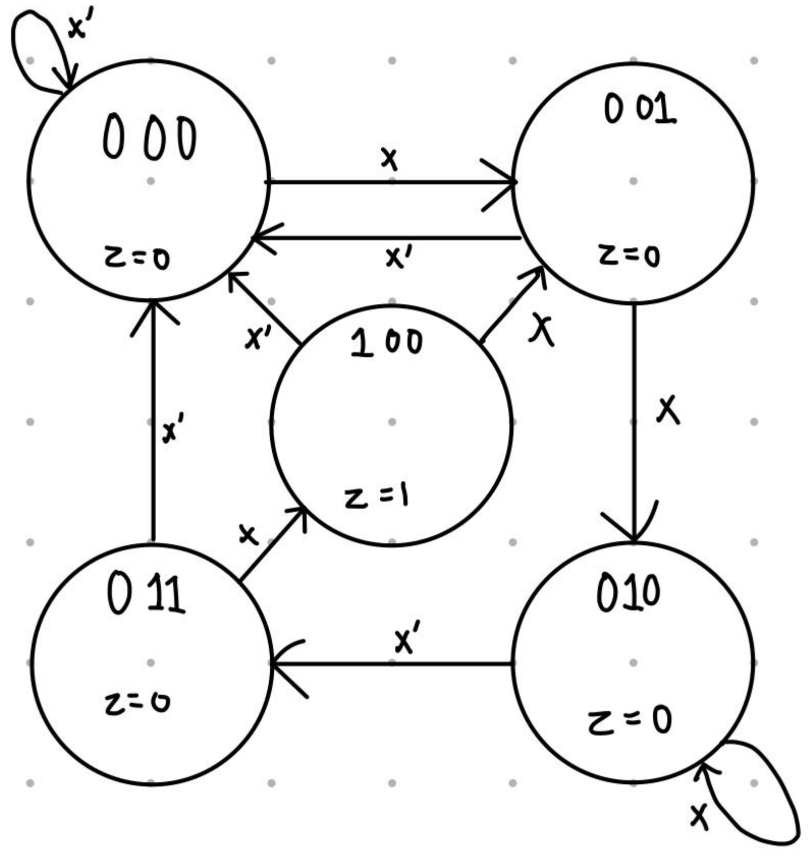
Name: Aaron Wang and Ethan Litte
CSE 20221 Logic Design
HW08: Finite State Machines

In this assignment, you will practice the finite state machine design process by designing three small finite state machines. This will set the stage for HW09 where you create a full high-level state machine, comprising a finite state machine and a datapath.

Problem 1 - Sequence Recognizer
In class on 11/5, we explored the design of a sequence recognizer using a Mealy finite state machine. Here, you will implement the sequence recognizer using our standard (Moore) finite state machine design process.

Sequence Recognizer Diagram
As a first step, write a finite state machine diagram for the sequence recognizer from class on 11/5. The sequence recognizer should use the same inputs (x, z) and recognize the same pattern (1101); see the class notes for details and examples. You may (neatly) write the diagram by hand, or use a digital tool (PowerPoint works fairly well). Rather than naming each state, just label the states with their unique encoding. For example, if you have a 3-bit state, your initial state should be labeled `000`. Make sure to clearly indicate your initial state and all transitions in your diagram.



Sequence Recognizer Existence and Uniqueness

In the space below, prove your finite state machine does not violate existence or uniqueness *for the sequence recognizer's initial state.*

```
Existence:
000 has transitions for x and x’: x+x’=1
Uniqueness:
000 has transitions for x and x’: xx’=0
```

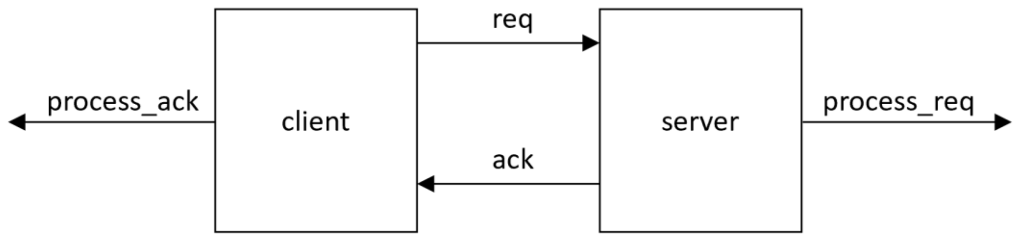
In the space below *for the next state reached after the sequence recognizer's initial state.* (If your diagram has multiple states that can be reached by the initial state, choose one.)

```
Existence:
001 has transitions for x and x’: x+x’=1
Uniqueness:
001 has transitions for x and x’: xx’=0
```

