# 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

## 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.**

**图示

描述已自动生成**

**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.**

**图示, 表格

描述已自动生成**

**Fig-2: Function Look Up Table**

## 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.

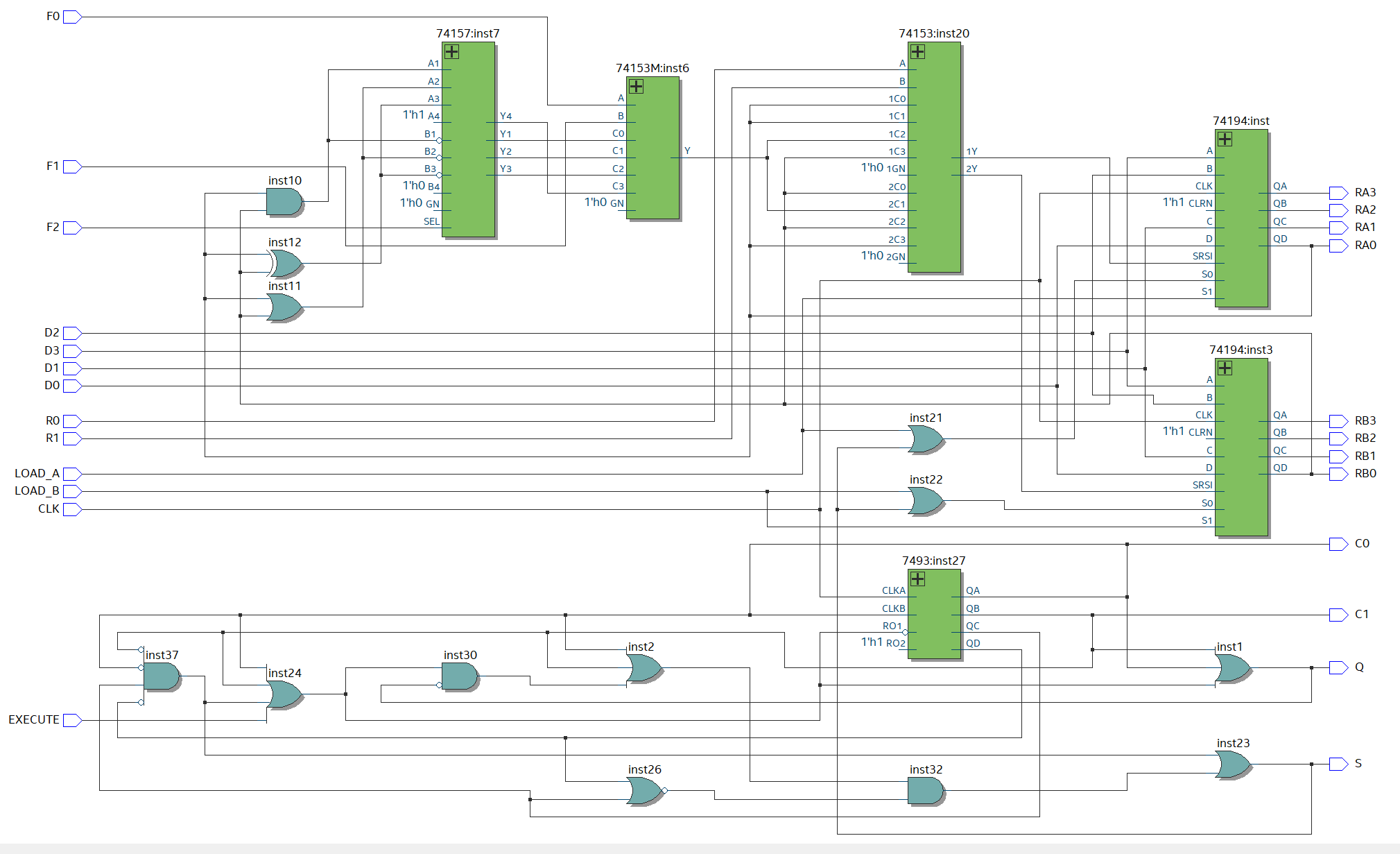
1. **Explain how a modular design such as that presented above improves testability and cuts down development time.**

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.

