ECE 327/627
Digital Hardware Systems

Lecture 11: Midterm Review

Andrew Boutros

andrew.boutros@uwaterloo.ca

Midterm Logistics

- Midterm is on Monday, Feb 23 @ 12:30 pm
- Exam duration is 90 minutes
- Closed-book with no electronic aids (e.g., laptop, tablet)
- Allowed to use a non-programmable calculator
- One letter-size double-sided handwritten cheat sheet
  - FAQs:
    - What do you mean by handwritten?
    - Can it be handwritten using an e-pencil on a table then printed?
    - Is it the same cheat sheet we will use for the final?
    - Can I use a magnifying glass?
- Can use pen or pencil. An answer that is too light to read or smudged will be assigned a grade of zero.
- **Disclaimer:** If I covered a specific concept in this review, that doesn't mean it is more important for the midterm than other concepts I did not cover.

Main Topics Covered in the Midterm

- Content of the first 9 lectures and 5 tutorials
- Anything we discussed is fair game!
- Main topics include but are not limited to:
  - SystemVerilog basics
  - Finite-state machines
  - Pipelining
  - Retiming
  - Latency-insensitive design

SystemVerilog Basics Recap

```
module <module_name> # (
    1 Module parameters
)
(
    2 Definition of module ports
);

3 Declaration of signals

4 Procedural blocks to describe some behavior of the module

5 Continuous assignments & instantiation of subcomponents

endmodule
```

"Compile-time" parameters to instantiate different copies of the module with different configurations

The module's interface to the outside world. Ports must define bitwidth & direction and they can be arrays

The wires and registers used later in describing the module behavior

always_ff → sequential logic or always_comb → combinational logic

Assign statements → combinational logic

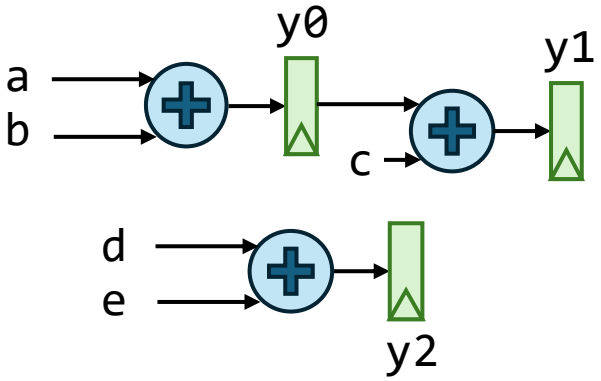
Blocking & Non-Blocking Assignments

What is the difference between the hardware circuit generated by these two snippets of code?

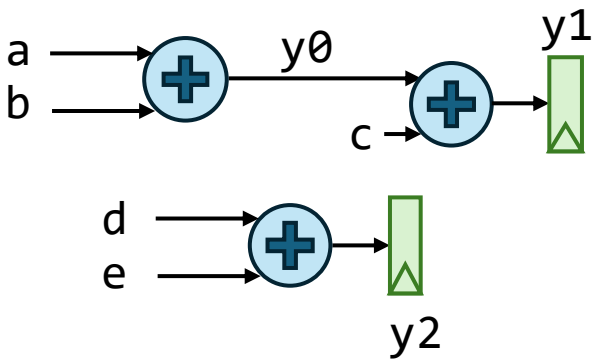
```
always_ff @ (posedge clk) begin
     y0 <= a + b;
     y1 <= y0 + c;
     y2 <= d + e;
end
```

```
always_ff @ (posedge clk) begin
     y0 = a + b;
     y1 = y0 + c;
     y2 = d + e;
end
```

All assignments are done concurrently on the positive edge of the clock.
Each non-blocking assignment translates to a register in hardware.



Blocking assignments describe behavior in sequential semantics.
The final assignments of the behavior translate to registers in hardware.



Bitwidths in SystemVerilog

- Since SystemVerilog is not strongly typed, we need to carefully reason about the bitwidth of each logic signal (wire or register)
- Narrower than needed bitwidth → can overflow resulting in functional bugs
- Wider than needed bitwidth → optimized away by the synthesis engine if not used, but can result in generation of unnecessary hardware in some cases
- To know the appropriate bitwidth for the output of a datapath:
  - Determine the range of input values
  - How does that translate to output values?
  - If maximum output value is N, Output bitwidth = $\lfloor log_2(N) \rfloor + 1$
  - Example: y = a * x * x + b * x + c, where all inputs are 8 bits
    - x, a, b, c can take any value from 0 to 255
    - y value can range from 0 to 255*255*255+255*255+255 = 16,646,655
    - Floor(log2(16,646,655)) + 1 = 24 bits

Other Important Concepts

- If-else & case statements → Multiplexers
  - Since if-else conditions can be non-mutually exclusive, order determines priority, which determines the order of cascaded MUXes
  - Missing conditions translate to feedback from outputs to inputs
    - In combinational circuit → latch

- For loops in procedural blocks
  - To describe behavior that is easy and concise to describe as a loop
  - Generated hardware is parallel → no notion of iteration

- Generate for loops
  - To instantiate many copies of a sub-module without repeating code

Practice Question #1

Design a circuit that receives a serial stream of bits (i.e., 1b per cycle) and produces an output flag that indicates whether the circuit has received more zeros or more ones so far. The output of the circuit is registered.

- Flag = 1'b0, when received zeros >= received ones
- Flag = 1'b1, when received zeros < received ones

Hint: You can assume that the gap between the number of received ones and zeros will not grow beyond 255.

Example questions:

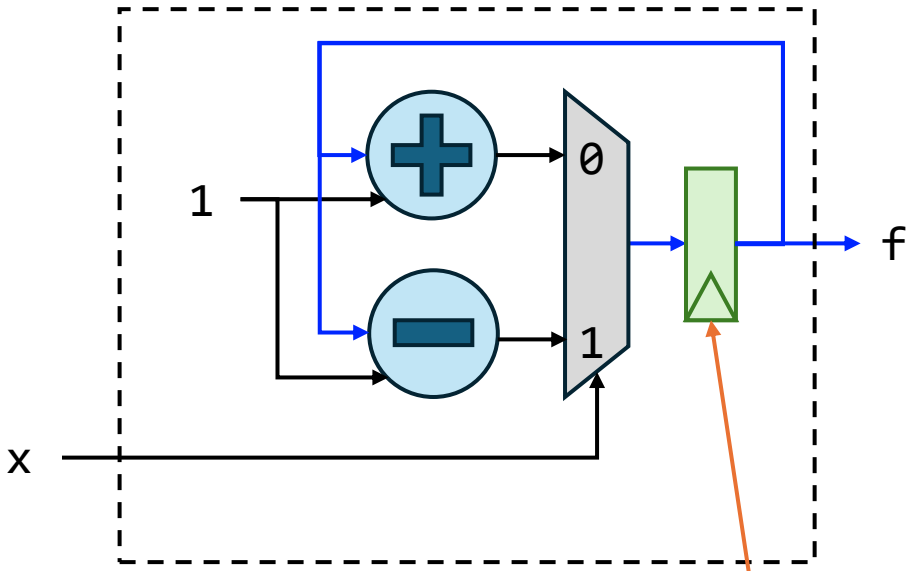
(a) Draw circuit diagram.

(b) Write SystemVerilog code to implement it.

(c) Complete a given waveform.

(d) Write a testbench that exercises a specific scenario.

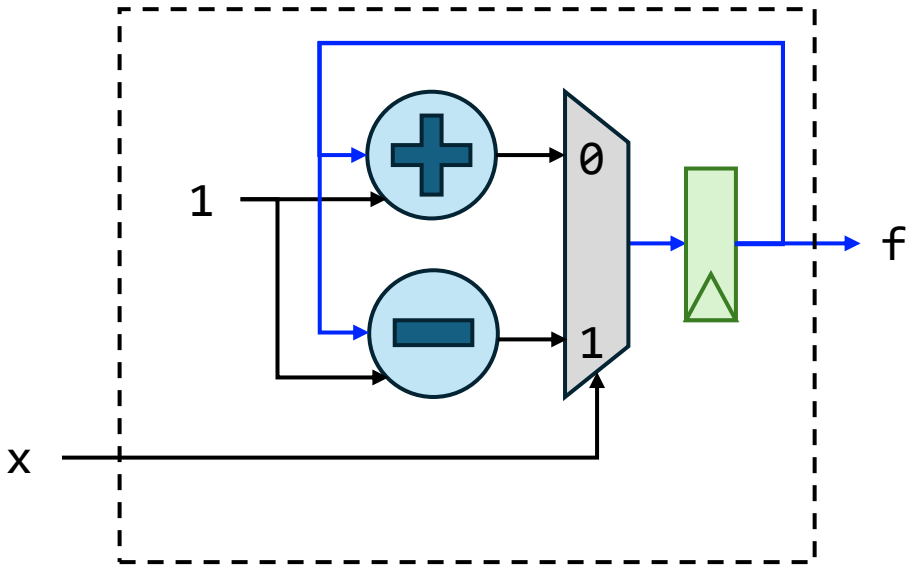
Practice Question #1



The output flag can simply be the sign bit of the register. If negative, MSB will be 1'b1 (which means received ones > received zeros). If positive or zero, MSB will be 1'b0 (which means received zeros >= received ones).

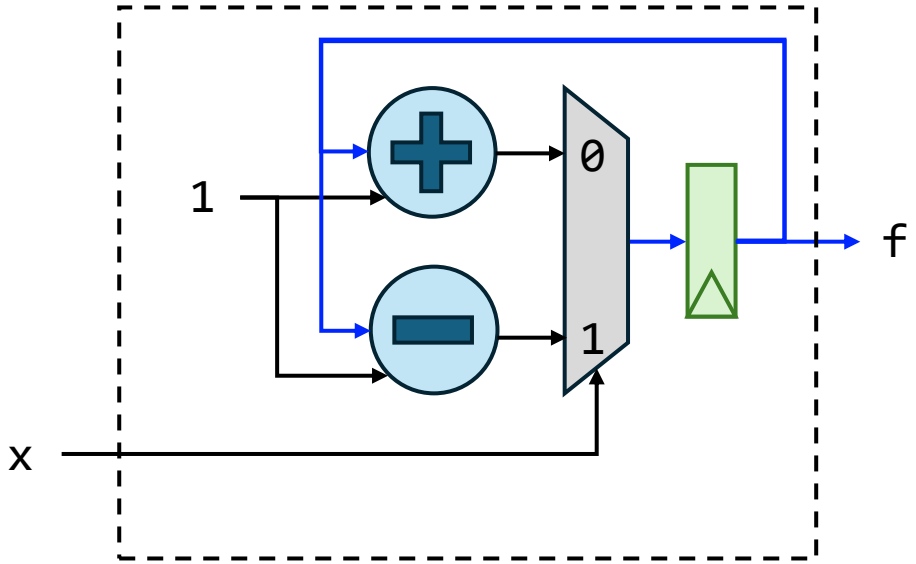
If difference between number of zeros and ones cannot grow beyond 255, then this register can have values ranging from -255 to 255 → 9 bits