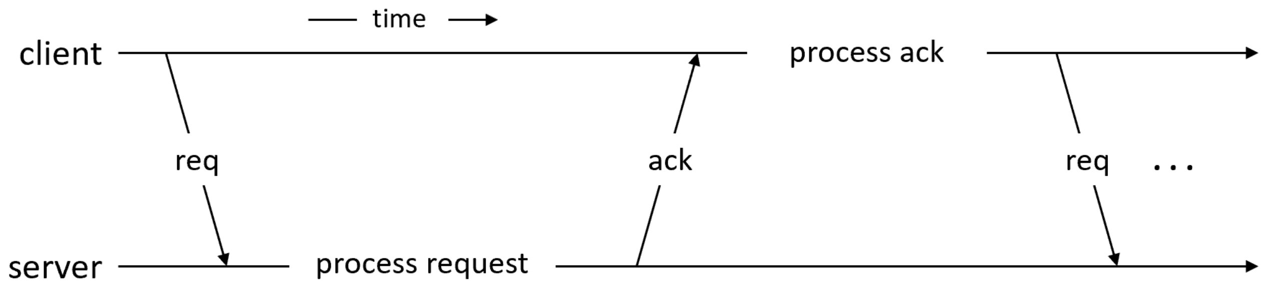
You are not required to implement the sequence recognizer in Logisim-evolution, but this would be good extra practice, and allow you to test your design. Another way to test your design - also not required - is to use the zyBooks FSM Simulator.

Problem 2 - Client/Server

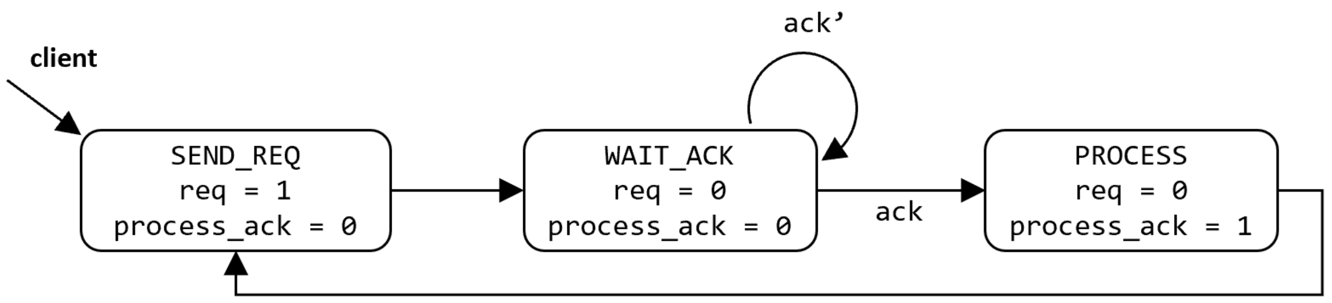
In general, a *server* is a system that provides a service to one or more *clients*. The service could be any of a variety of operations, such as accessing data from a database, using a coprocessor to run an algorithm efficiently, or providing an interface to an external input/output (I/O). There are a variety of standard protocols for communication between a client and a server, but the simplest is based on two control signals: a *request* (`req`) from the client to the server and an *acknowledgement* (`ack`) from the server to the client that the service has been completed.

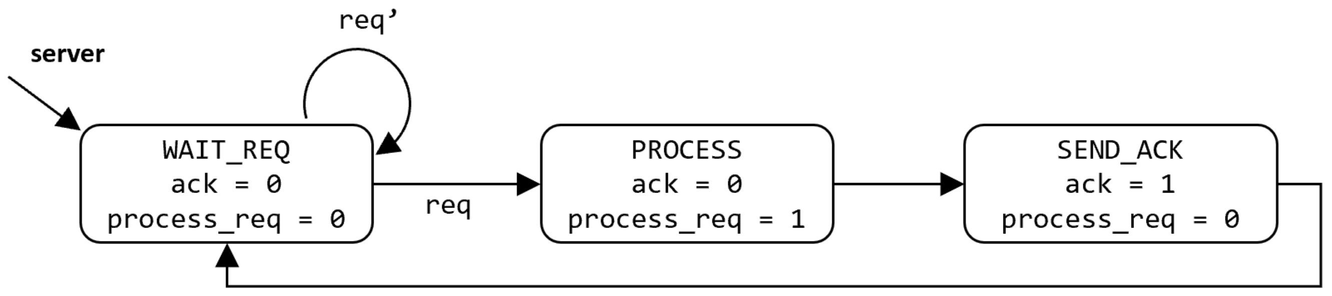


Note that there may also be data exchanged in either direction between the client and server alongside the request and acknowledge control signals; in this example, we are only considering the "handshake" involving the control signals. When a server receives a request from a client, it starts processing that request by raising a `process_req` signal that would be connected to the circuitry on the server side that performs the service. This may take some unknown number of clock cycles before it sends an acknowledgement that the processing is complete and that valid results are available. The client can then read those results and in turn process them by raising a `process_ack` signal connected to circuitry on the client side, before making another request and repeating the process.