1. **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

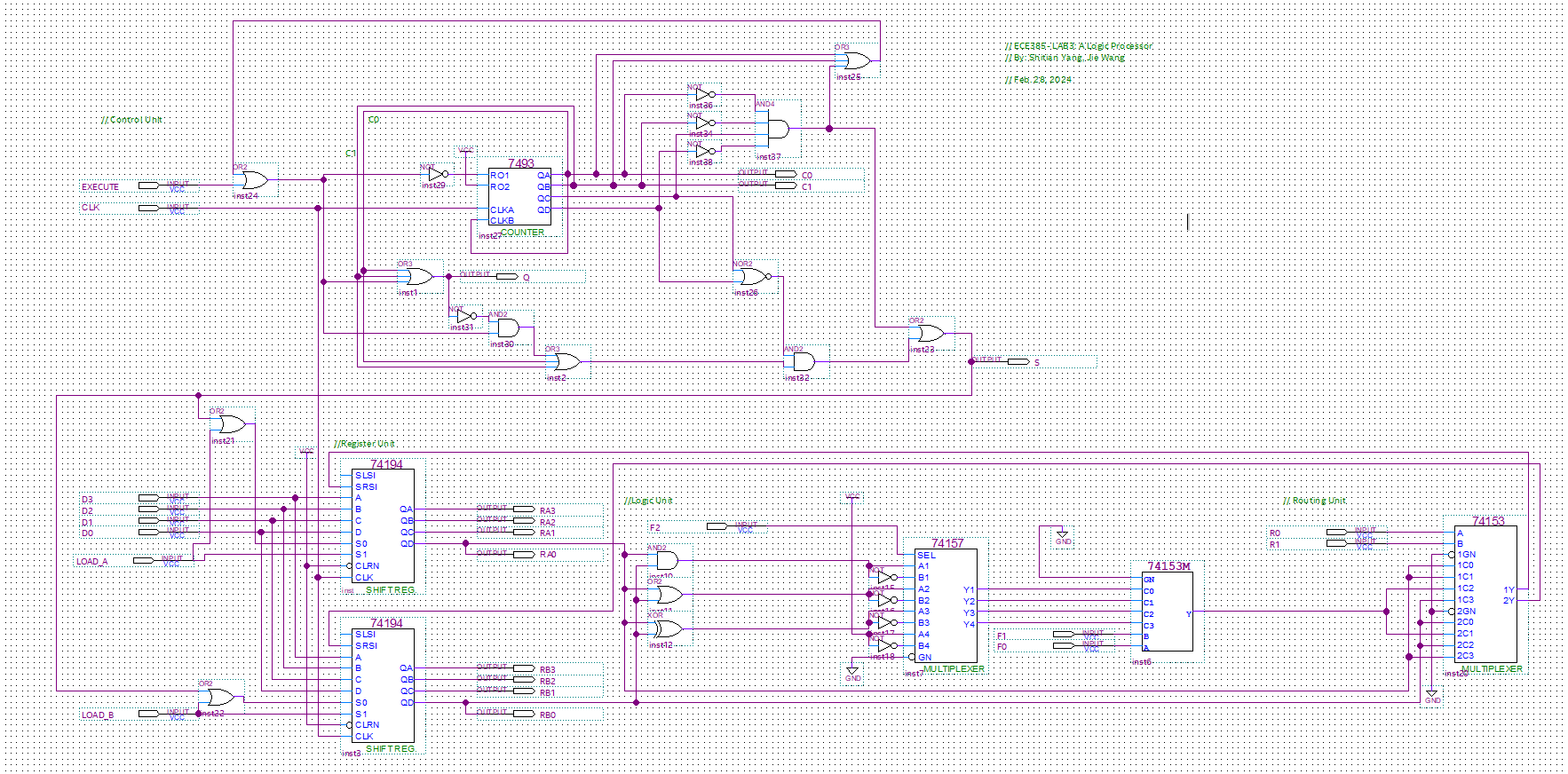


**Fig-3:** RTL Viewer of Our Circuit

a. Loading Data into Registers: - Describe the process of using switches to load data into the A and B registers.

b. Initiating Computation and Routing: - Outline the steps for using switches to start a computation and routing operation.

