

More Digital Circuits

Administrivia

- Lecture quizzes will now be due on Saturday each week
- No lab this week
- Project 2 Deadlines
 - Part A due this Thursday (2/25)
 - Part B due next Thursday (3/4)
- Check-Ins continue this week
- Midterm exam coming up on 3/17
 - We'll be releasing more information about this soon

Type of Circuits

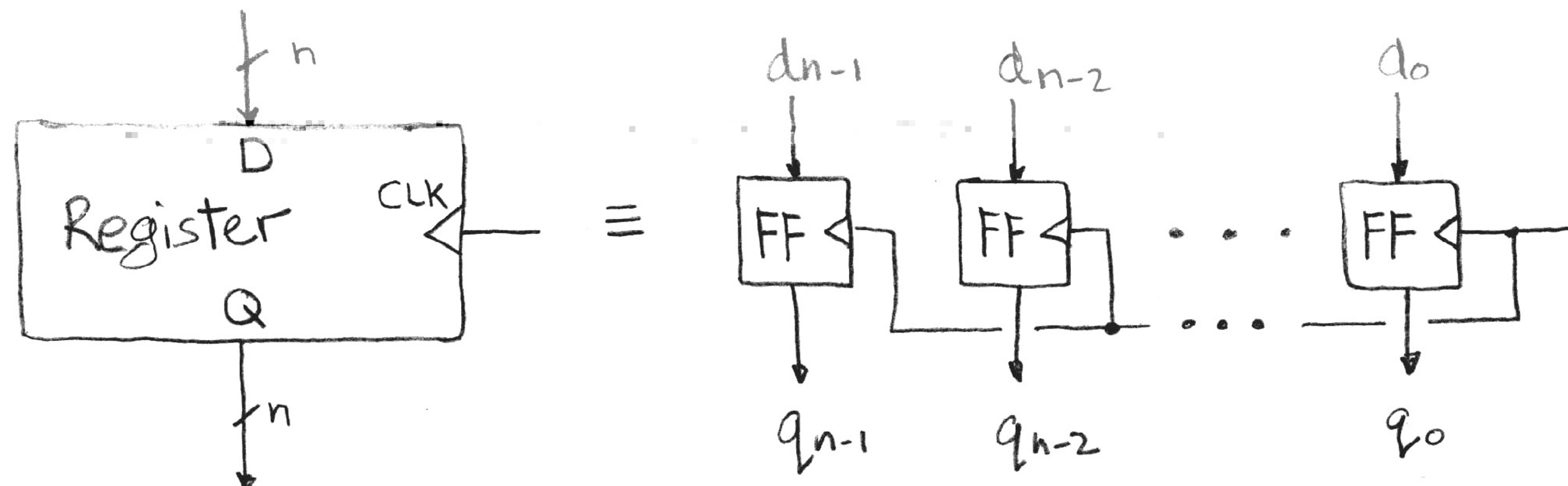
- *Synchronous Digital Systems* consist of two basic types of circuits:
 - Combinational Logic (CL) circuits
 - Output is a function of the inputs only, not the history of its execution
 - E.g., circuits to add A, B (ALUs)
 - Sequential Logic (SL)
 - Circuits that “remember” or store information
 - aka “State Elements”
 - E.g., memories and registers (Registers)

Uses for State Elements

- Place to store values for later re-use:
 - Register files (like x1-x31 in RISC-V)
 - Memory (caches and main memory)
- *Help control flow of information between combinational logic blocks*
 - State elements hold up the movement of information at input to combinational logic blocks to allow for orderly passage

