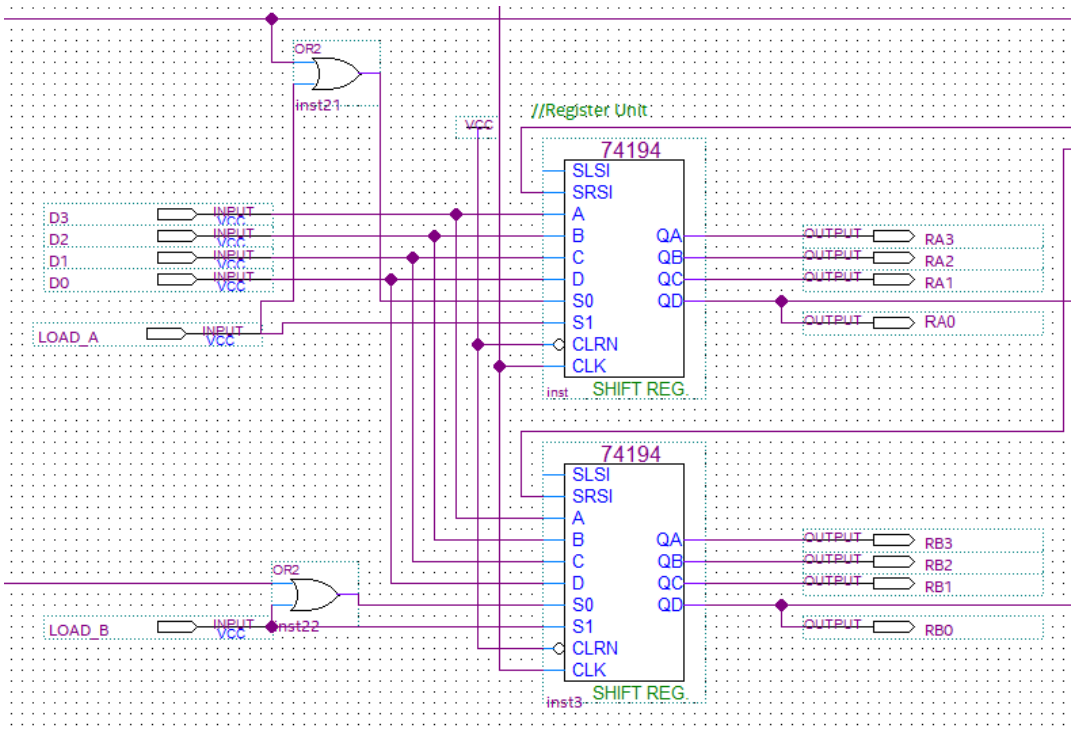


Fig-5: Register File

Logic Unit

To simplify the circuit, we did a binary layer design here. Firstly, we used **74157 2-line-to-1-line data selector** to filter the SELCTION signal F2. This is because we noticed that the **Function Look Up Table** is evenly distributed. Then, we used a **74153-MINI 4-to-1 multiplexer** to select the **F1 and F0** signal, satisfying the logical operation while reducing the complexity of overall circuit.