## 4. Processor Description and Diagrams



**Fig-4:** Overview of the circuit

The followings are the details of our circuit:

### Register File

图示, 示意图

描述已自动生成

**Fig-5: Register File**

**As referred to the lab manual, we selected 74194 as the storage register.**

### Logic Unit

图示, 示意图

描述已自动生成

**Fig-6: Logic Computation Unit**

### Routing Unit

图示, 示意图

描述已自动生成

**Fig-7: Routing Unit**

### Control Unit

图示, 示意图

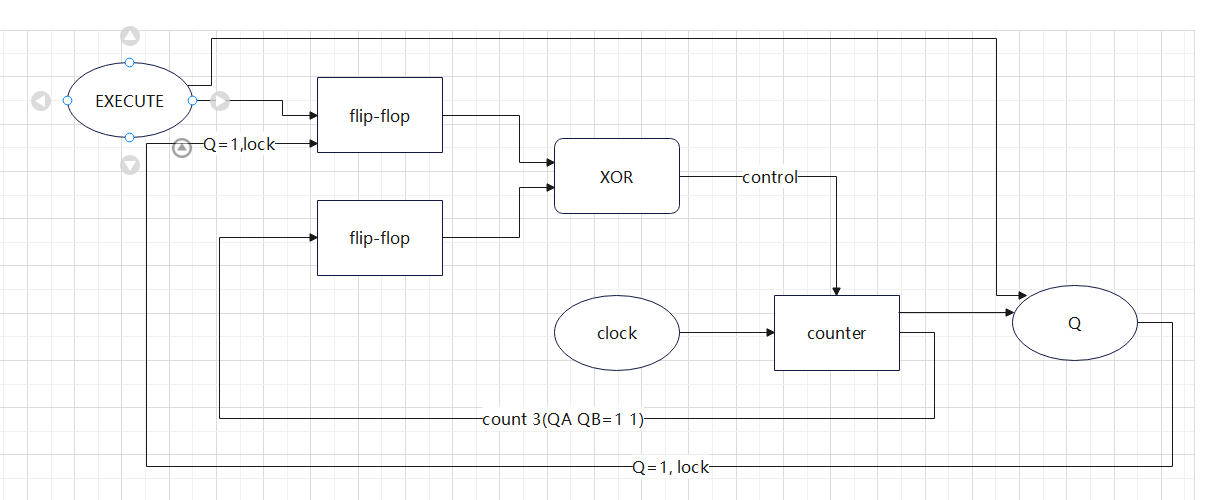
描述已自动生成

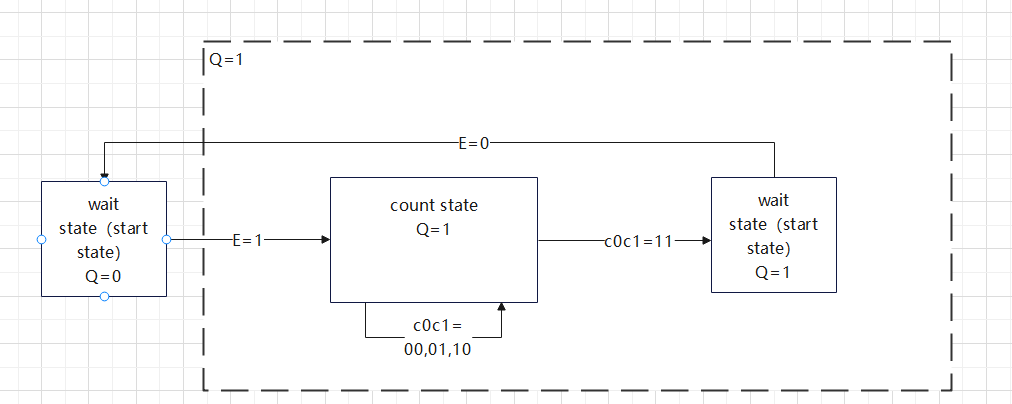
**Fig-8; Control Unit with Counter Design**

1. State Machine Diagram: - Indicate whether a Mealy or Moore machine was used. - Label and describe each state and transition. - Include a table for detailed information if necessary.