Both the client and server processes can be represented as finite state machines, as shown below. Here, processing of the request by the server and processing of the acknowledgement by the client is shown as a single state, although it could be multiple states taking multiple clock cycles in other applications. (Note actions/outputs are shown inside the states rather than below.)





In the remainder of this assignment, you will design the client and server FSMs and use timing diagrams to simulate and test them individually, and the interaction between them.

Part 1 - Design and Test of the Client

You will implement the client using the approach from class where we define the state transition table (next state and output truth table), find the Boolean equations for the next state and output combinational logic, and then explicitly connect the combinational logic to the state register.

Client State Transition Table Combinational Logic

Complete the state transition table for the client FSM. If you would like, you can modify the table to combine p1, p0 (and n1, n0) into multi-bit `p[1:0]` and `n[1:0]`. (This will slightly simplify the design in Logisim-evolution.)

state	p1	p0	ack	n1	n0	req	process_ack

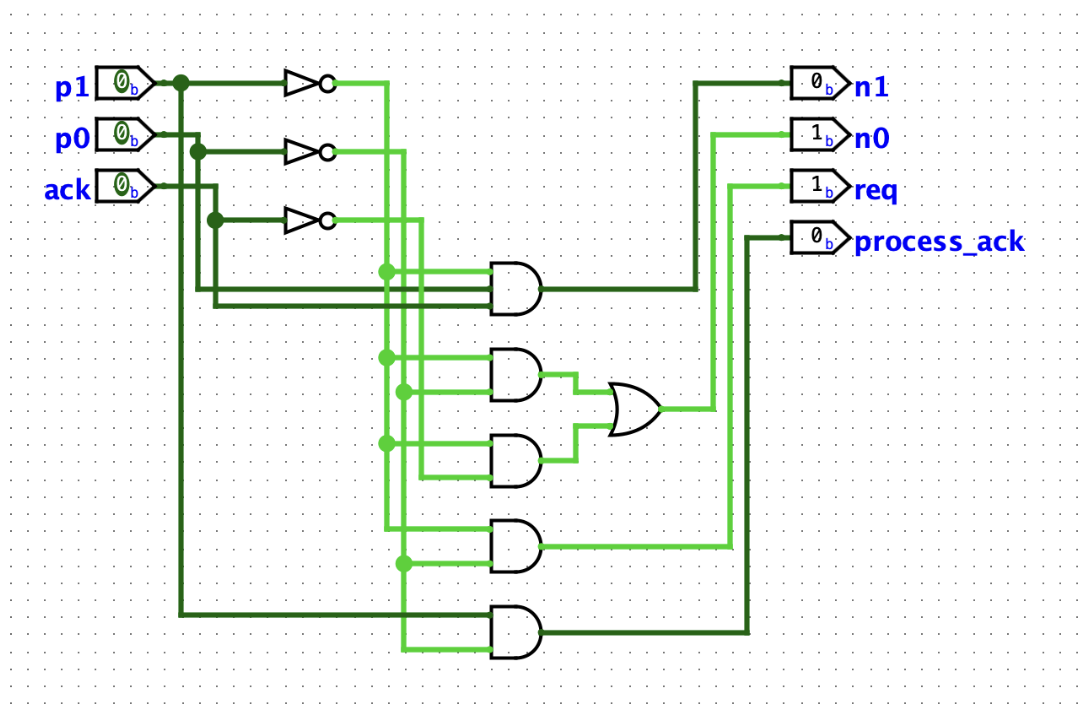
SEND_REQ	0	0	0	0	1	1	0
SEND_REQ	0	0	1	0	1	1	0
WAIT_ACK	0	1	0	0	1	0	0
WAIT_ACK	0	1	1	1	0	0	0
PROCESS	1	0	0	0	0	0	1
PROCESS	1	0	1	0	0	0	1
unused	1	1	0	0	0	0	0
unused	1	1	1	0	0	0	0

Optional: This box is not worth any points, but for extra practice as we head toward the (cumulative) final exam, consider completing this section. Solutions will be provided.