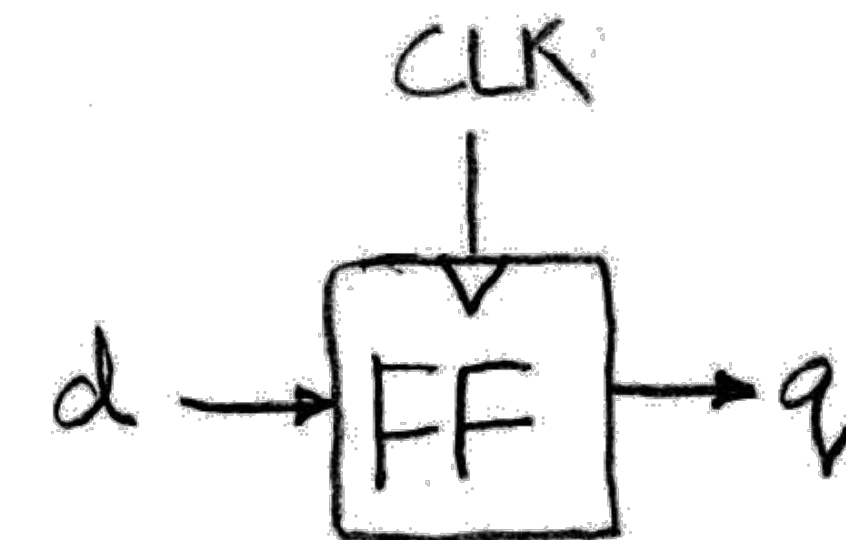
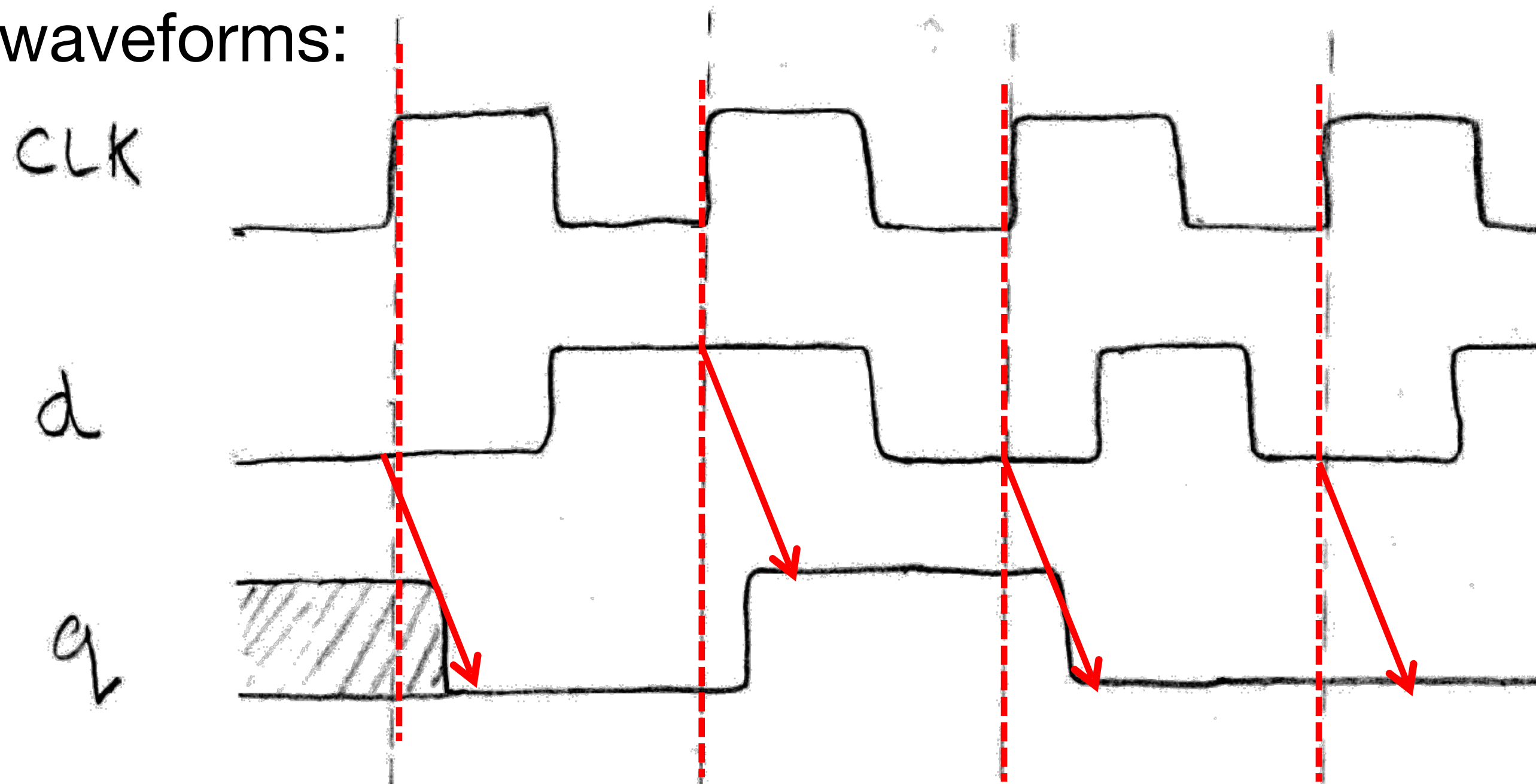
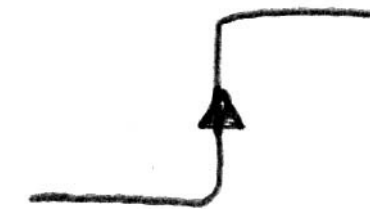
Register Internals