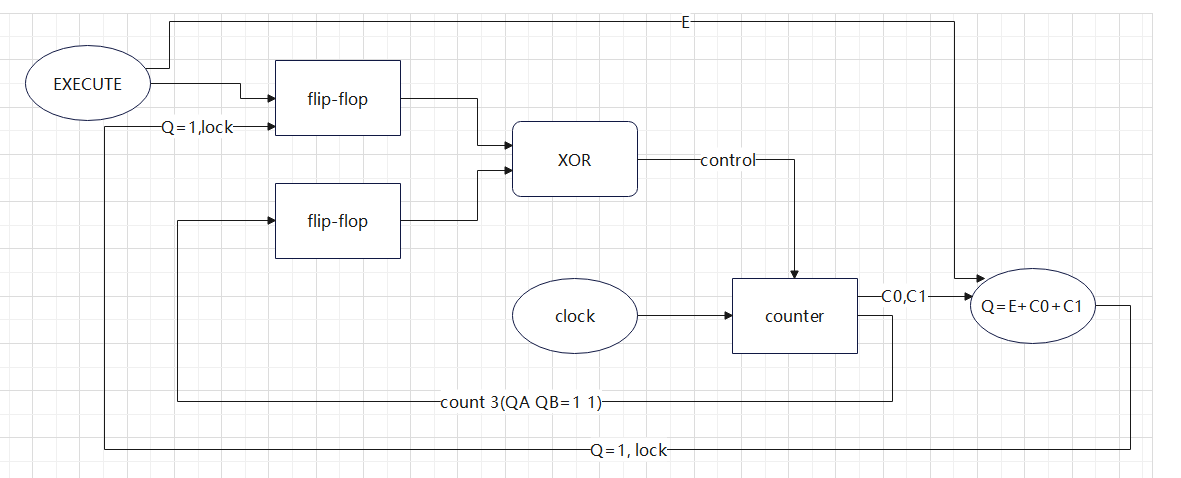
### State Machine Diagram

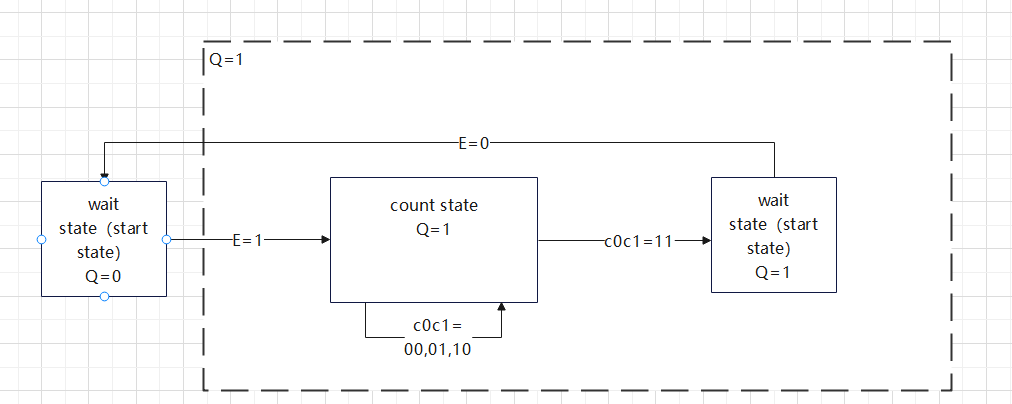
1. We use some meanly machine. We use a counter and two flip-flops to ensure the state machine.
2. When execute gets a up signal, the counter will going to work and use Q to lock the flip-flop of the execute. When counter get the “11” signal, it will give signal to another flip-flop and use xor to stop 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.





1. 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.





## 5. Design Steps and Circuit Schematic

a. Design Procedure: - Document the design steps, including the use of K-maps or truth tables. - Discuss design considerations and tradeoffs. b. Circuit Schematic: - Gate level schematic of the circuit. - Detailed schematic for complex components like the control unit. - Label inputs, outputs, intermediate signals, and mode pins.

王杰：明天写上之前推导的原理图

## 6. 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.

## 7. Conclusion

a. Lab Summary: - Concise overview of the lab experience and results.