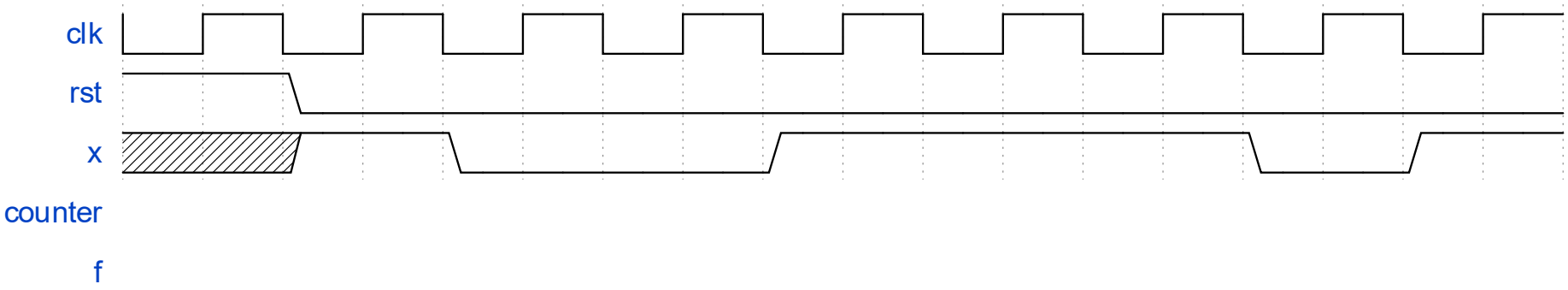
Practice Question #1



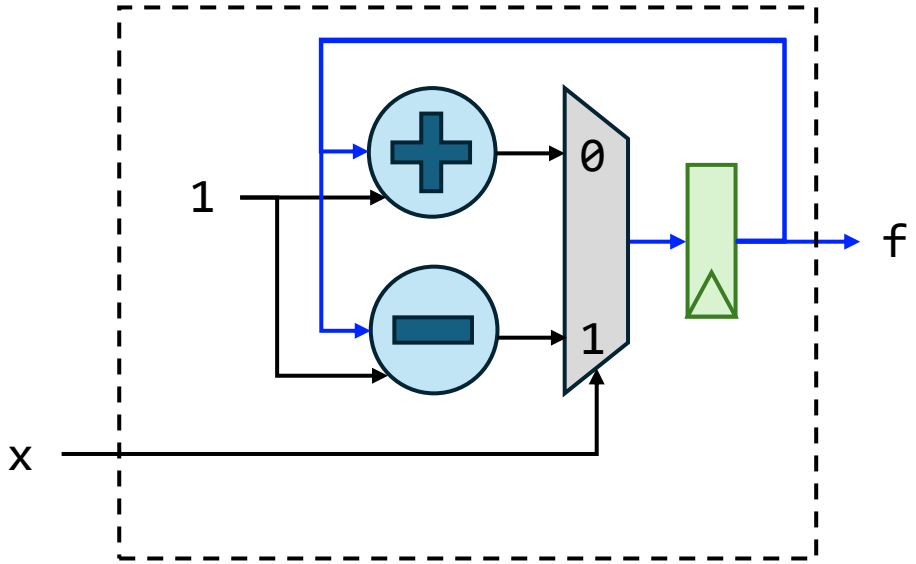

```
module detector # (
  parameter N = 9
)(
  input  clk,
  input  rst,
  input  x,
  output f
);

logic signed [N-1:0] counter;

always_ff @ (posedge clk) begin
  if (rst) begin
    counter <= 'd0;
  end else begin
    if (x) counter <= counter - 1;
    else counter <= counter + 1;
  end
end

assign f = counter[N-1];

endmodule
```