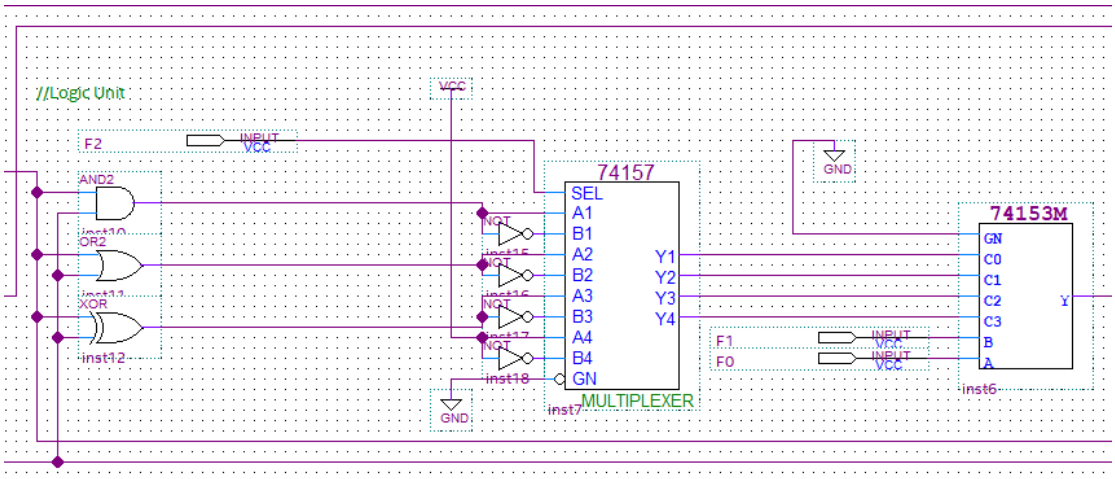


Fig-6: Logic Computation Unit

Routing Unit

The Routing unit is fairly easy, a **74153 4-to-1 multiplexer** is enough to meet the functionality. The key is to config the **R0, R1** as two-digit input. We firstly made a mistake that we wrongly swap them.

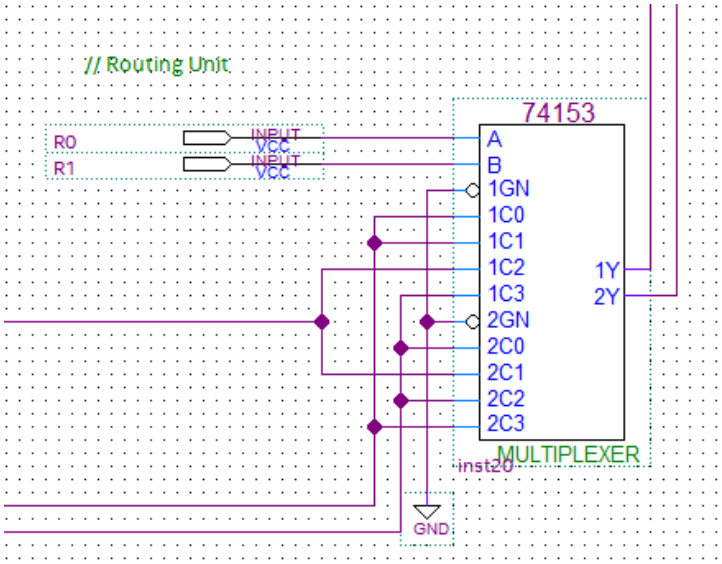


Fig-7: Routing Unit

Control Unit

On the contrary to normal FSM built by **four D-flip-flops** (told by TA and classmates after demo)**,** we did a mixture of Counter and FSM, gating the EXECUTE signal to ensure the cycle stay consistent to the FSM. The detailed mechanism can be referred as below:

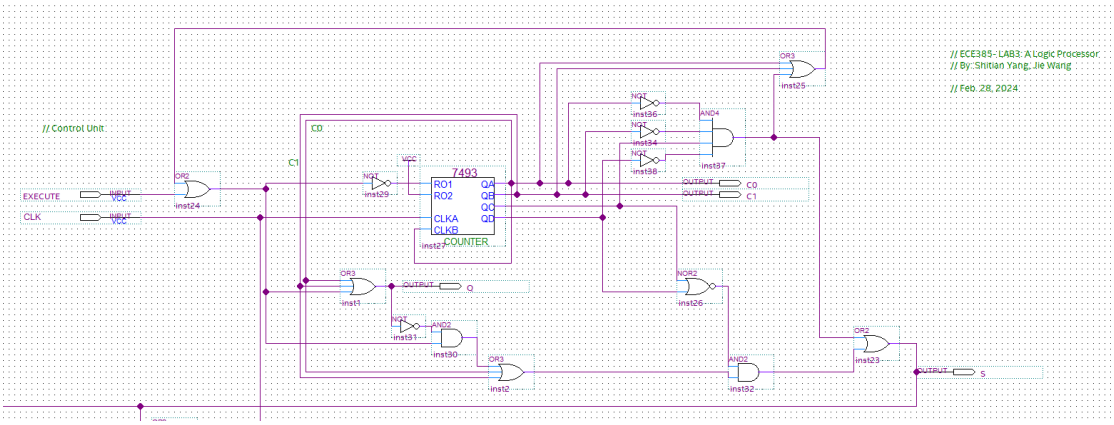


Fig-8; Control Unit with Counter Design

State Machine Diagram

I) We use meanly machine. We use a counter and two flip-flops to ensure the state machine.