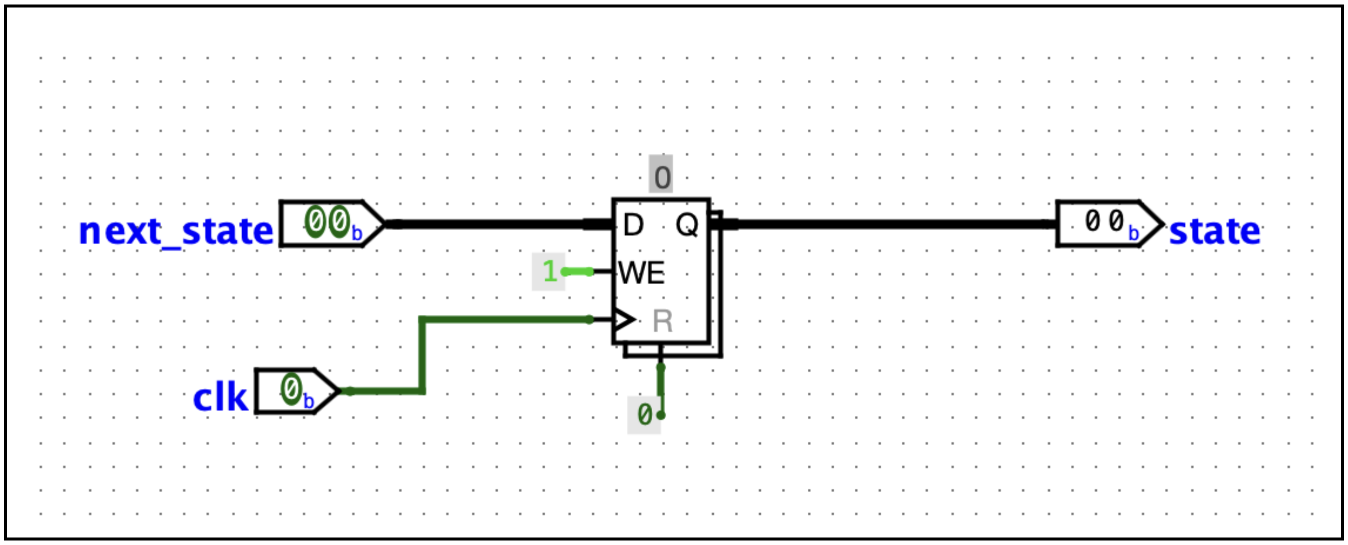
Use Karnaugh maps to determine the Boolean equations for `n1`, `n0`, `req`, and `process_ack` in minimized sum-of-products form. Show your work in the box below, including the K-maps and the minimized sum-of-products equations extracted from each K-map.

Client Combinational Logic

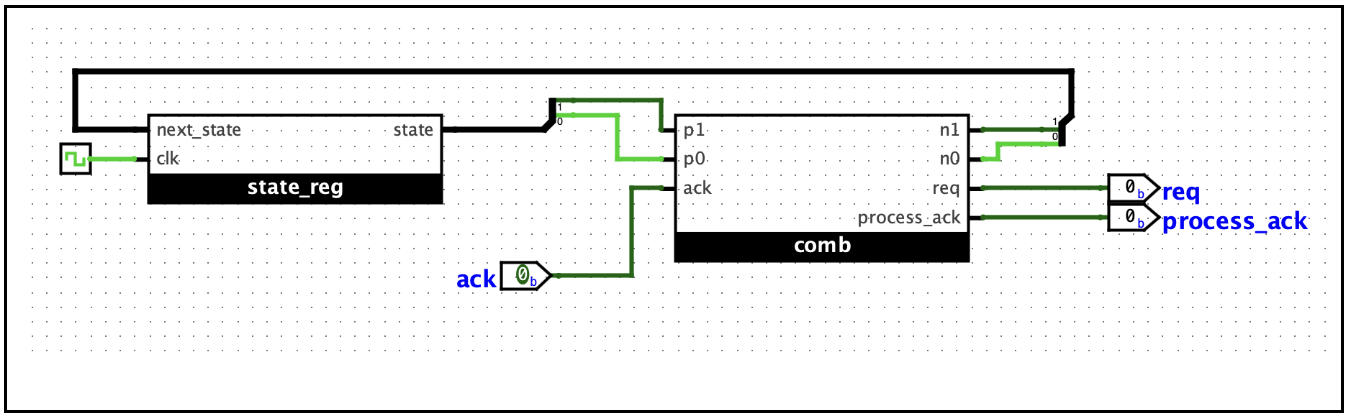
Create the combinational logic for the client FSM using the Combinational Analysis tool in Logisim-evolution. Copy a screenshot of your combinational logic into the box below.



Client State Register
Create an appropriately-sized state register for the client FSM in Logisim-evolution. Copy a screenshot of your state register into the box below. As mentioned in class, use of a *reset* signal is optional.