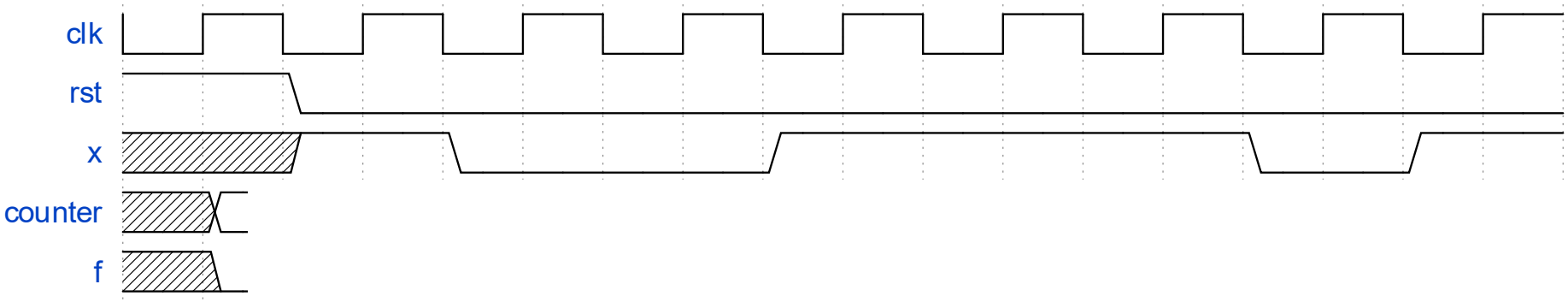
Practice Question #1



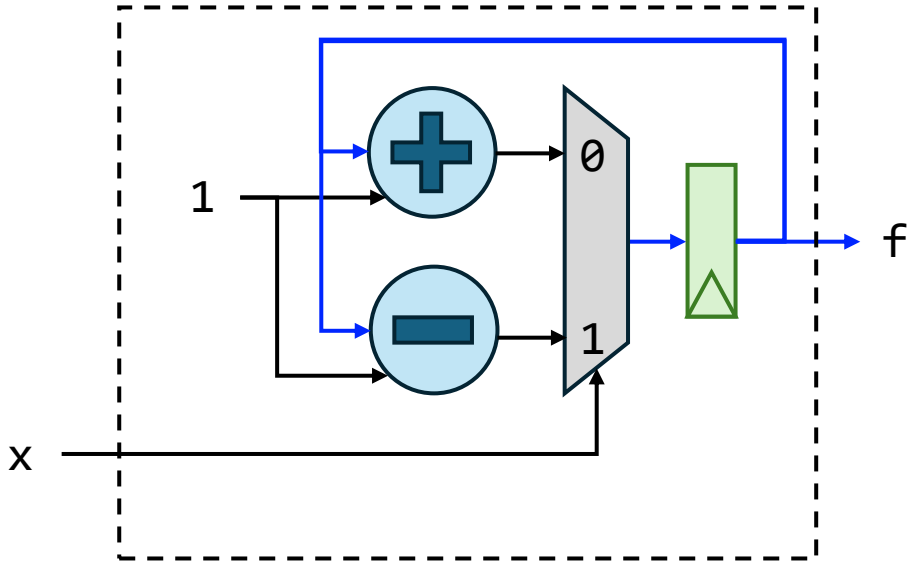


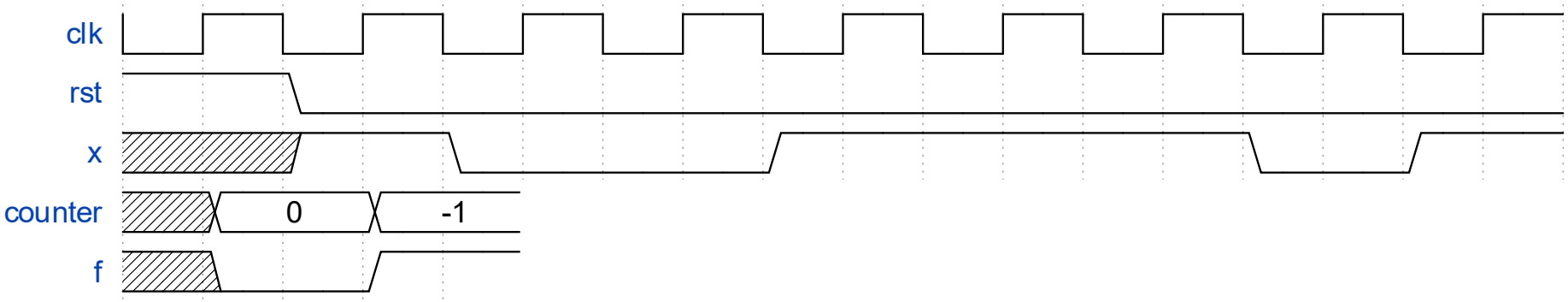
Practice Question #1



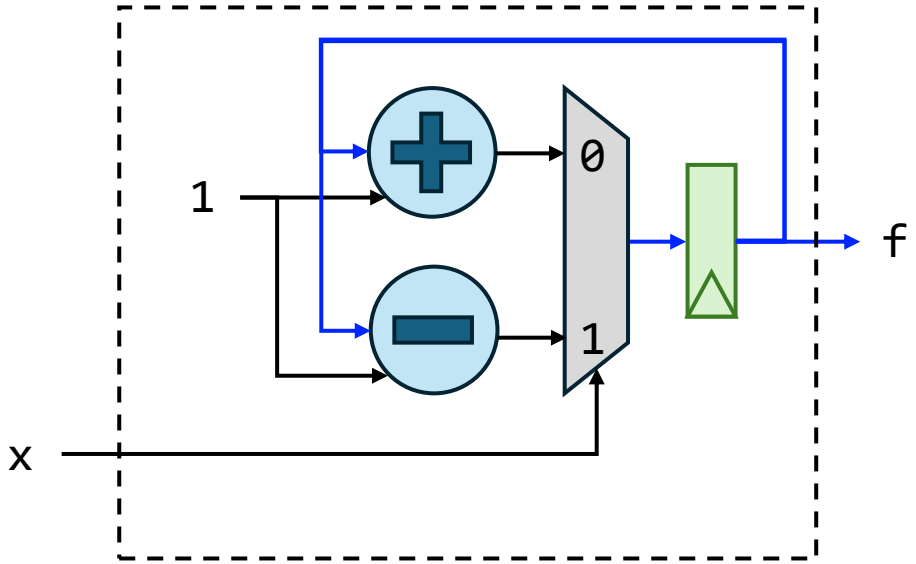


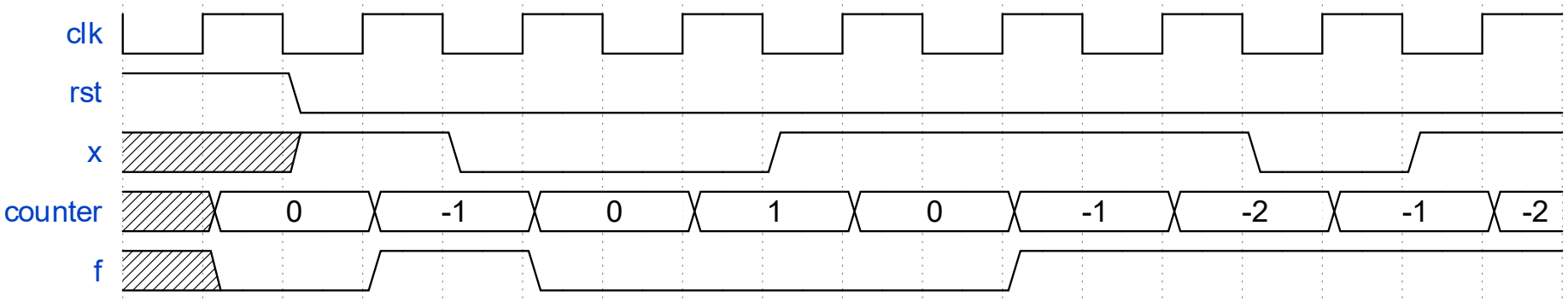
Practice Question #1



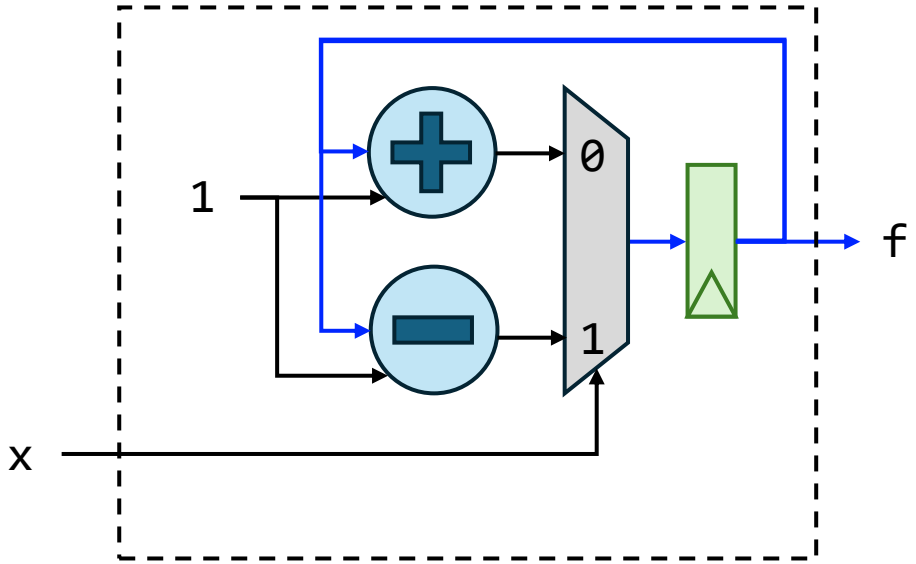


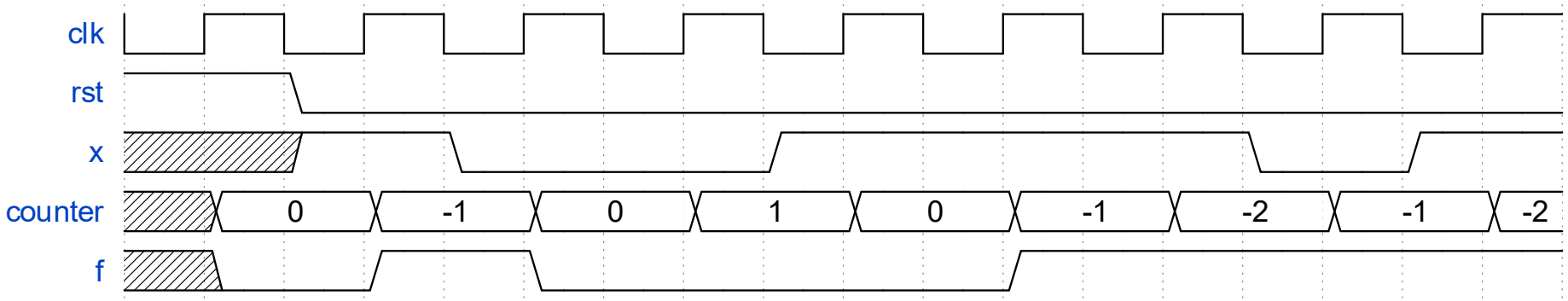
Practice Question #1





Practice Question #1



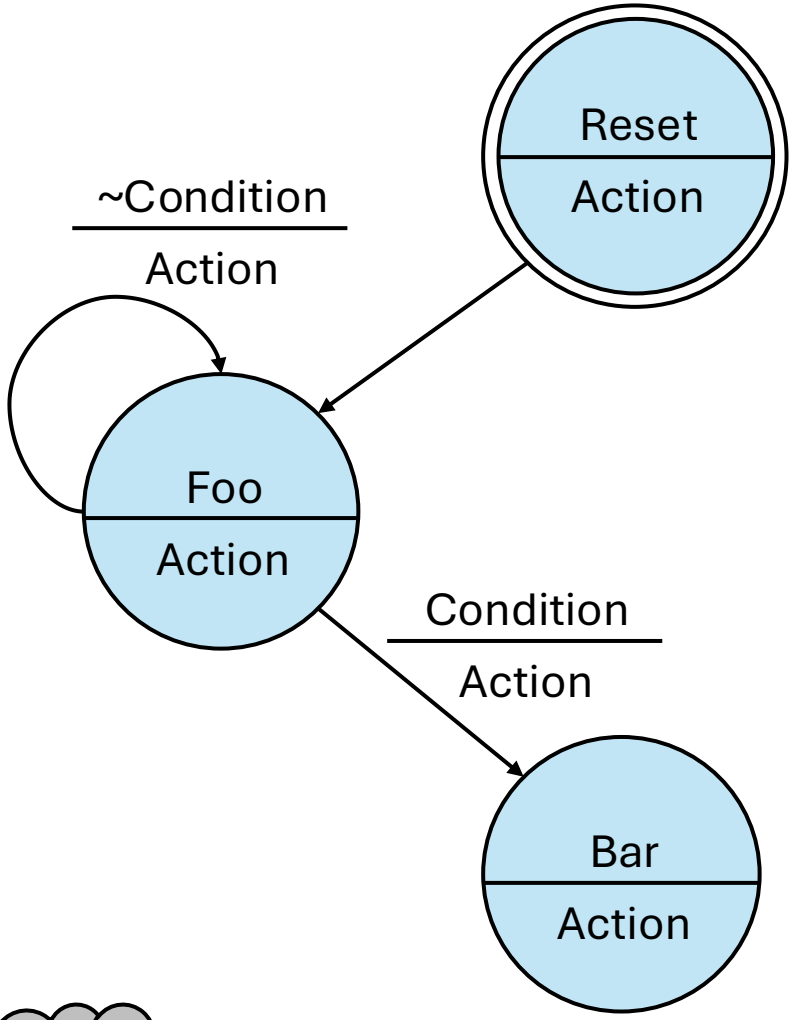


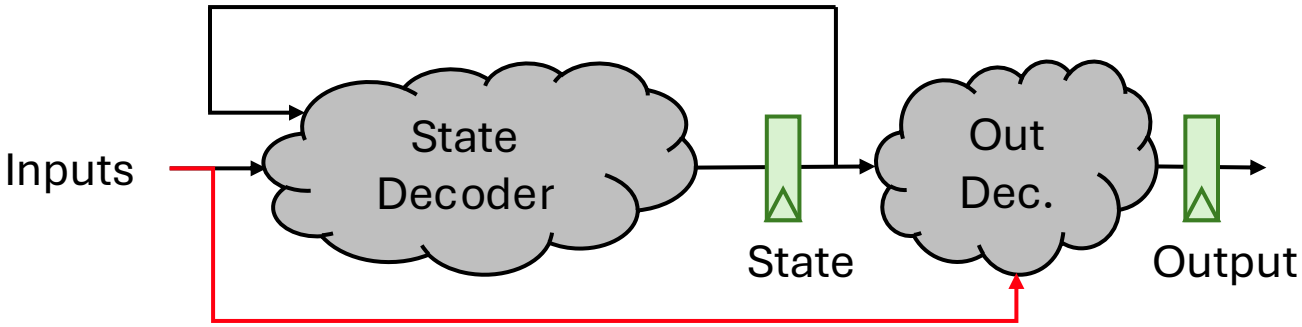
Other ideas for practicing:

- *Implement an N-bit shift add multiplier*
- *Implement a two-digit binary coded decimal (BCD) counter*

Finite-State Machines

- An FSM consists of:
  - States
  - Transitions between states
  - Actions (in state or on transitions)
- FSM SystemVerilog code template:
  - Declarations of states, regs, wires
  - always_ff → for state/output regs
  - always_comb with case statement → for state and output decoders



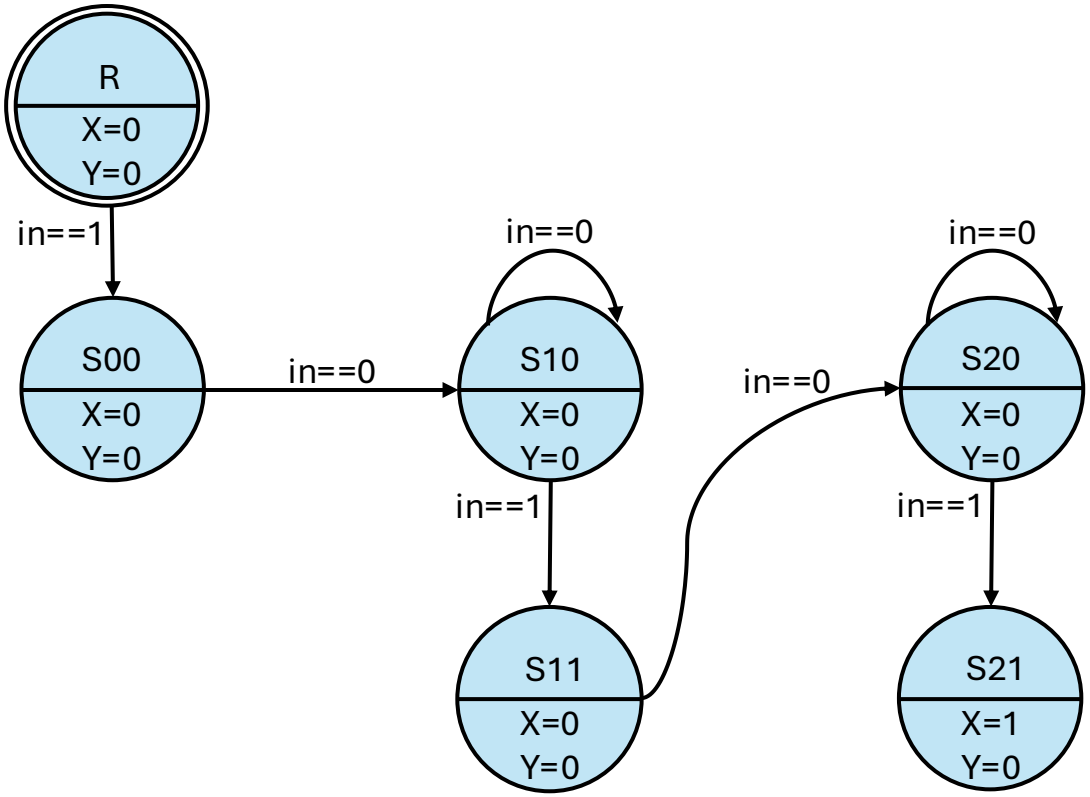


Inputs can be used by output decoder to implement transition actions

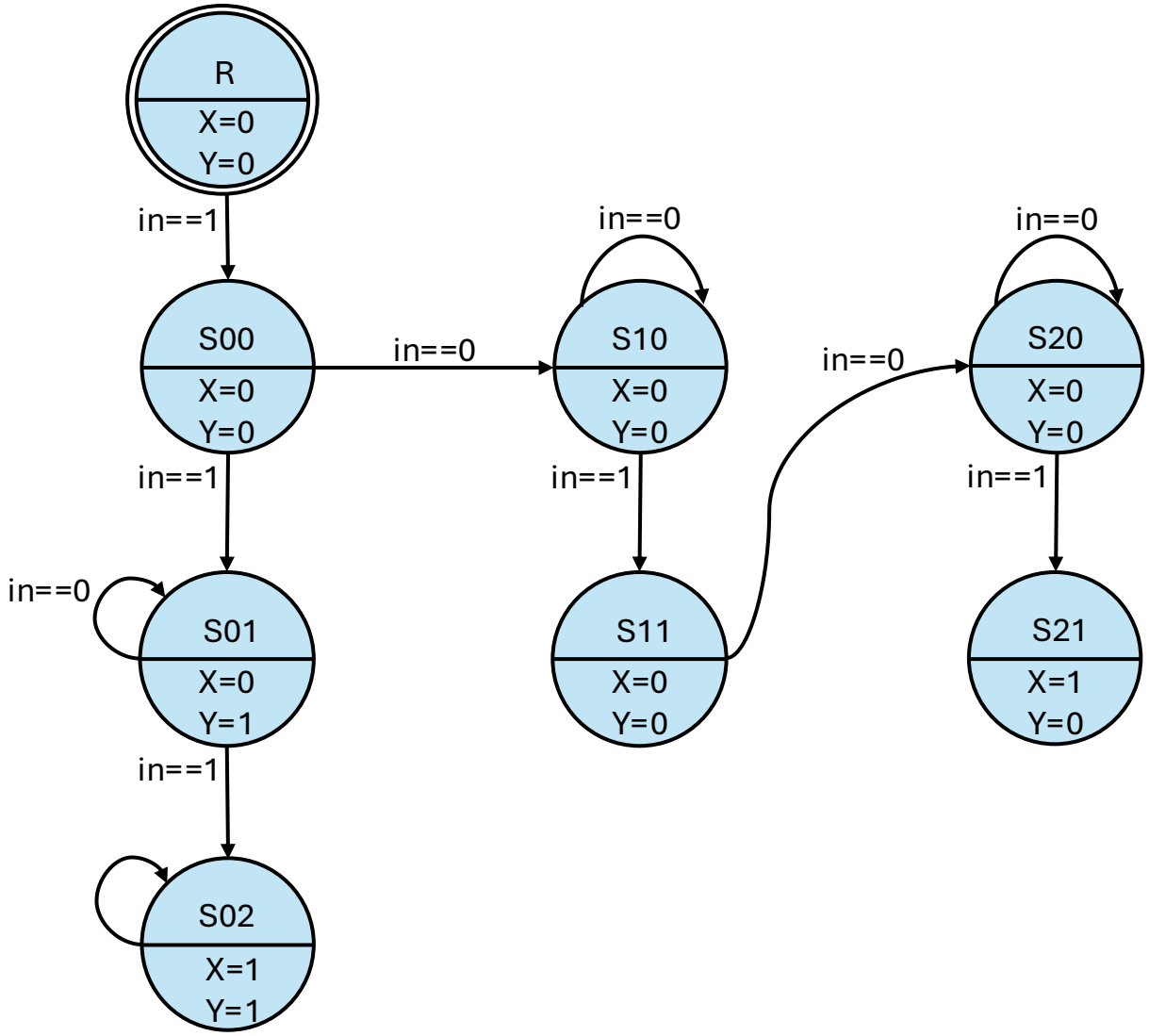
Practice Question #2

Design an FSM-based system with a single 1-bit input and two 1-bit outputs (X and Y). X is set to high if three input 1s are received (not necessarily consecutive). Y is set to high if two consecutive 1s are received.