II) When execute gets a up signal, the counter will going to work and use Q to lock the flip-flop of the execute. When counter get the “11” signal, it will give signal to another flip-flop and use xor to stop the counter. If E is 0 and Q will be zero to unlock the flip-flop of the execute. The two flip-flops have same value, the counter will wait. The two flip-flops have different values, the counter will count.

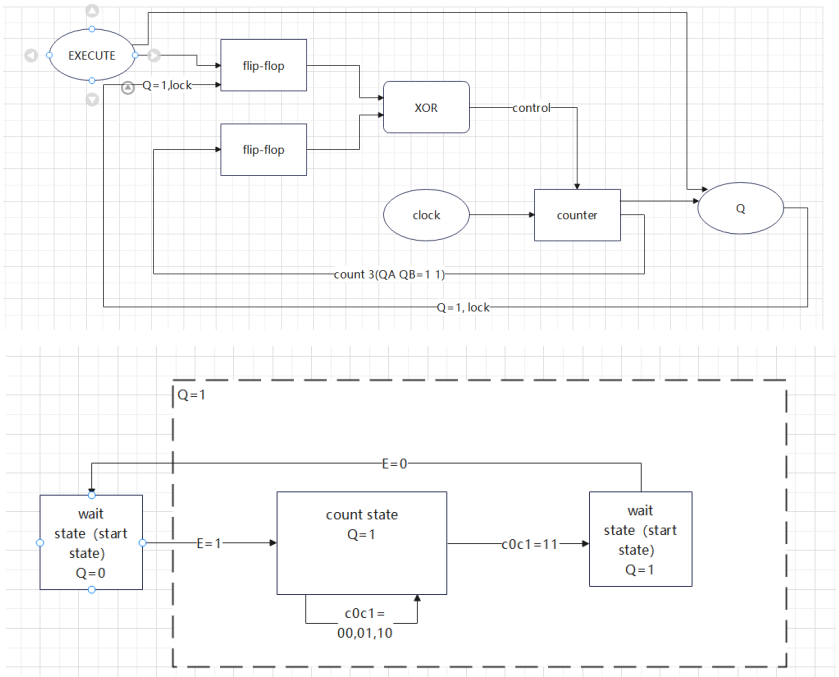


Fig-9: Detailed Mechanism of Our Mealy FSM

III) Q=E+C0+C1. S*=EQ'+C0+C1. Those two flip-flops are parallel work and send signal to xor to keep the state. And Q=1 will lock the execute to ensure it will be recognized as don't care when counter is working.

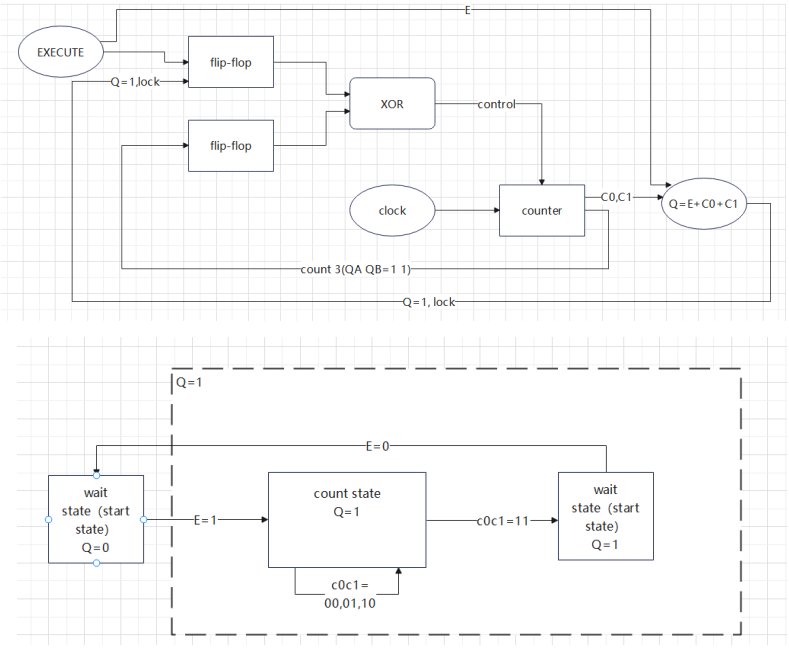


Fig-10: Q state Mechanism

K-Map

Below is the K-maps of our mealy FSM, using the same mealy state diagram in the slide.