- n instances of a “Flip-Flop”
- Flip-flop name because the output flips and flops between 0 and 1
- D is “data input”, Q is “data output”
- Also called “D-type Flip-Flop”

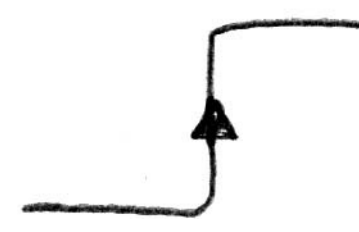


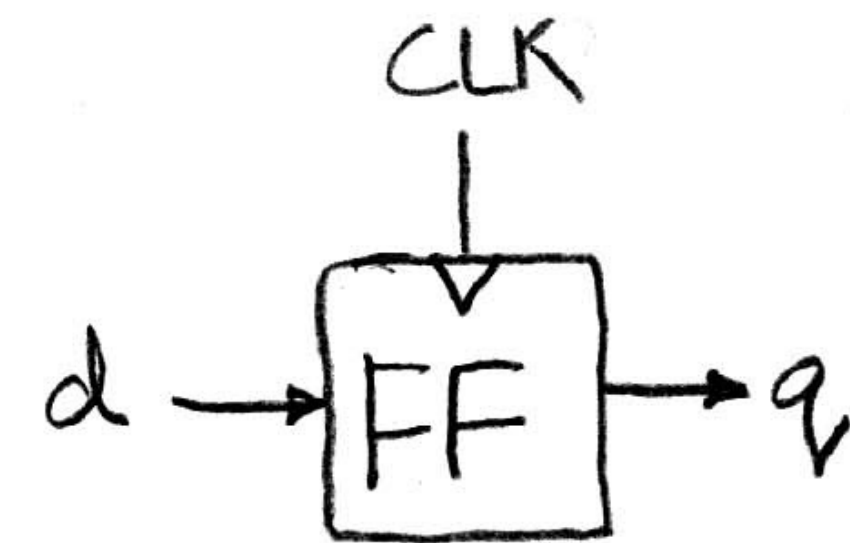