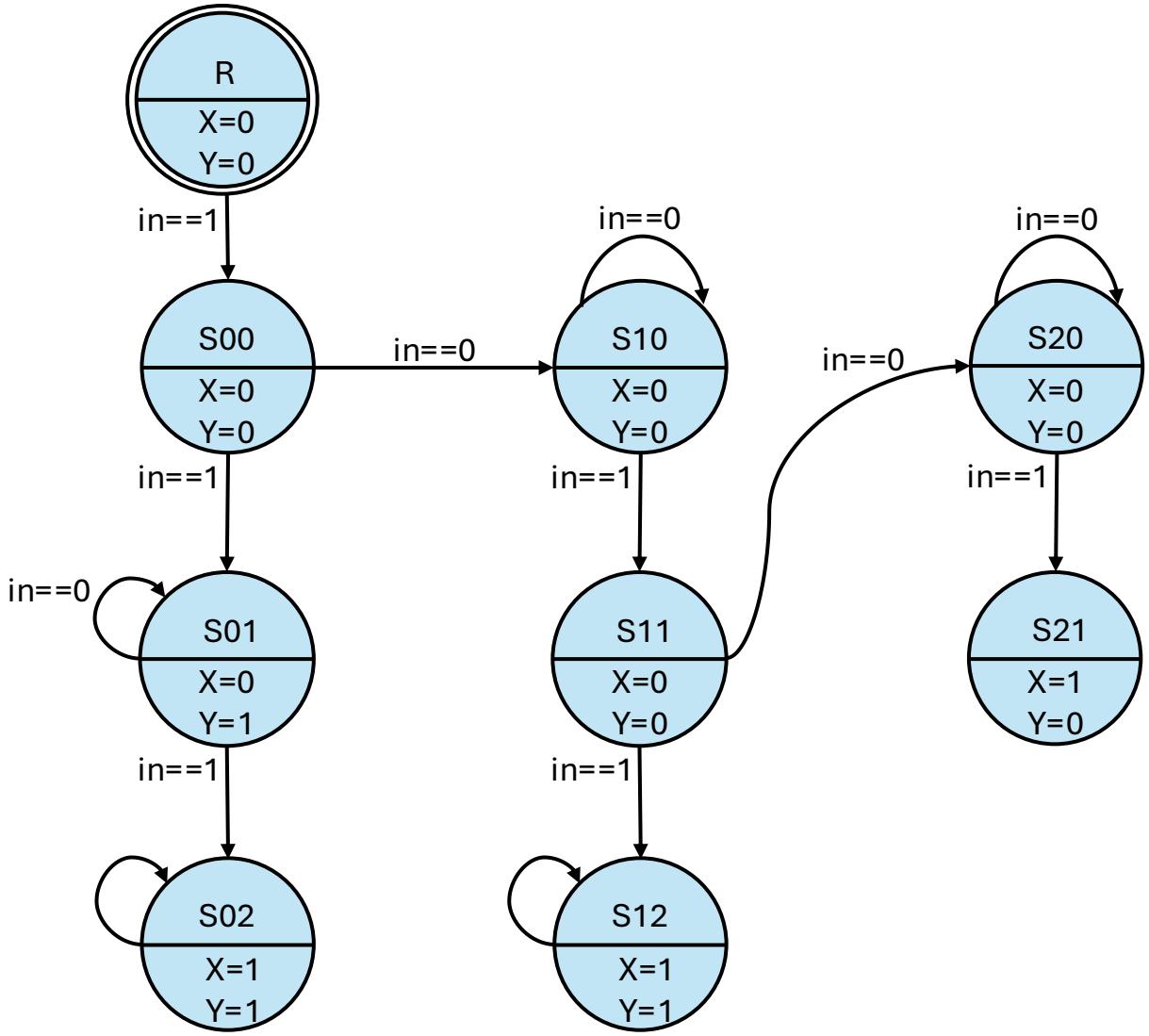
Practice Question #2



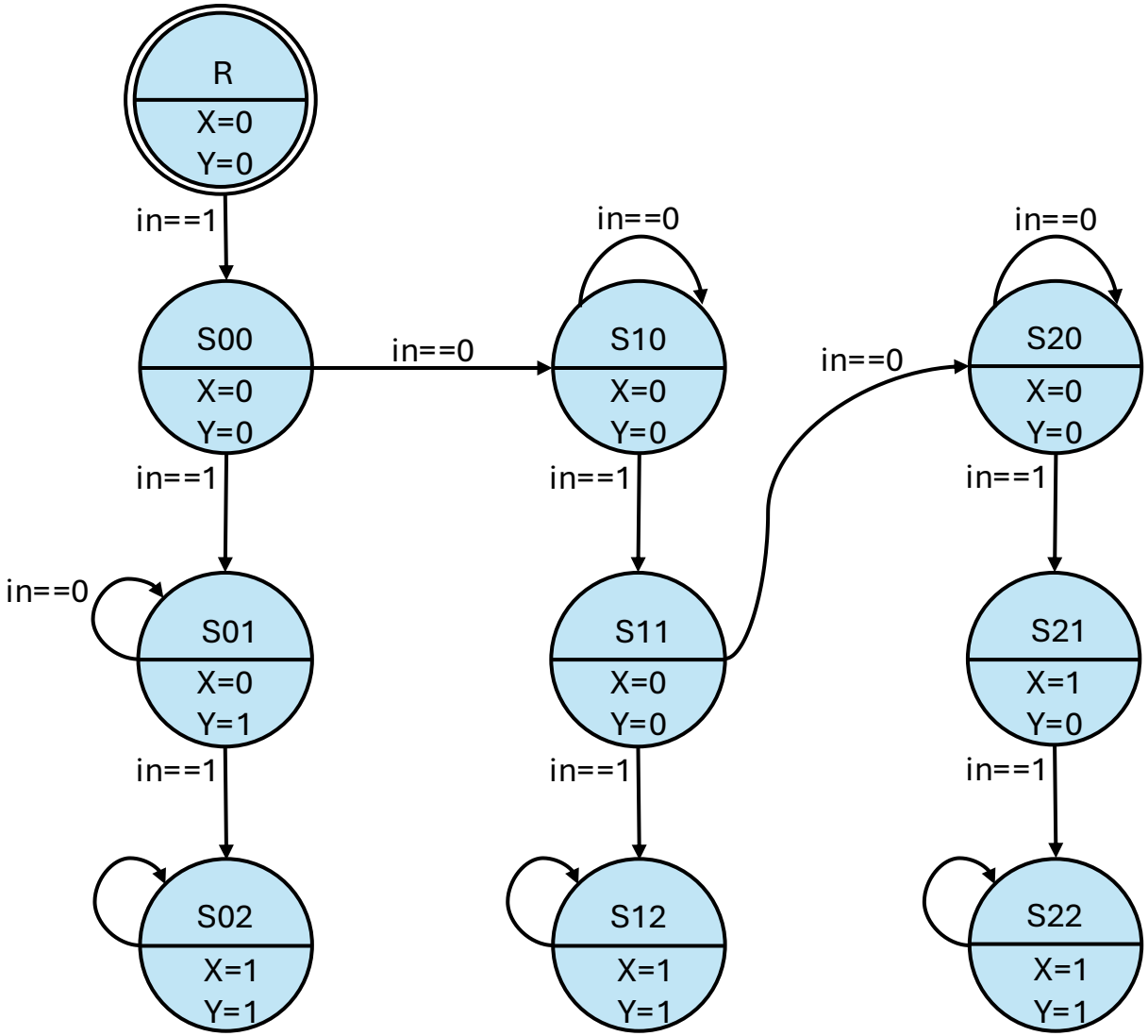
Practice Question #2



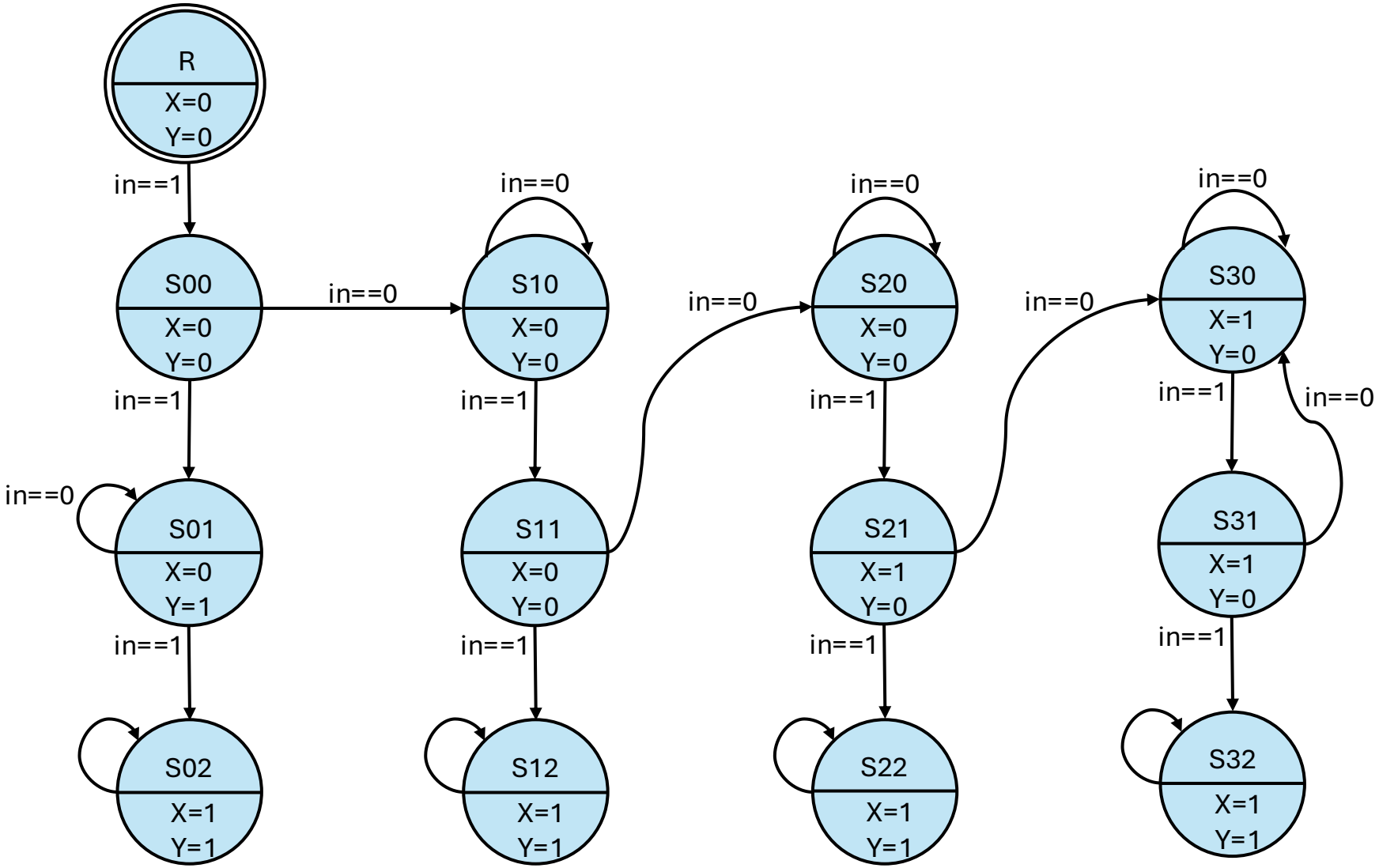
Practice Question #2



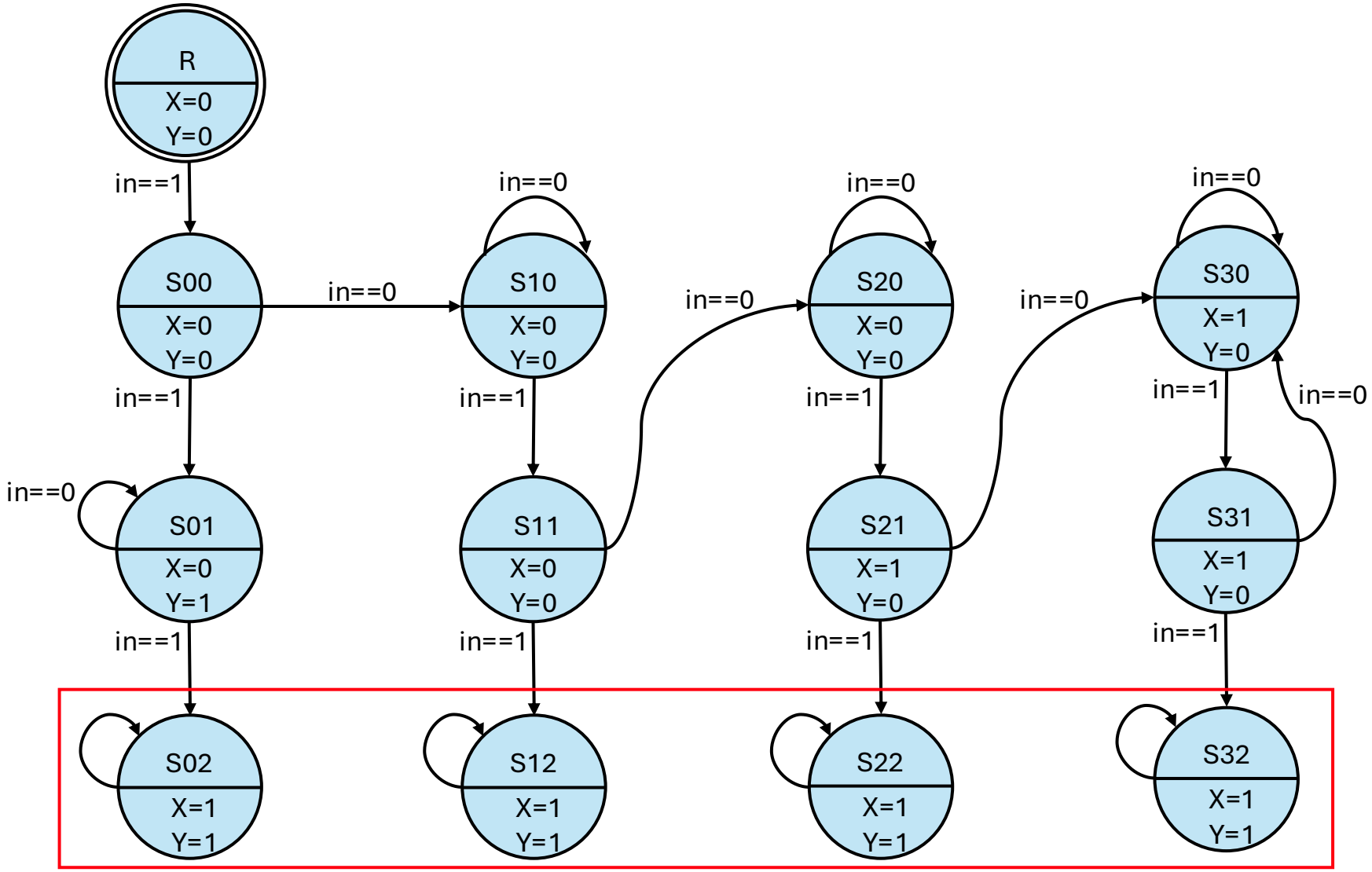
Practice Question #2



Practice Question #2

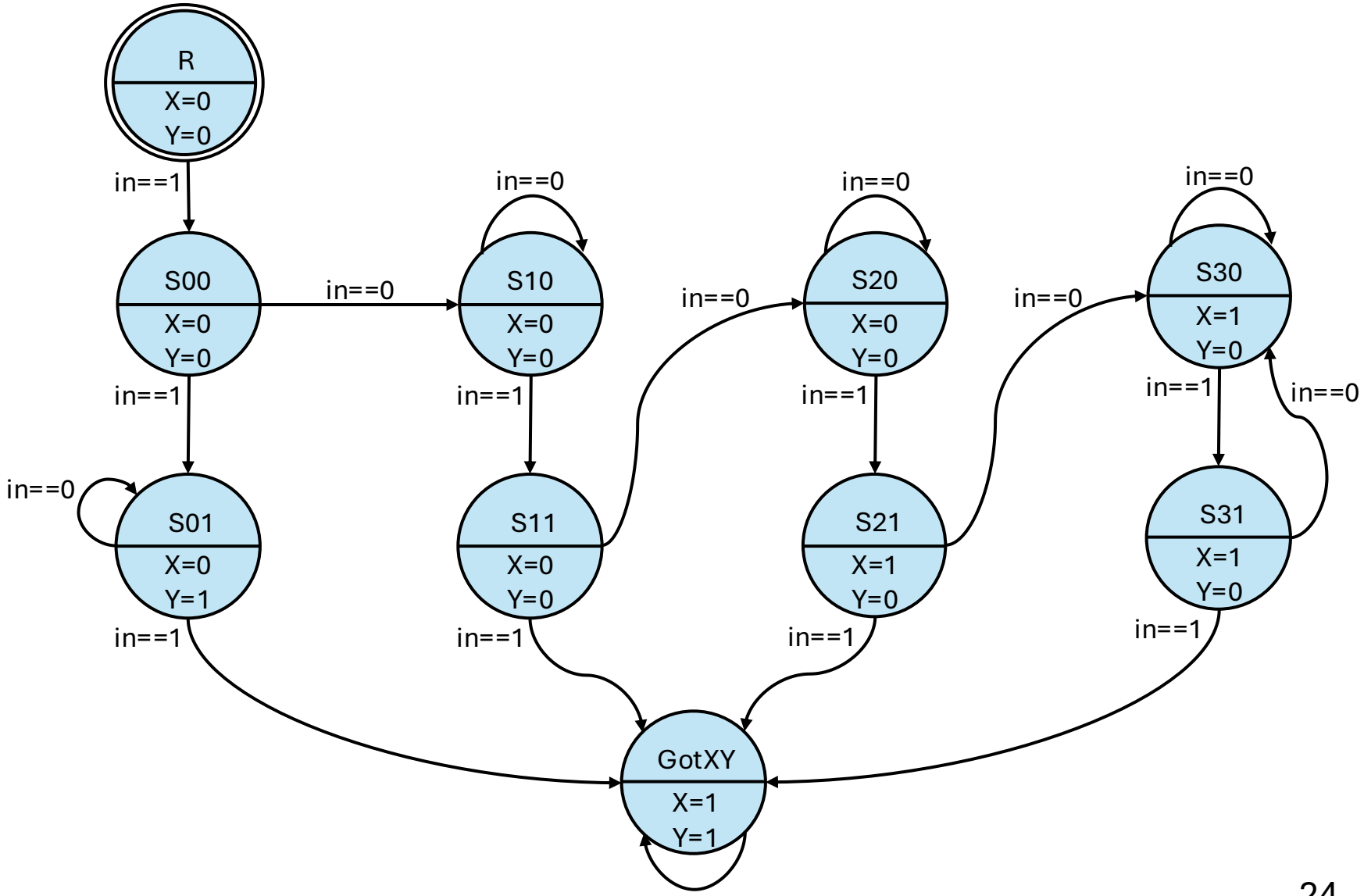


Practice Question #2

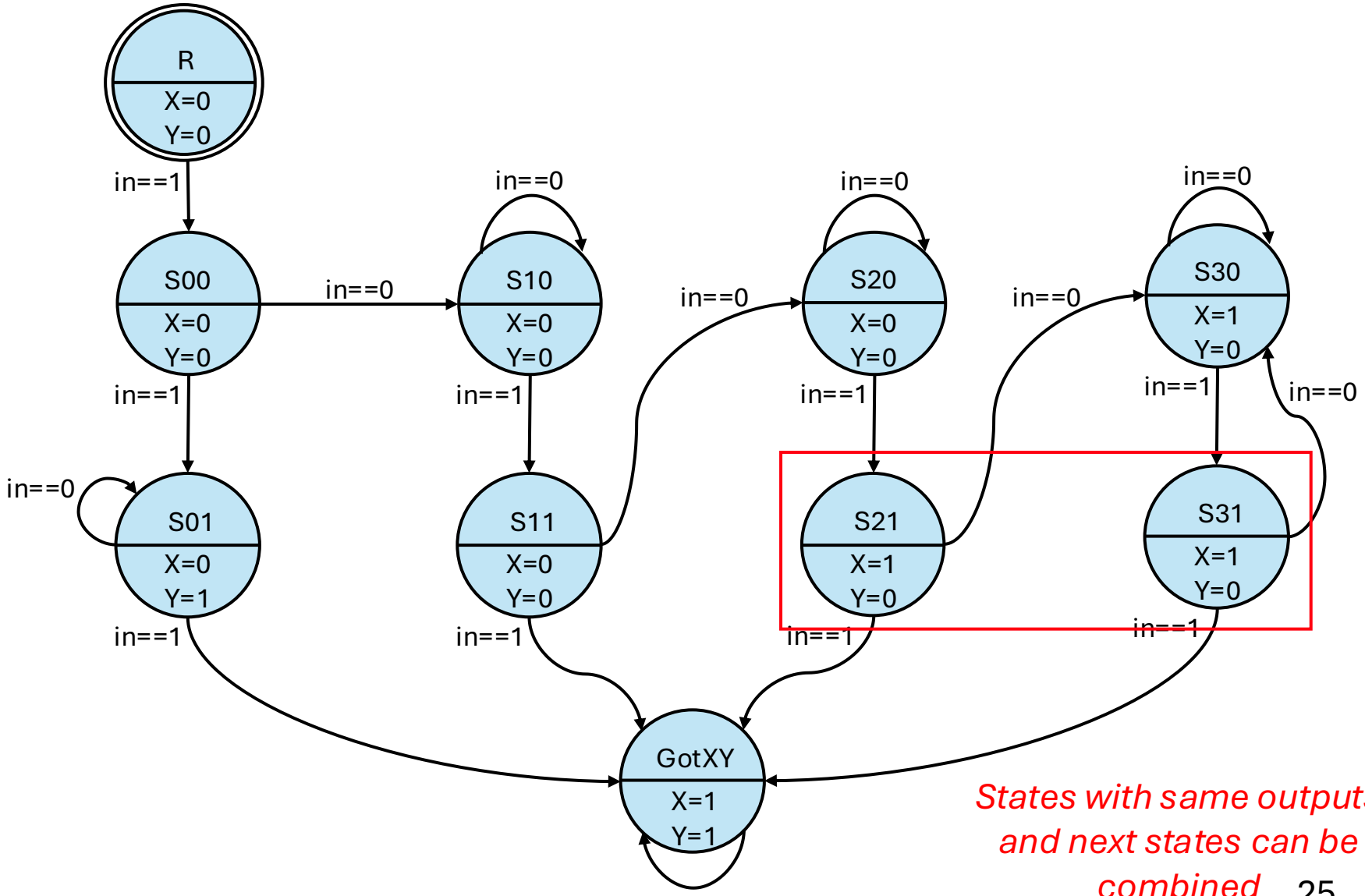


States with same outputs and next states can be combined

Practice Question #2

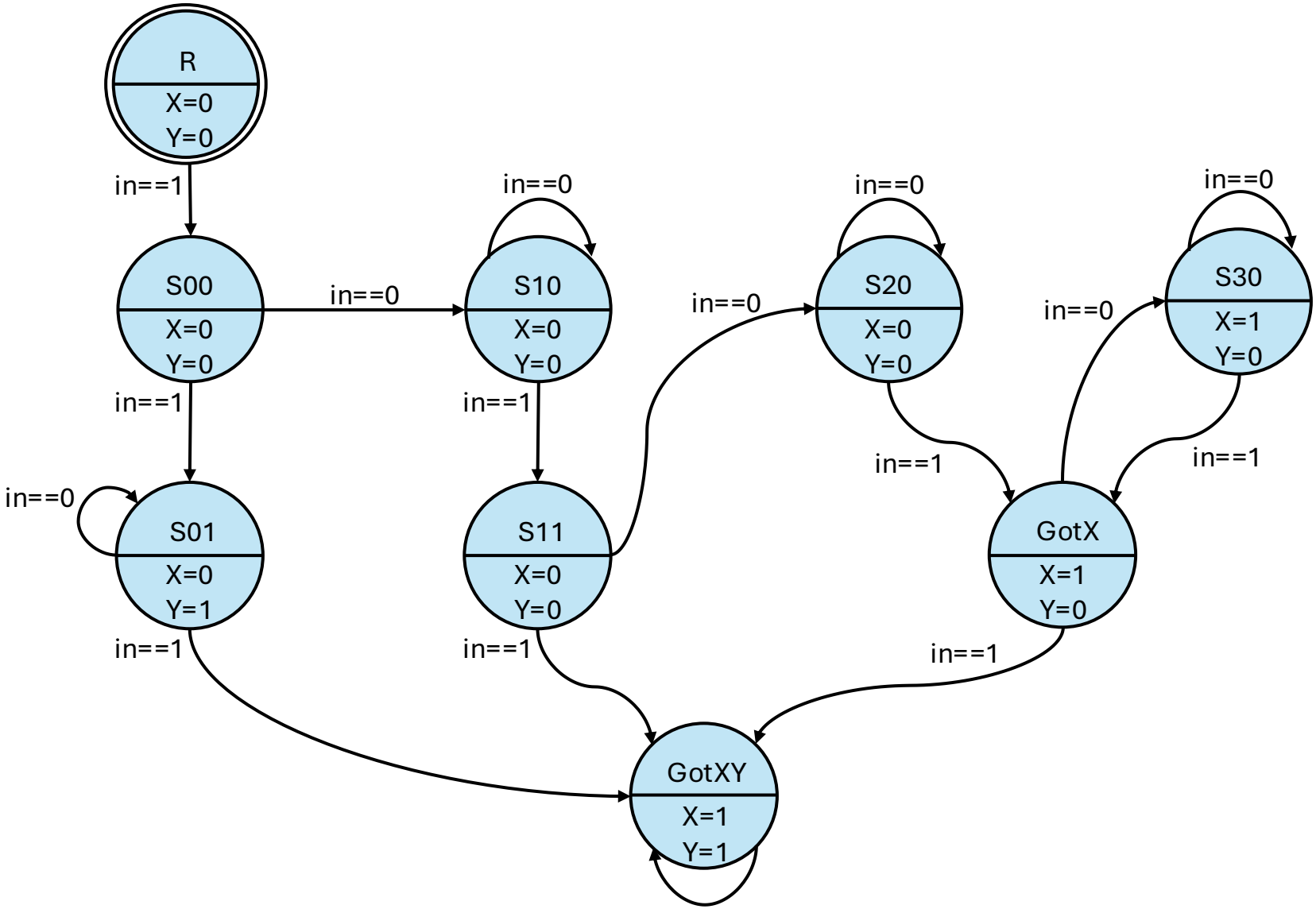