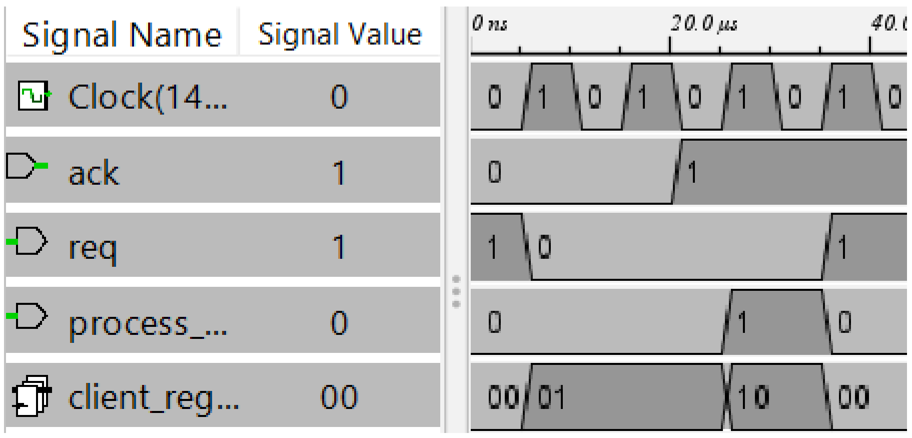


Full Client FSM
Connect the combinational logic and state register to complete the client finite state machine. Copy a screenshot of your full client FSM into the box below.



Client FSM Timing Diagram
Test your client finite state machine, demonstrating that it can transition between all states. Upon reaching WAIT_ACK, first set ack = 0 to demonstrate the FSM stays in that state for 1 clock cycle, and then change set ack = 1 to demonstrate the transition out on the next cycle. Your timing diagram should show all FSM inputs/outputs, as well as the state register's internal (current) state.

Copy a screenshot of your timing diagram into the box below.



Part 2 - Design and Test of the Server, and Client-Server Interaction
You will implement the server using the same approach as the client.

Server State Transition Table Combinational Logic

Complete the state transition table for the server FSM. A skeleton table is not provided for this problem; follow the approach from our in-class examples and the client FSM, and make sure you understand what each row/column in the table means. If you would like, you may again combine any multi-bit signals into a single bus. (This will slightly simplify the design in Logisim-evolution.)

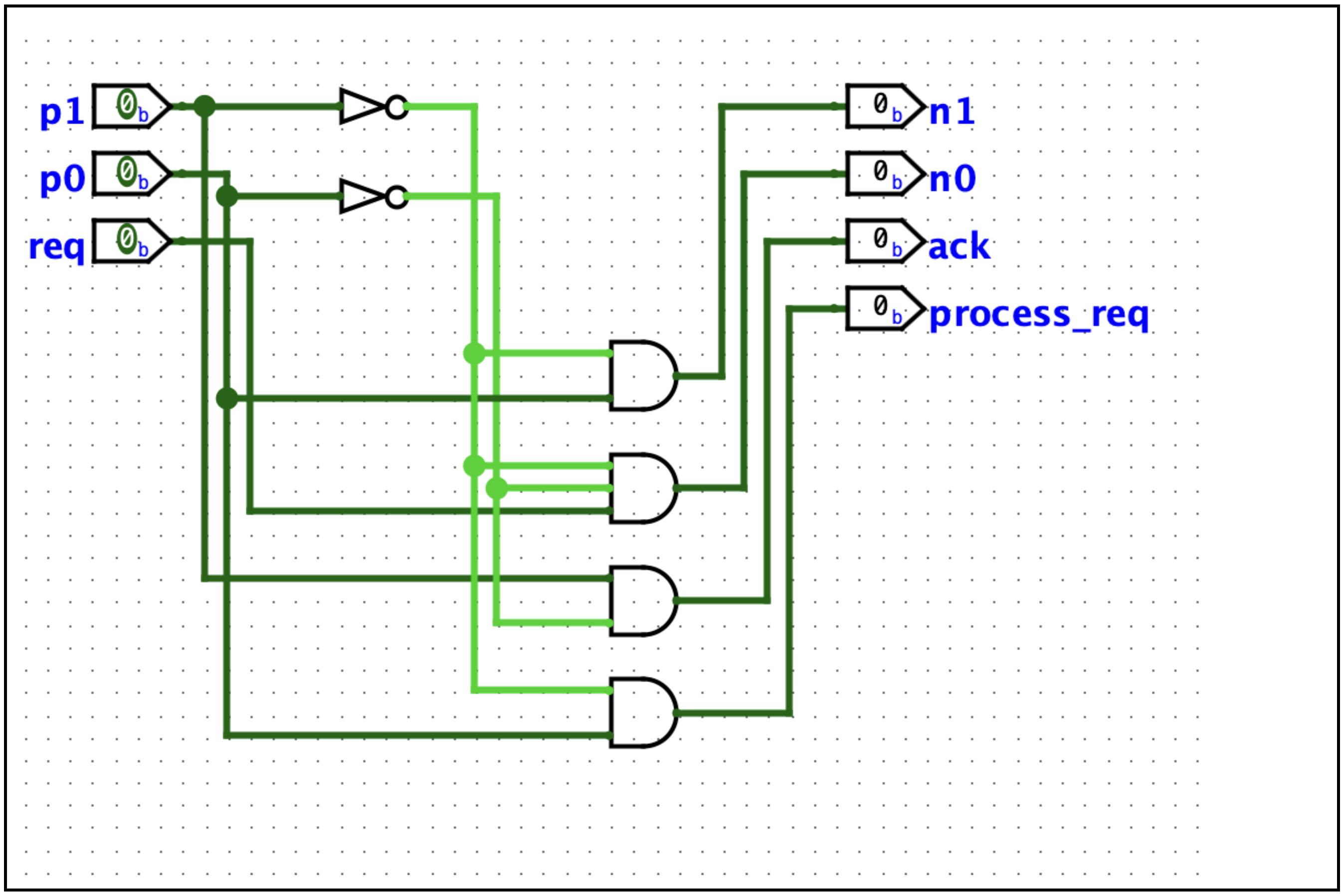
state	p1	p0	req	n1	n0	ack	process_req
WAIT_REQ	0	0	0	0	0	0	0
WAIT_REQ	0	0	1	0	1	0	0
PROCESS	0	1	0	1	0	0	1
PROCESS	0	1	1	1	0	0	1
SEND_ACK	1	0	0	0	0	1	0
SEND_ACK	1	0	1	0	0	1	0
unused	1	1	0	0	0	0	0
unused	1	1	1	0	0	0	0