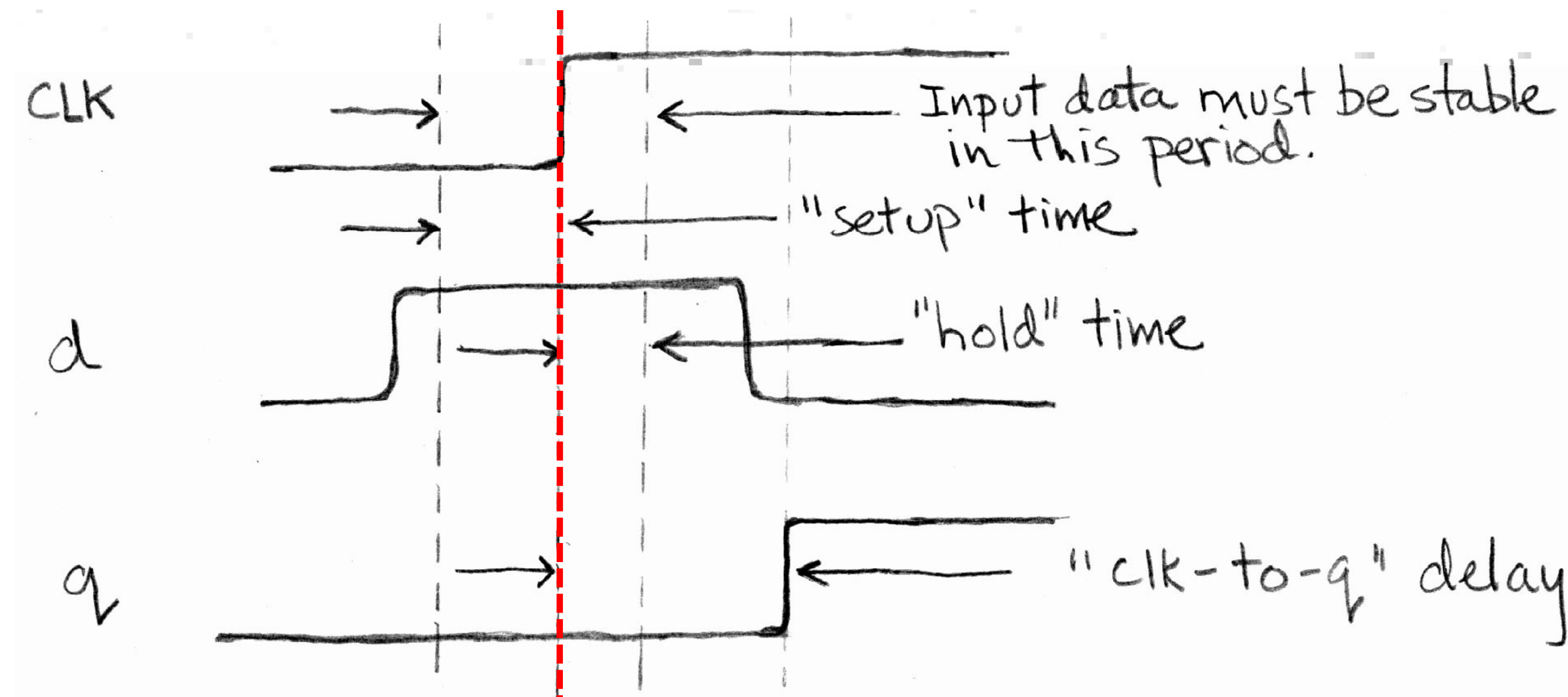
Flip-Flop Operation

- Edge-triggered d-type flip-flop
 - This one is “positive edge-triggered”
- “On the rising edge of the clock, the input d is sampled and transferred to the output. At all other times, the input d is ignored.”
- Example waveforms:



Flip-Flop Timing

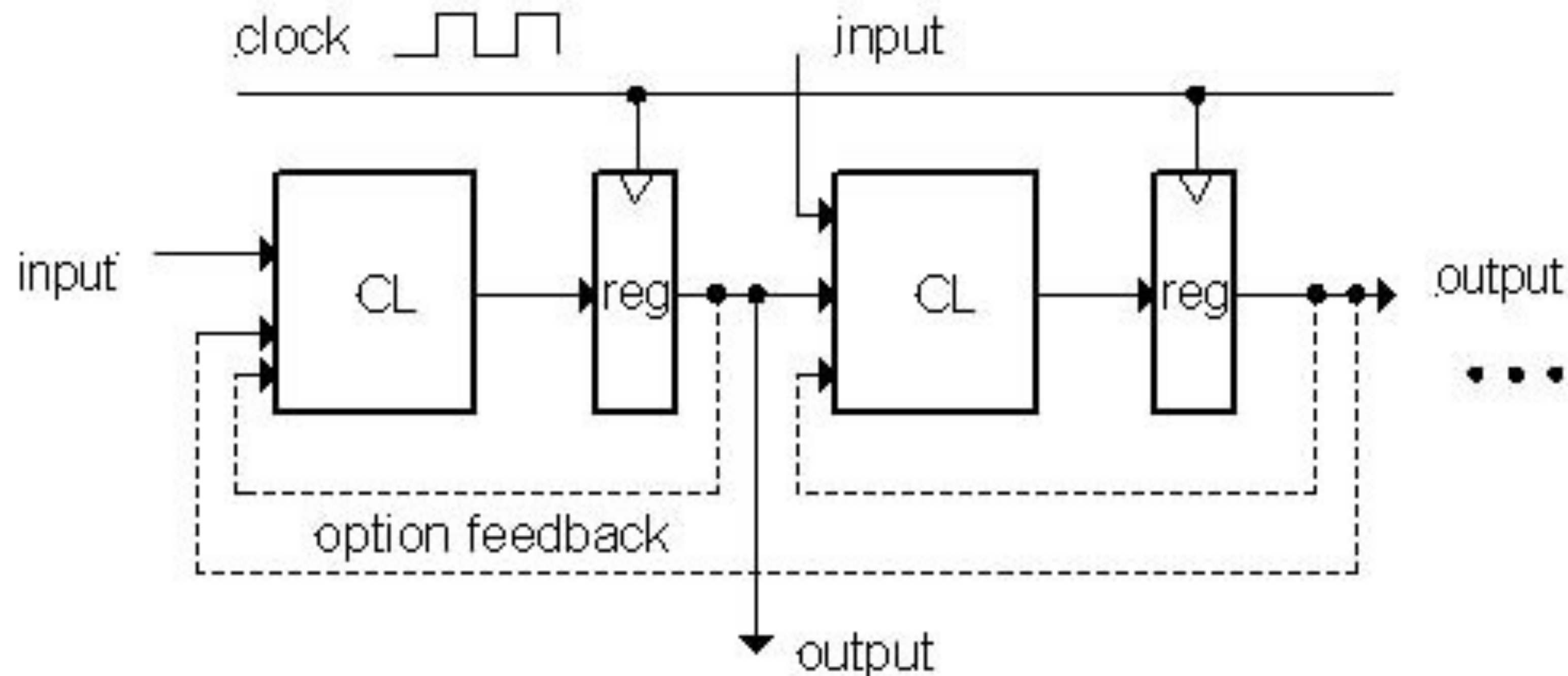
- Edge-triggered d-type flip-flop
 - This one is “positive edge-triggered” 
- “On the rising edge of the clock, the input d is sampled and transferred to the output. At all other times, the input d is ignored.”
- Example waveforms (more detail):



Hardware Timing Terms

- **Setup Time:** when the input must be stable *before* the edge of the CLK
- **Hold Time:** when the input must be stable *after* the edge of the CLK
- **“CLK-to-Q” Delay:** how long it takes the output to change, measured from the edge of the CLK