Practice Question #2



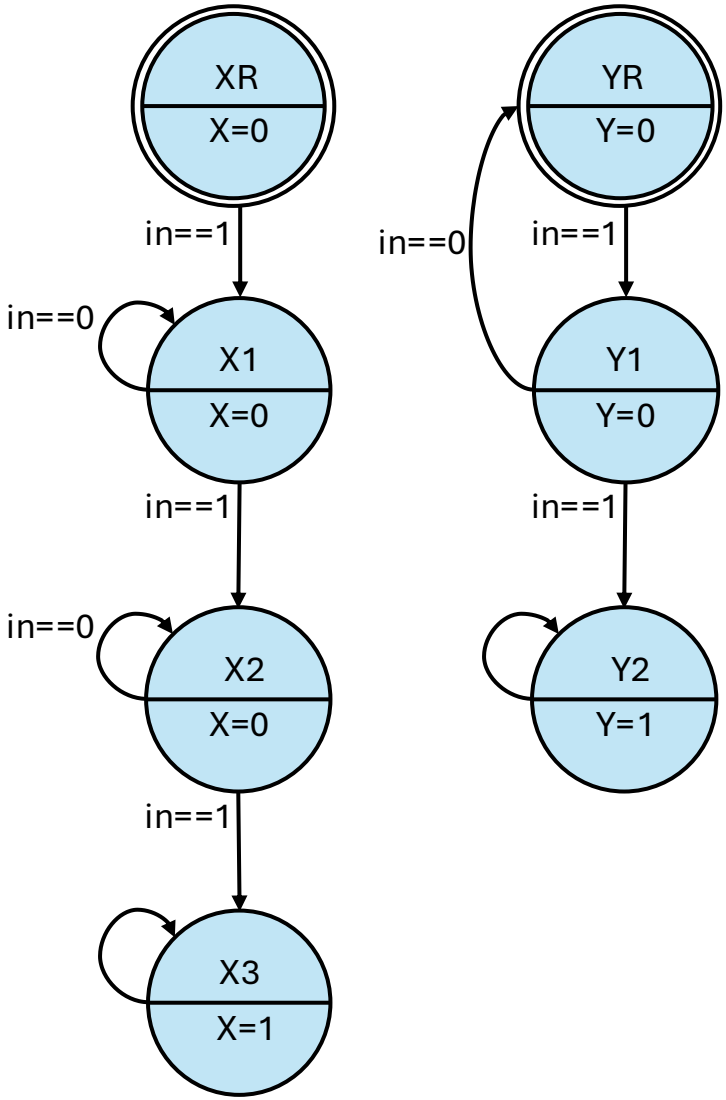
States with same outputs and next states can be combined

Practice Question #2



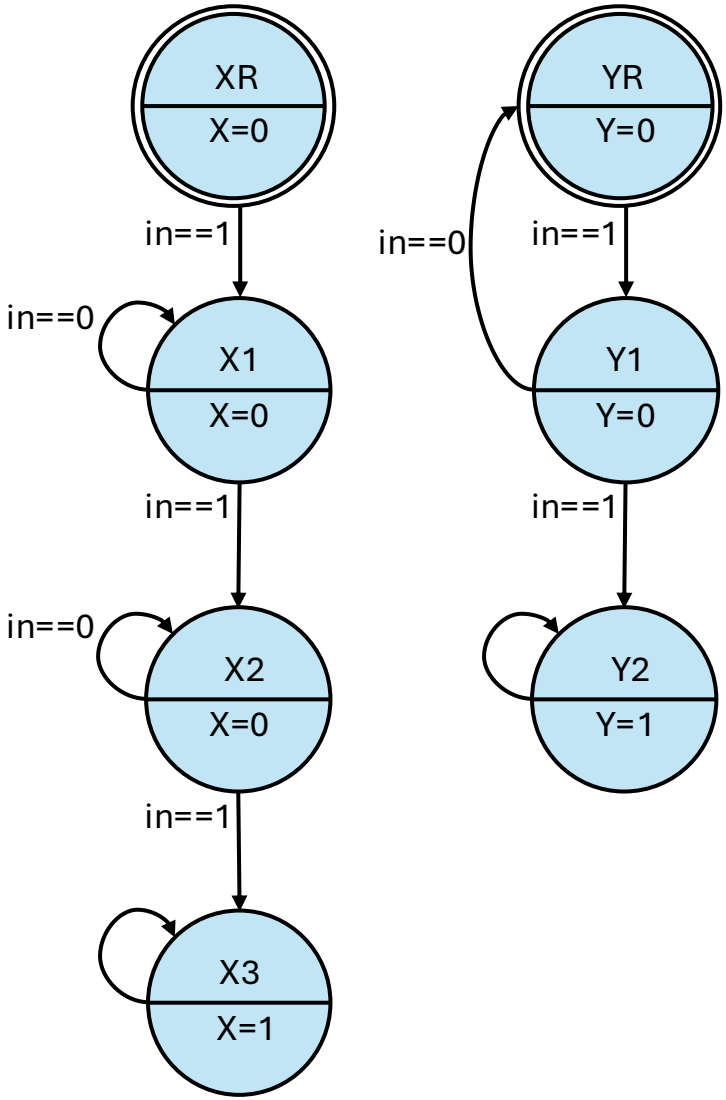
Practice Question #2

Sometimes solutions become very simple when a problem is broken into two FSMs working in parallel ☺



Practice Question #2

Sometimes solutions become very simple when a problem is broken into two FSMs working in parallel ☺



Other ideas for practicing:

- *Keypad lock*

Pipelining