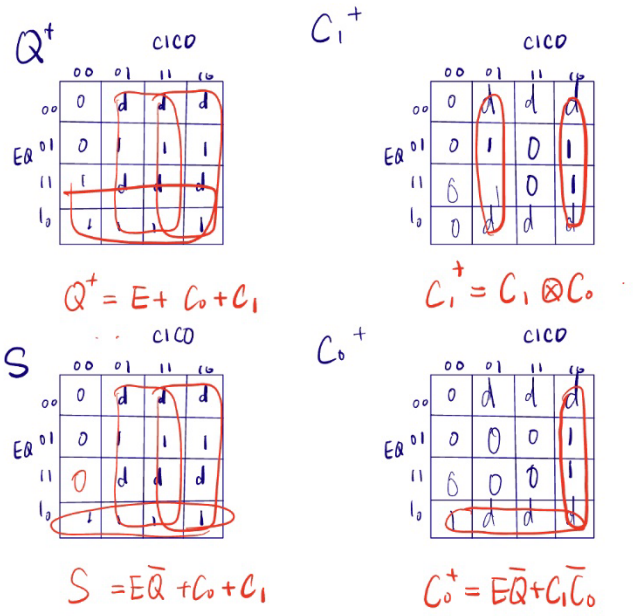


Fig-11: K-map and Equation for Q+, S, C1+, C0+

5. Bug Log

➢ **Description of all bugs encountered, and corrective measures taken:**

1. Incorrect Function Selection:

- **Issue:** When observing the output, we noticed that some functions were incorrectly swapped.
- **Cause:** This was traced back to an incorrect order in the F2-F0 connections.
- **Solution:** We checked the wiring of F2-F0 and corrected their alignment to the appropriate ports on the chip. This realignment ensured the functions matched the intended output when read from the wave mapping function (wmf).

2. Unstable Q State in Control Unit:

- **Issue:** The Q state was unstable, with the Execute (E) input directly influencing the Q state, leading to inconsistent behavior.
- **Cause:** The direct dependency of Q state on the E input without any locking mechanism.
- **Solution:** To stabilize the Q state, we integrated two flip-flops to lock its state. This design ensured that the E input's transition would not inadvertently affect the Q state, especially locking the E's flip-flop when Q is high, thus achieving the desired stability.

3. Inaccurate Shift Signal Duration:

- **Issue:** The shift signal duration was consistently shorter than required, lasting only 3 clock cycles instead of the intended 4.
- **Cause:** Upon investigation, we found that the original design of the trigger condition was flawed, not accounting for signal delay inherent in the flip-flop and counter components.
- **Solution:** We revised the logic that dictates the shift signal duration. By redesigning the trigger conditions and accounting for component delays, we successfully extended the shift signal to the correct duration of 4 clock cycles.

4. Counter Malfunction with 7493 Chip:

- **Issue:** The counter was not functioning as expected. Notably, the QB output port on the 7493 chip failed to produce any output.
- **Cause:** After a thorough review of the 7493 chip specifications, we realized the issue stemmed from a misconnection in the chip setup.
- **Solution:** We corrected the issue by connecting the QB output to the clkB input. This adjustment enabled the counter to surpass a count of 2, aligning with the chip's operational requirements and ensuring proper counting functionality.

6. Conclusion

Lab 3 is comprehensive in the design and implementation of a bit-serial logic operation processor. We refamiliarize with the assembly of 4-bit shift registers, multiplexers, a counter, and FSM. We debugged issues like incorrect function selection and unstable control unit states, really getting the feel for the iterative nature of engineering. This lab wasn't just about getting circuits to work; it taught us the importance of a methodical approach and the value of modular design in simplifying complex tasks. All in all, it was a solid learning experience, blending theory with practical skills, and it's prepped us for more challenging projects ahead.

7. Post Lab Question

Discuss the design process of our FSM:

In Experiment 3, we adjusted the circuit design, especially in the control and logic computing units. Debugging allows us to solve the instability problem by adding a trigger to stabilize the Q state in the control unit. We also rearranged the F2-F0 connections in the logical unit to correct the feature selection error.