## 8. Post Lab Question

这个部分要大改

### **Discuss the design process of our FSM:**

When designing a state machine for a digital system like the logic processor in ECE 385, the process typically involves several steps:

1. **Requirement Analysis**: Understand the functional requirements of the system. In this case, it involved controlling a bit-serial logic operation processor for different logical operations and routing.
2. **State Identification**: Determine the various states the system can be in. This involves considering all possible scenarios in the logic operation, like loading data, executing operations, and routing outputs.
3. **State Transition Logic**: Define how the system transitions from one state to another. This includes identifying the conditions under which these transitions occur, which are usually based on input signals or internal conditions.
4. **Output Logic Definition**: For each state, define the outputs of the system. In a Moore machine, the outputs are determined by the states alone, whereas in a Mealy machine, they depend on the states and the inputs.
5. **State Diagram and Table Creation**: Draw the state diagram and create a state transition table. This visual representation helps in understanding the flow of the state machine and is crucial for further implementation steps.
6. **Implementation**: Translate the state machine into hardware using logic gates, flip-flops, and other digital components. This often involves simplifying the logic with tools like Karnaugh maps.

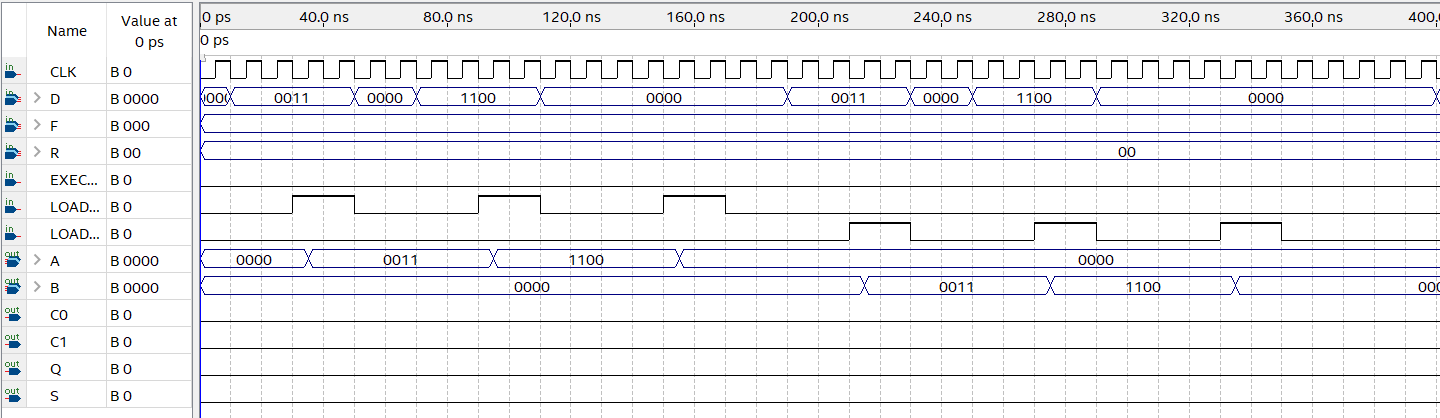
### **What are the tradeoffs of a Mealy machine vs a Moore machine?**

* **Response Time**: Mealy machines can often respond faster to inputs as their outputs can change as soon as the input changes, without waiting for a state transition. Moore machines, however, have a one-clock-cycle delay as outputs depend only on states.
* **Complexity**: Mealy machines can be less complex in terms of the number of states because the same state can produce different outputs depending on the input. Moore machines might need more states for the same functionality since each state corresponds to a fixed output.
* **Predictability**: Moore machines offer more predictability in output, as the outputs are solely based on states. In Mealy machines, since outputs depend on both states and inputs, they can be less predictable, especially in systems with asynchronous inputs.
* **Implementation**: Moore machines are generally easier to design and implement as the output logic is separated from the state transition logic. Mealy machines might require more careful handling to avoid issues like glitches due to the direct dependence of outputs on inputs.

## 9. 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

图形用户界面, 应用程序, 表格, Excel

描述已自动生成

### 800-1200ns

图形用户界面, 应用程序, Excel

描述已自动生成

### 1200-1600ns

图形用户界面, 应用程序, Excel

描述已自动生成

### 1600-2000ns

图形用户界面

描述已自动生成

### 2000-2400ns

图形用户界面

描述已自动生成

### 2400-3000ns

图形用户界面, 应用程序

描述已自动生成