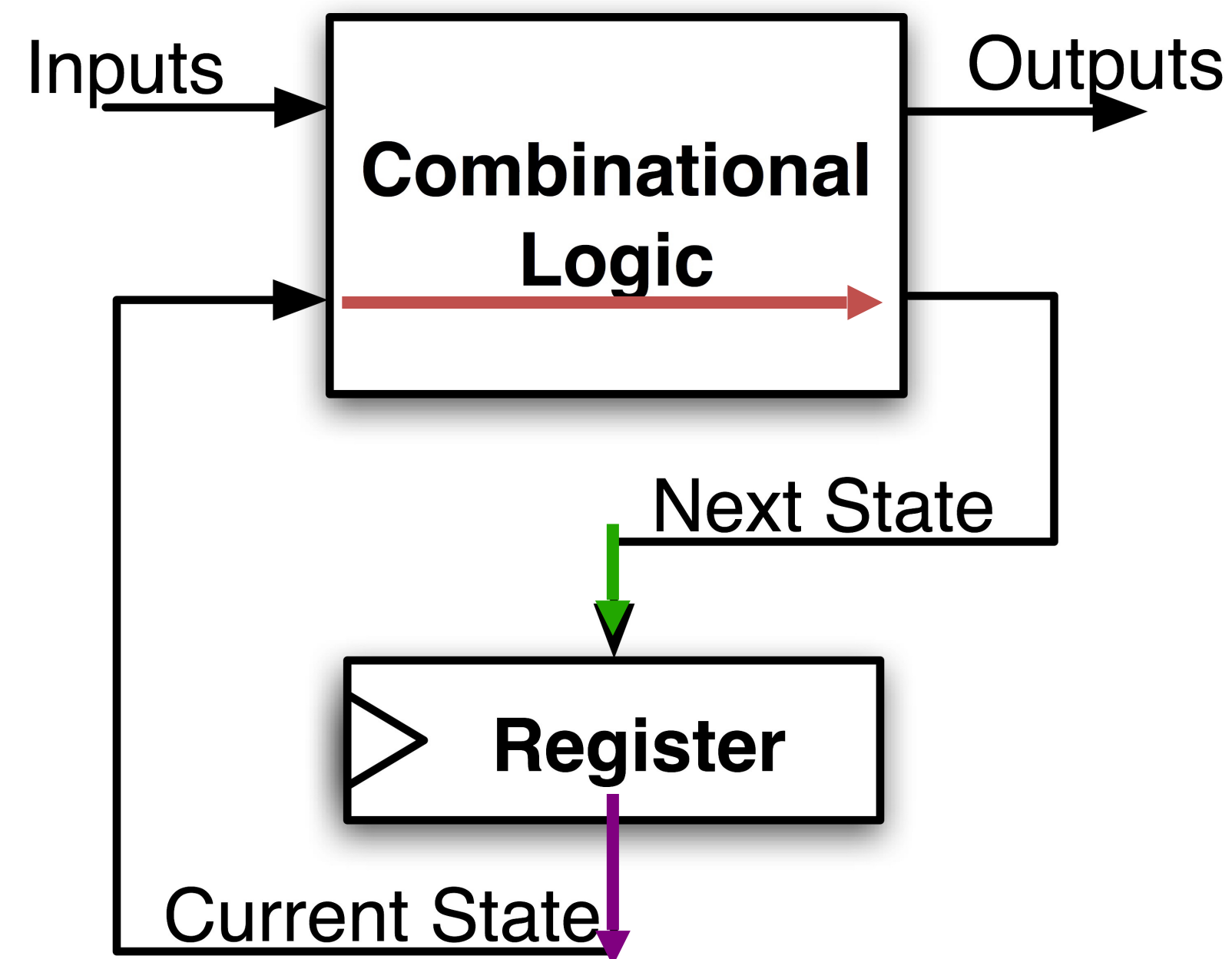
Model for Synchronous Systems



- Collection of Combinational Logic blocks separated by registers
- Feedback is optional
- Clock signal(s) connects only to clock input of registers
- Clock (CLK): steady square wave that synchronizes the system
- Register: several bits of state that samples on rising edge of CLK (positive edge-triggered) or falling edge (negative edge-triggered)