Optional: This box is not worth any points, but for extra practice as we head toward the (cumulative) final exam, consider completing this section. Solutions will be provided.

Use Karnaugh maps to determine the Boolean equations for the server's combinational logic outputs in minimized sum-of-products form. Show your work in the box below, including the K-maps and the minimized sum-of-products equations extracted from each K-map.

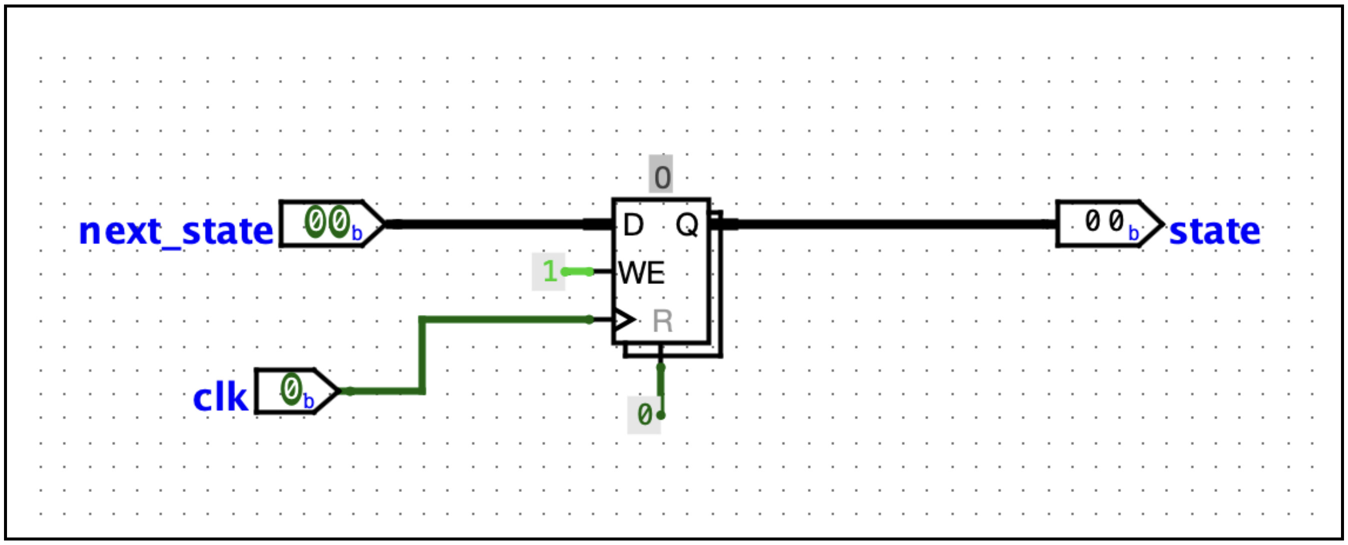
Server Combinational Logic

Create the combinational logic for the server FSM using the Combinational Analysis tool in Logisim-evolution. Copy a screenshot of your combinational logic into the box below.