The modular design approach is quite helpful in debugging. It lets us solve problems module by module, such as isolating the control unit to fix the Q state and focusing on the logical unit to sort out the feature selection.

What are the tradeoffs of a Mealy machine vs a Moore machine?

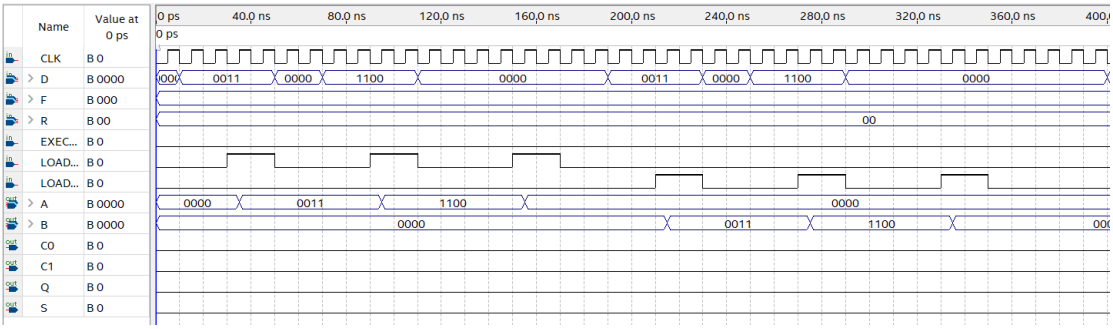
In designing our state machine, we chose a Mealy machine configuration. The decision between a Mealy and a Moore machine involves several trade-offs:

- **Responsiveness**: Mealy machines tend to be more responsive than Moore machines as their outputs are directly dependent on the inputs. This was crucial in our design, where immediate response to the Execute (E) input was needed.
- **Complexity**: Mealy machines can be more complex to design, as the output logic is dependent on both the state and input. This complexity was evident in our debugging process, especially when stabilizing the Q state.
- **Predictability**: Moore machines, with their outputs depending solely on the state, are generally more predictable and simpler in terms of timing. This predictability could have simplified some aspects of our design but at the cost of responsiveness.

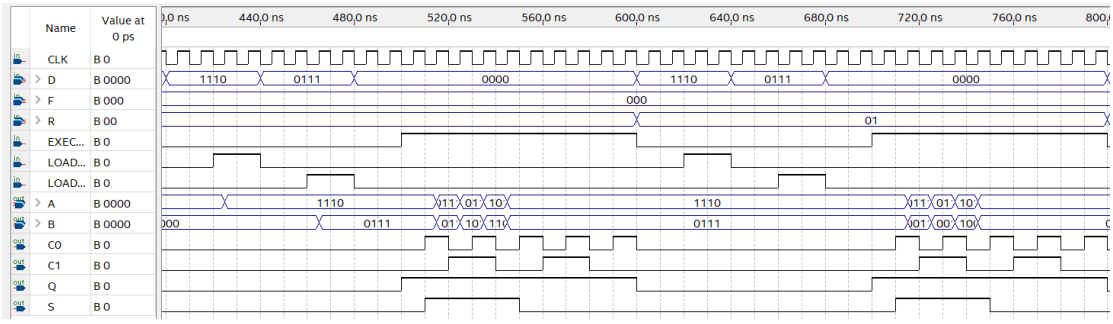
8. Appendix: Simulation Results

Our circuit passed all the test cases, winning 5/5 points in the demo. Here are the waveform screenshots captured every 400ns from standard testing input case ***'waveform.vmf'***:

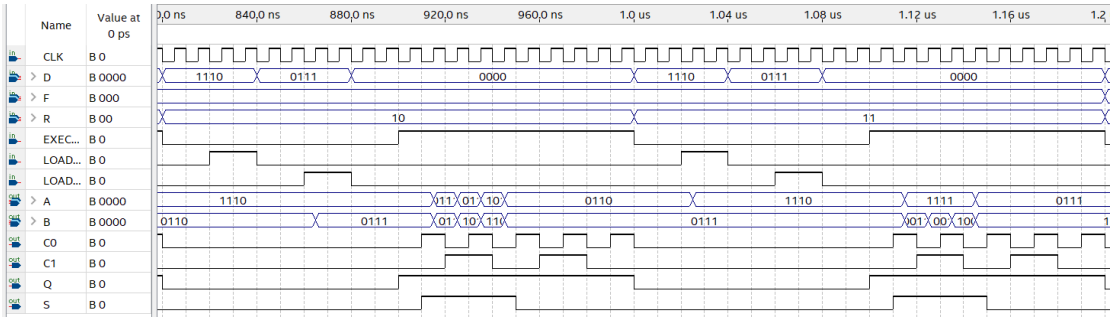
0-400ns



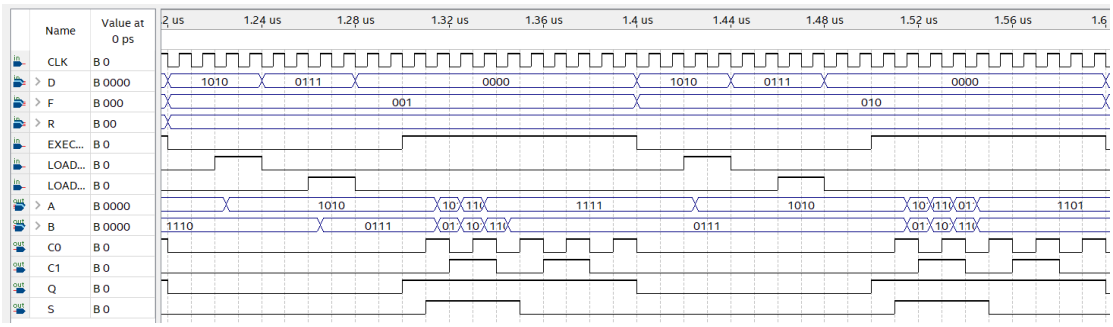
400-800ns



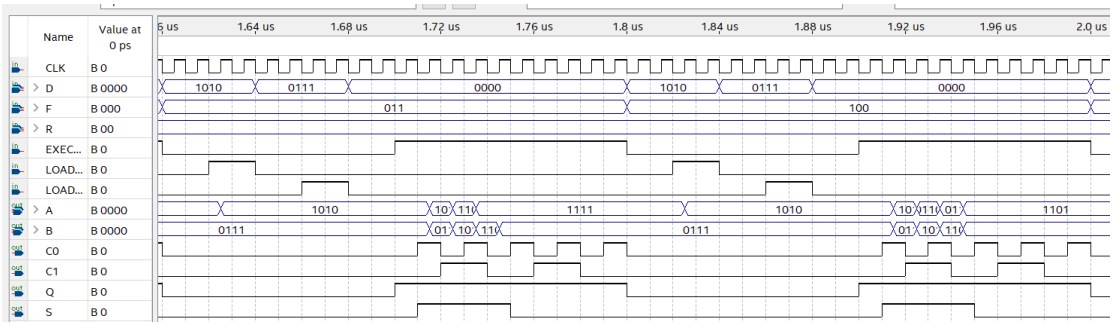
800-1200ns



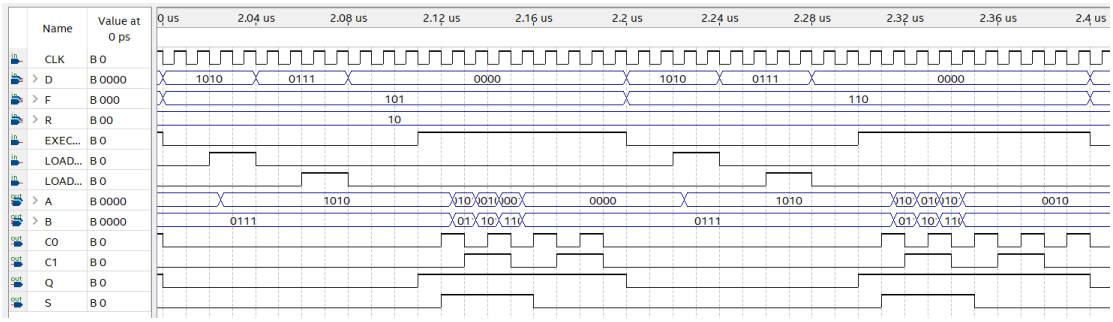
1200-1600ns



1600-2000ns



2000-2400ns



2400-3000ns