For a given circuit, more pipelining means …

Critical Path ↓

Clock Period ↓

Frequency ↑

No. of Cycles ↑

Throughput = Operations per cycle x Frequency

Latency = No. of Cycles x Clock Period

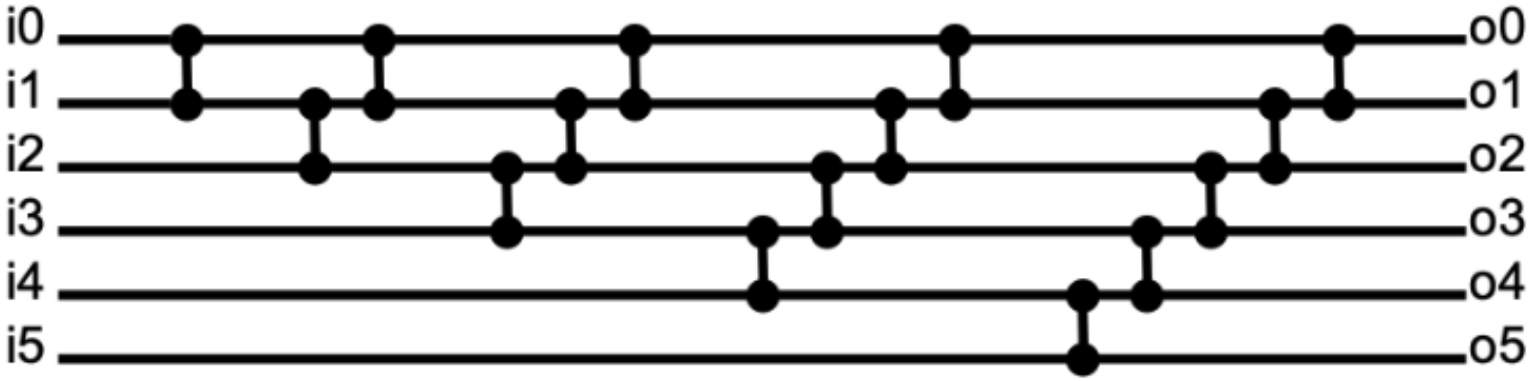
Silicon Area ↑

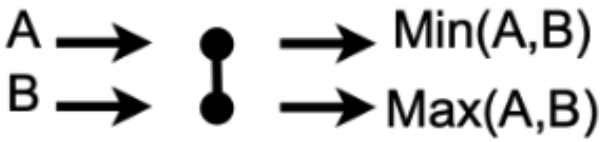
Power ↑

Clock Distribution Network Design Complexity ↑

Practice Problem #3

A circuit takes as inputs 6 numbers (i0 – i5) and sorts them using compare-and-swap modules as shown in the diagrams below.