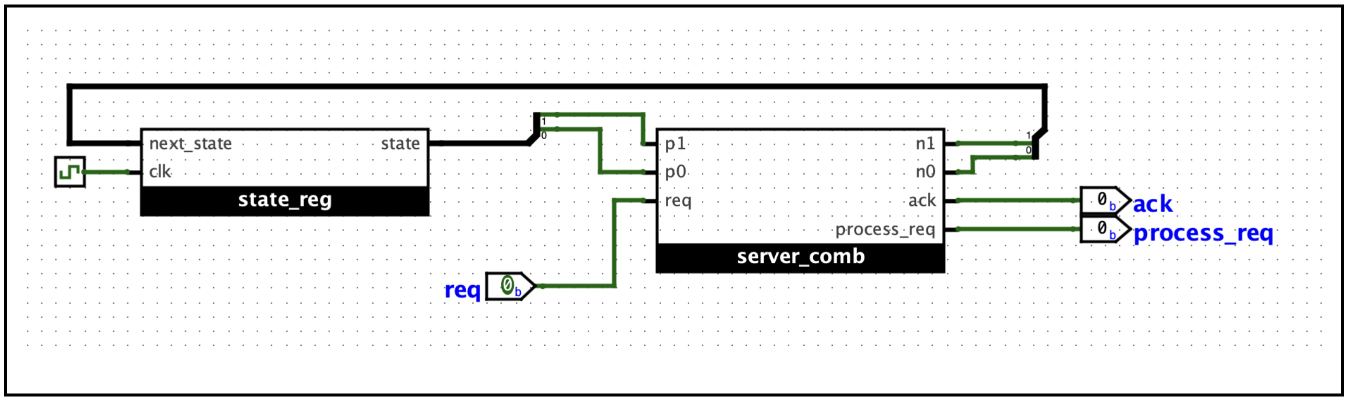


Server State Register

Create an appropriately-sized state register for the server FSM in Logisim-evolution. Copy a screenshot of your state register into the box below. As mentioned in class, use of a *reset* signal is optional. If your server state register is the same size as the client's, do *not* create a new circuit, re-use it for the server and take a screenshot of the existing state register.

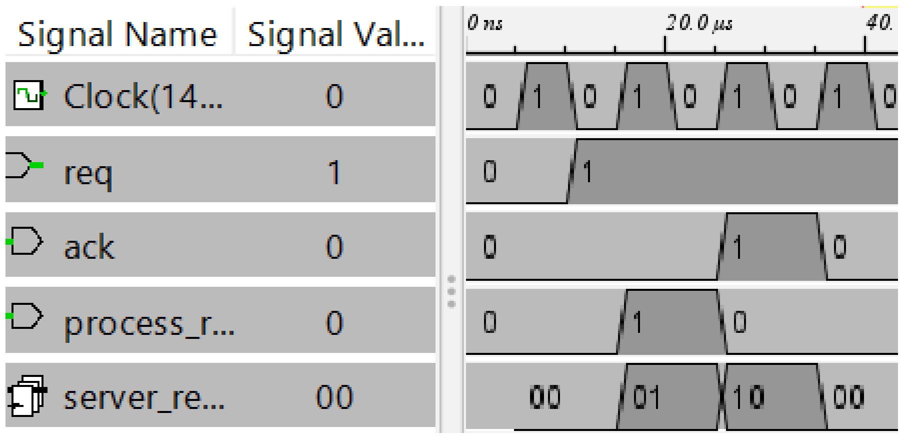


Full Server FSM
Connect the combinational logic and state register to complete the server finite state machine. Copy a screenshot of your full server FSM into the box below.

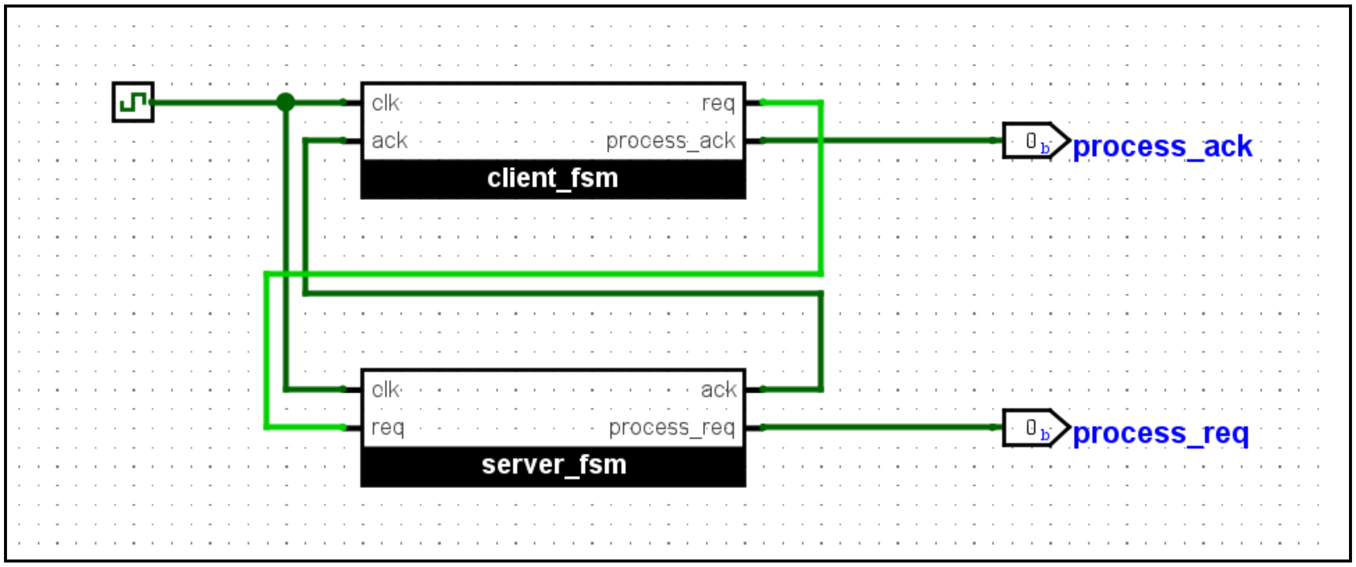


Server FSM Timing Diagram
Test your server finite state machine, demonstrating that it can transition between all states. Upon reaching WAIT_REQ, first set req = 0 to demonstrate the FSM stays in that state for 1 clock cycle, and then set req = 1 to demonstrate the transition out on the next cycle. Your timing diagram should show all FSM inputs/outputs, as well as the state register's internal (current) state.

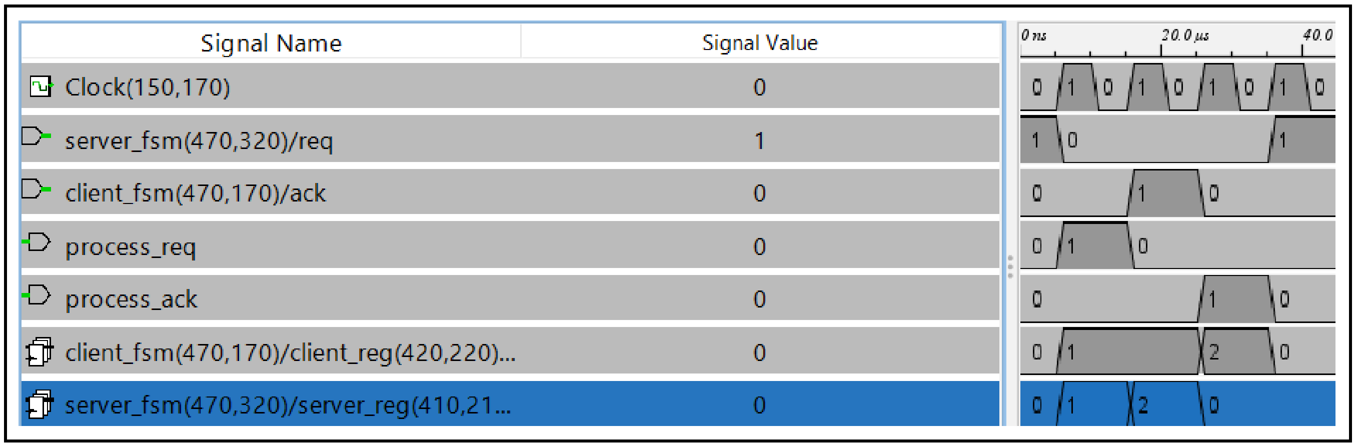
Copy a screenshot of your timing diagram into the box below:



Simulating Client-Server Interaction
The final step is to create a timing diagram to simulate the client-server interaction. Connect the client and server into a single top-level schematic called `client_server_system`. Copy a screenshot of your schematic into the box below.



Create a timing diagram that shows the client/server interaction. The client/server should each transition through all their states in your timing diagram. Include the following signals in this order: `clk`, `req`, `ack`, `process_req`, and `process_ack`, as well as the value of client's state register, and finally the server's state register. Set the radix to hexadecimal for the non-Boolean signals. Copy a screenshot of your timing diagram into the box below.



Signal Name	Signal Value
Clock(150,170)	0
server_fsm(470,320)/req	1
client_fsm(470,170)/ack	0
process_req	0
process_ack	0
client_fsm(470,170)/client_reg(420,220)...	0
server_fsm(470,320)/server_reg(410,21...	0

Deliverable Checklist

Problem 1 - Sequence Recognizer (20 points)

Deliverable	Points
Diagram	10
Existence and uniqueness for initial state	5
Existence and uniqueness for one additional state	5

Problem 2 Part 1 - Client FSM (25 points)

Deliverable	Points
State transition table	8

Combinational logic screenshot	2
State register screenshot	2
Complete client FSM	5
Timing diagram	8

Problem 2 Part 2 - Server FSM and Client-Server Interaction (35 points)

Deliverable	Points
State transition table	8
Combinational logic screenshot	2
State register screenshot	2
Complete client FSM	5
Timing diagram for server FSM in isolation	8
Complete client_server_system schematic	5
Timing diagram for client + server FSM	5