Practice Problem #3

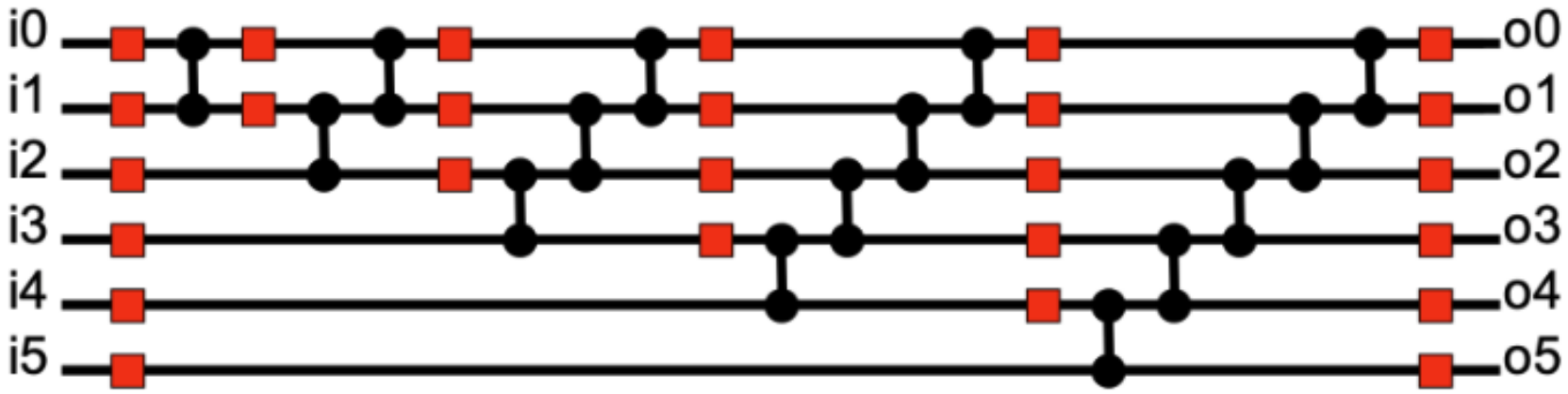
The circuit was pipelined as shown in the following diagram.

(a) What is the latency of the circuit in cycles?

(b) What is the critical path of the circuit?

(c) What is the throughput of the circuit in operations/cycle? Consider sorting of 6 numbers is counted as 1 operation?

(d) How can the circuit be pipelined to maximize throughput without decreasing frequency vs. the current version?



Practice Problem #3

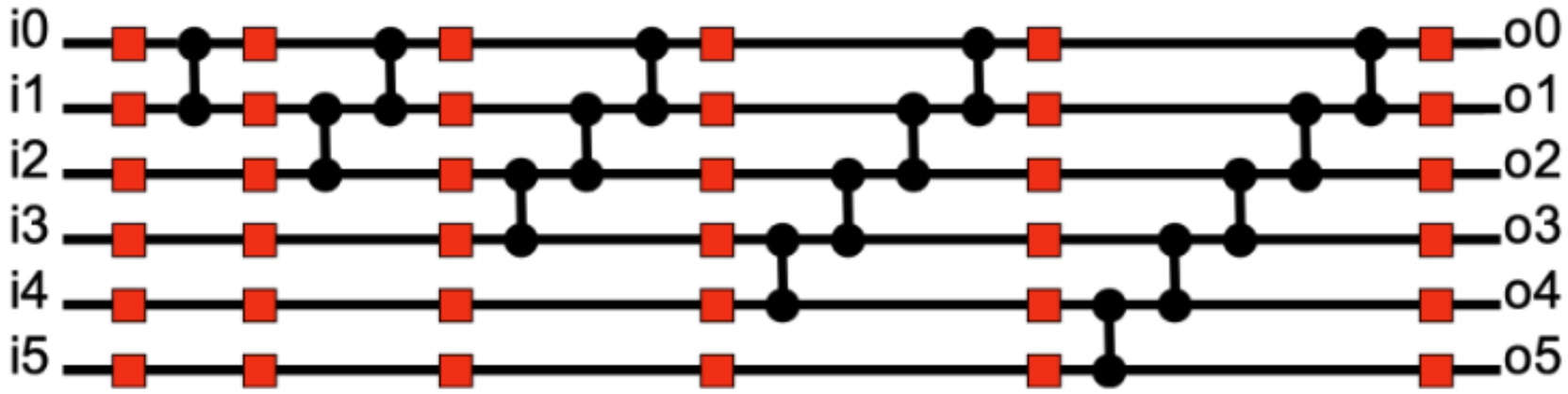
The circuit was pipelined as shown in the following diagram.

(a) What is the latency of the circuit in cycles?

(b) What is the critical path of the circuit?

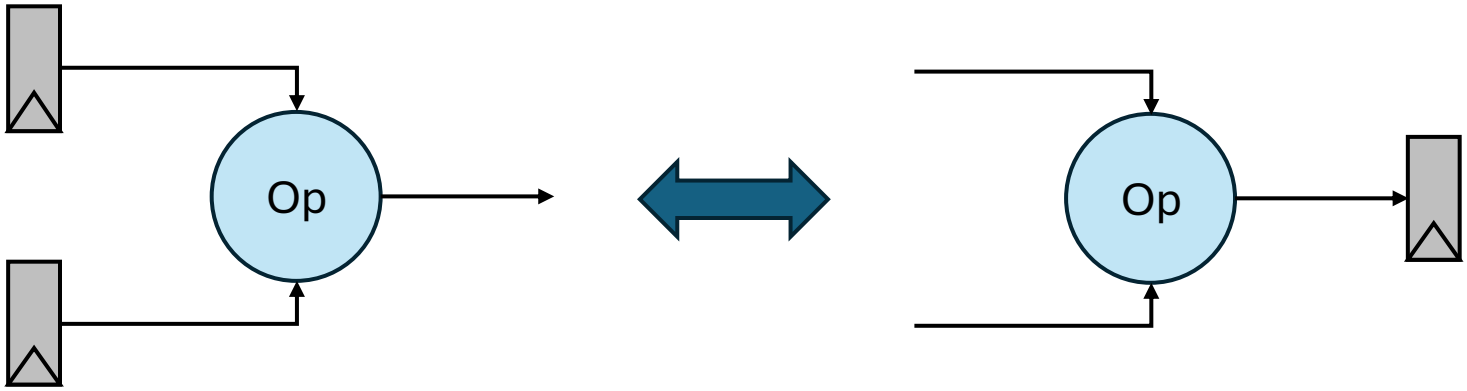
(c) What is the throughput of the circuit in operations/cycle? Consider sorting of 6 numbers is counted as 1 operation?

(d) How can the circuit be pipelined to maximize throughput without decreasing frequency vs. the current version?



Retiming

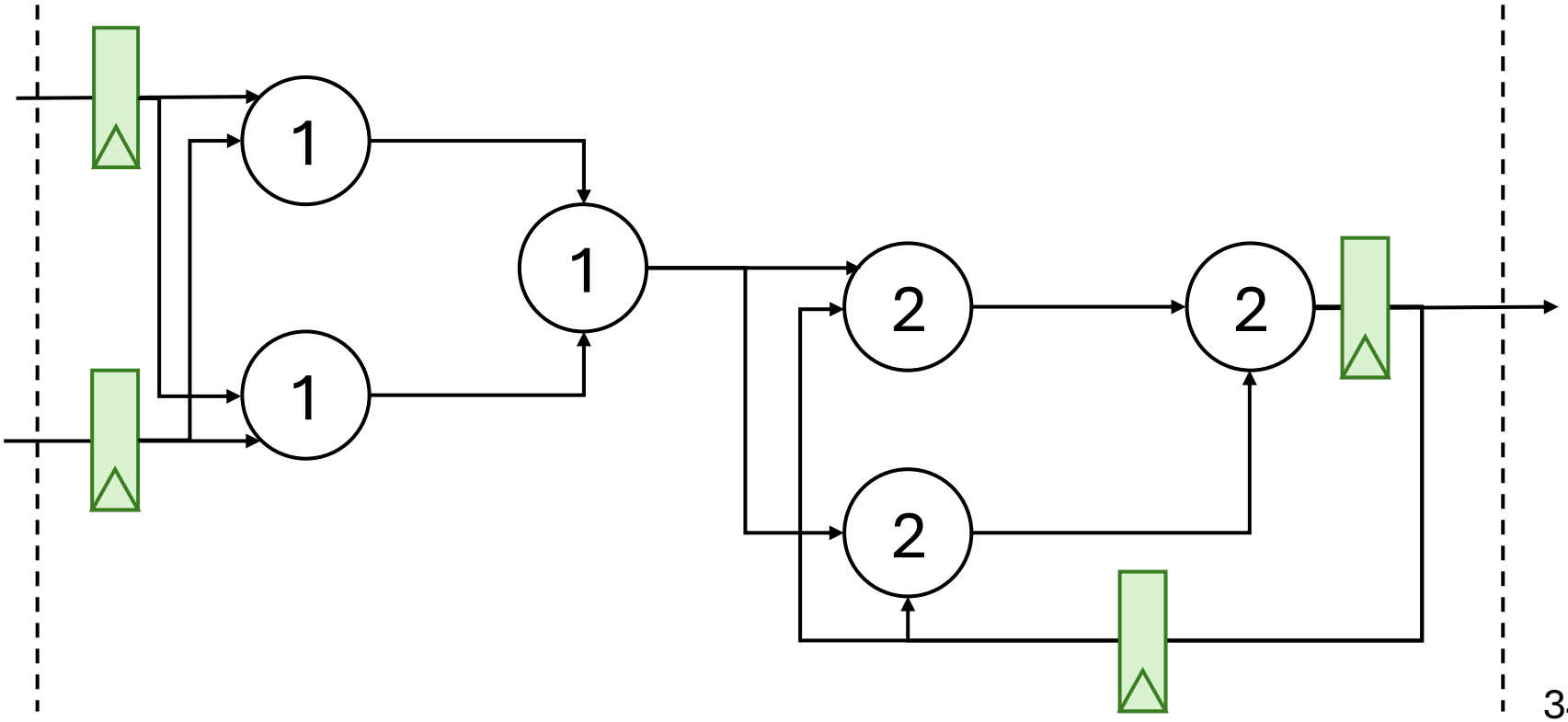
- Move **existing** registers around to optimize frequency of the circuit → does not add new registers



Practice Problem #4

Given the circuit diagram below, where each circle is an operator and the number in it is its delay in nanoseconds. Input/Output registers are represented as dashed lines and cannot be moved.

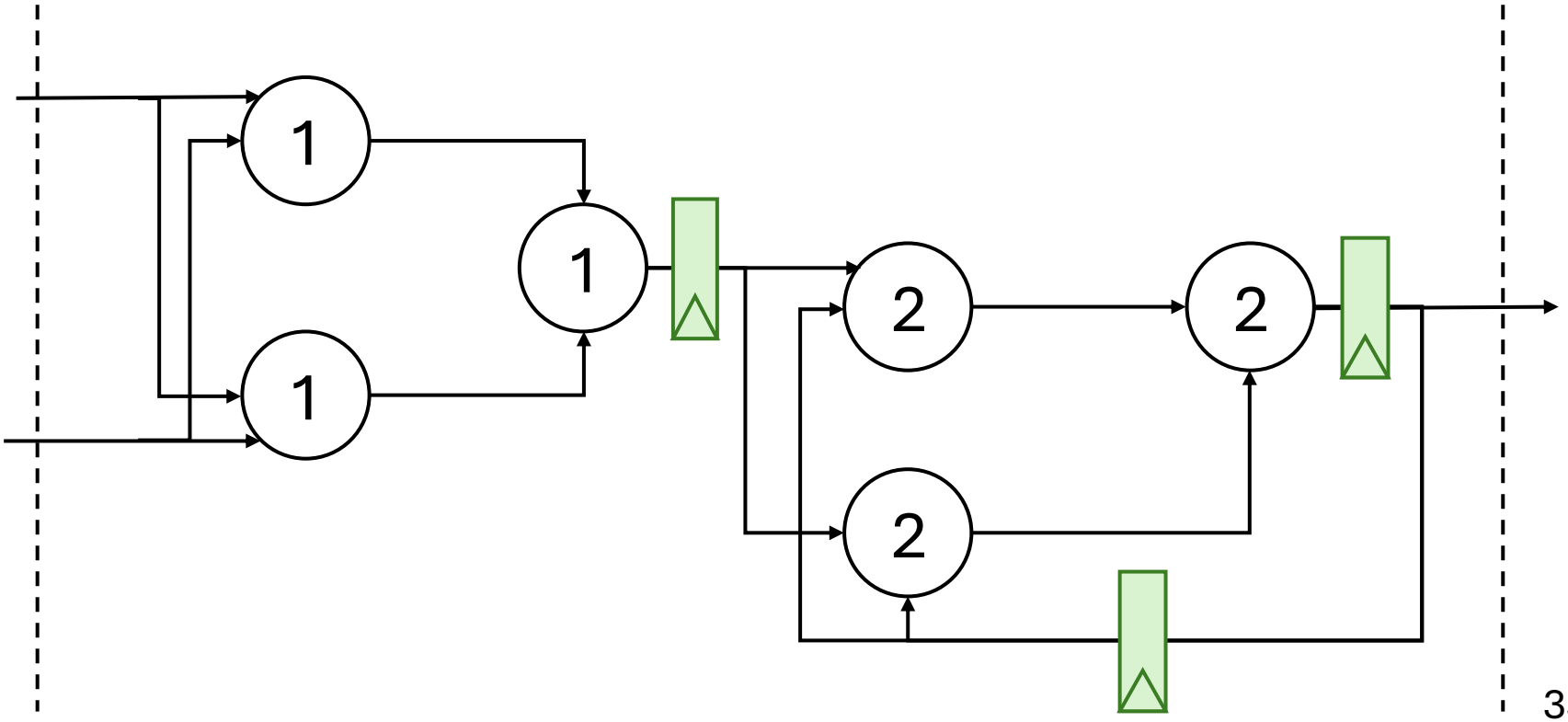
(a) Retime the circuit to reduce critical path delay given that you need to reduce the area of the circuit by one register. What is the critical path?



Practice Problem #4

Given the circuit diagram below, where each circle is an operator and the number in it is its delay in nanoseconds. Input/Output registers are represented as dashed lines and cannot be moved.

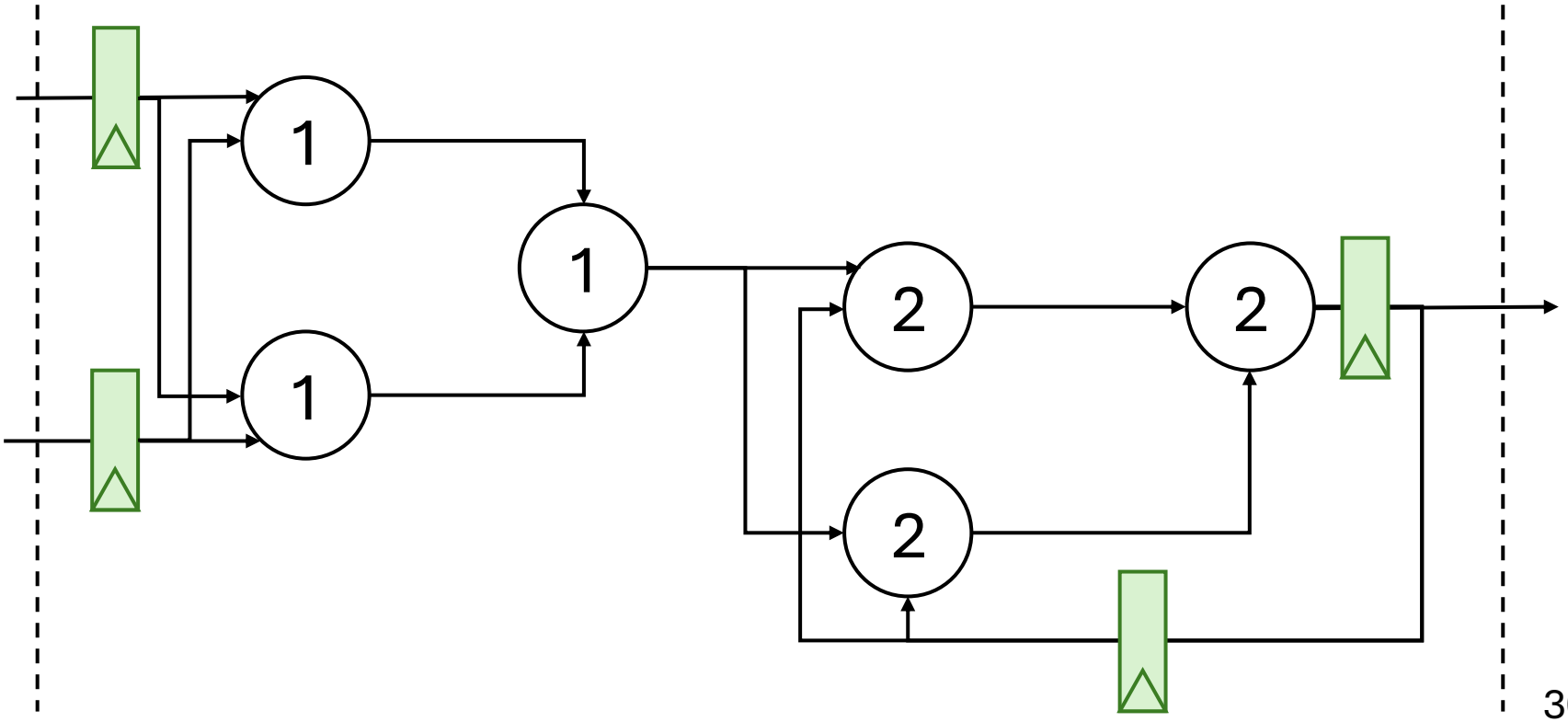
(a) Retime the circuit to reduce critical path delay given that you need to reduce the area of the circuit by one register. What is the critical path?



Practice Problem #4

Given the circuit diagram below, where each circle is an operator and the number in it is its delay in nanoseconds. Input/Output registers are represented as dashed lines and cannot be moved.

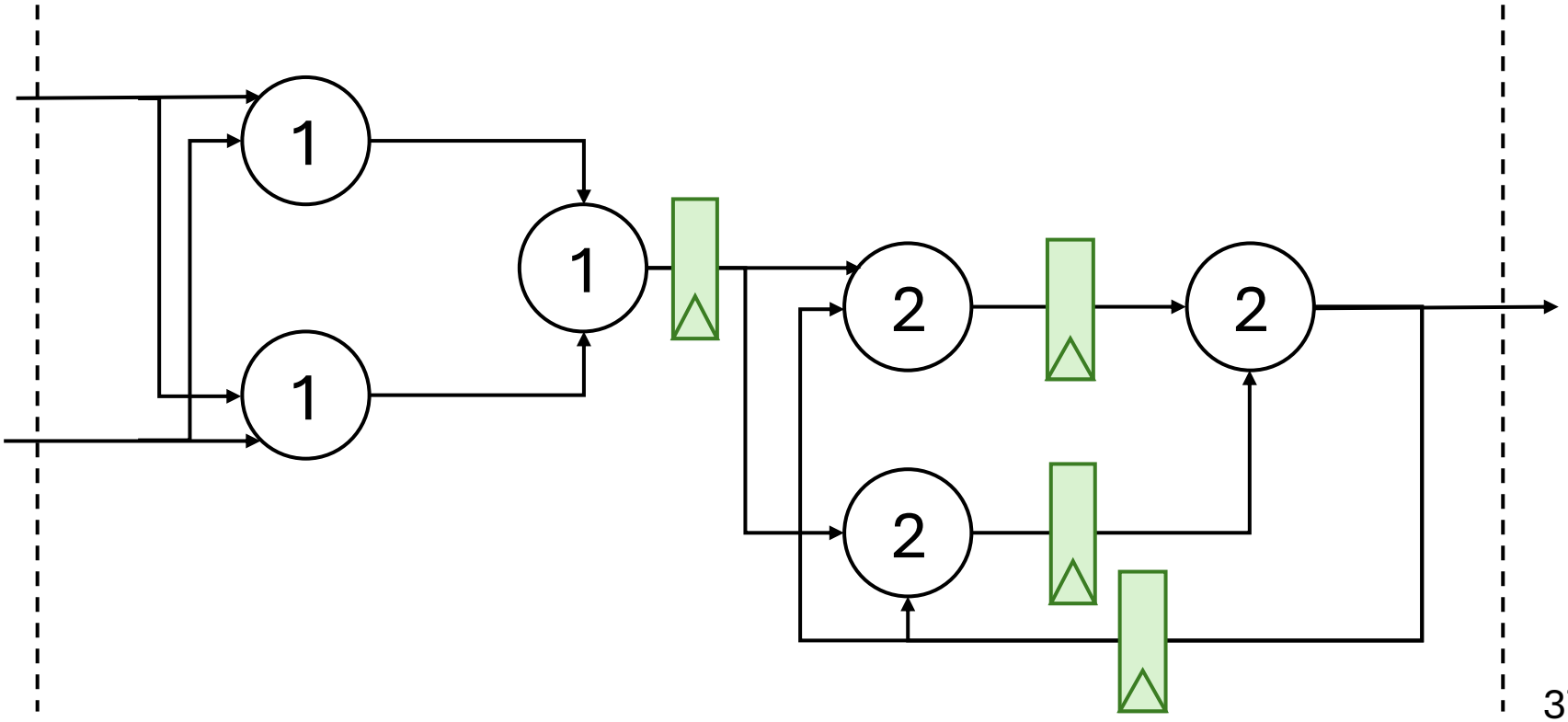
(b) Retime the circuit to minimize critical path delay if you don't need to reduce the area of the circuit. What is the critical path?



Practice Problem #4

Given the circuit diagram below, where each circle is an operator and the number in it is its delay in nanoseconds. Input/Output registers are represented as dashed lines and cannot be moved.

(b) Retime the circuit to minimize critical path delay if you don't need to reduce the area of the circuit. What is the critical path?



Latency-Insensitive Design

- Ready-valid handshakes between sender and receiver:
  - Transfer happens if ready && valid
  - Regardless of when that happens, functionality is not broken
- To stall a pipelined latency-insensitive channel:
  - Purely combinational ready signal to clock enable all registers
    - Very simple and works fine for localized connections
    - Critical path for global connections
  - Relay stations
    - Pipelines all signals in a latency-insensitive channel
    - Higher cost (2x the registers + control FSM)

No Lectures/Tutorials for the Next Two Weeks

Good luck with all the midterms!