Maximum Clock Frequency

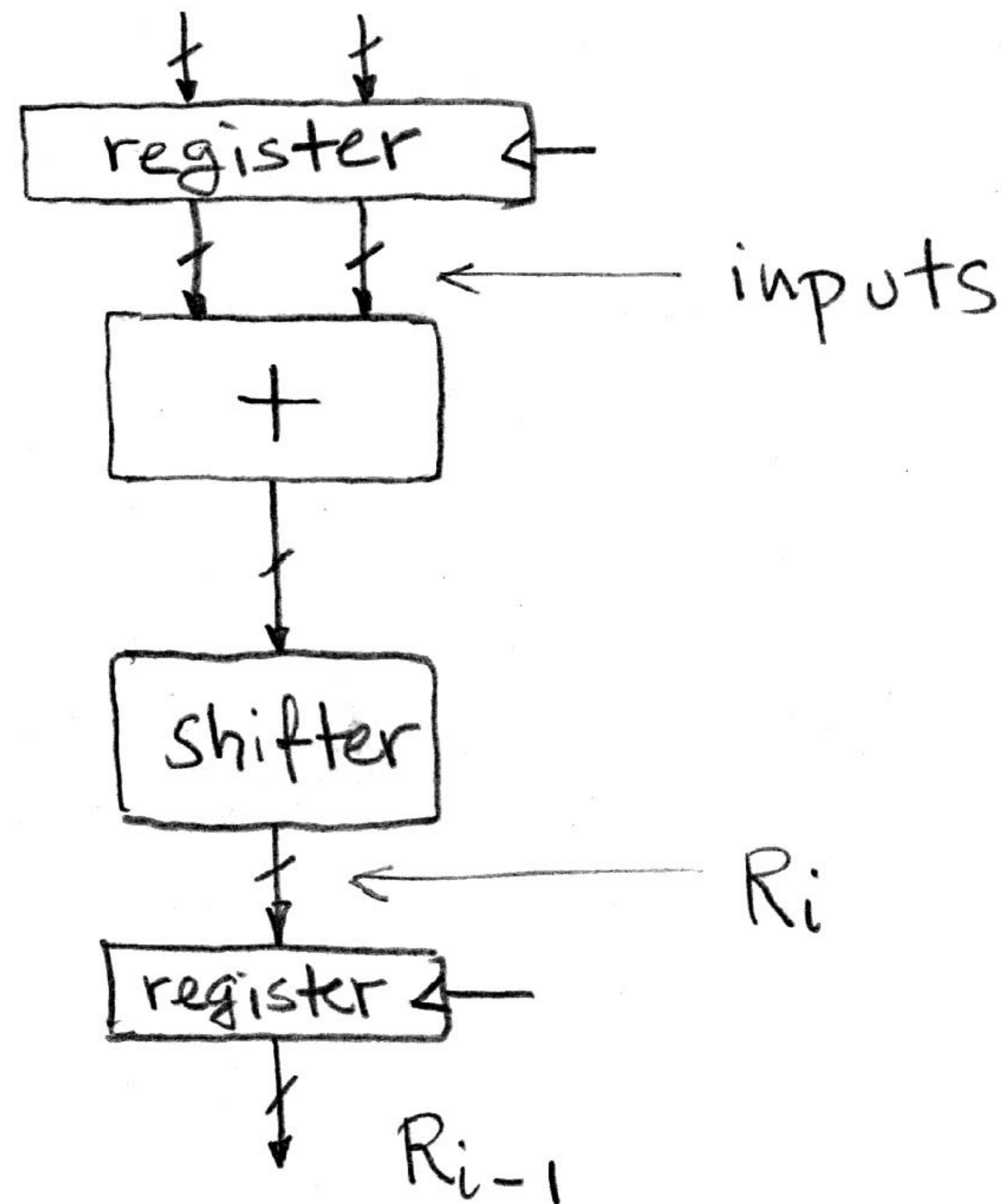
- What is the maximum frequency of this circuit?



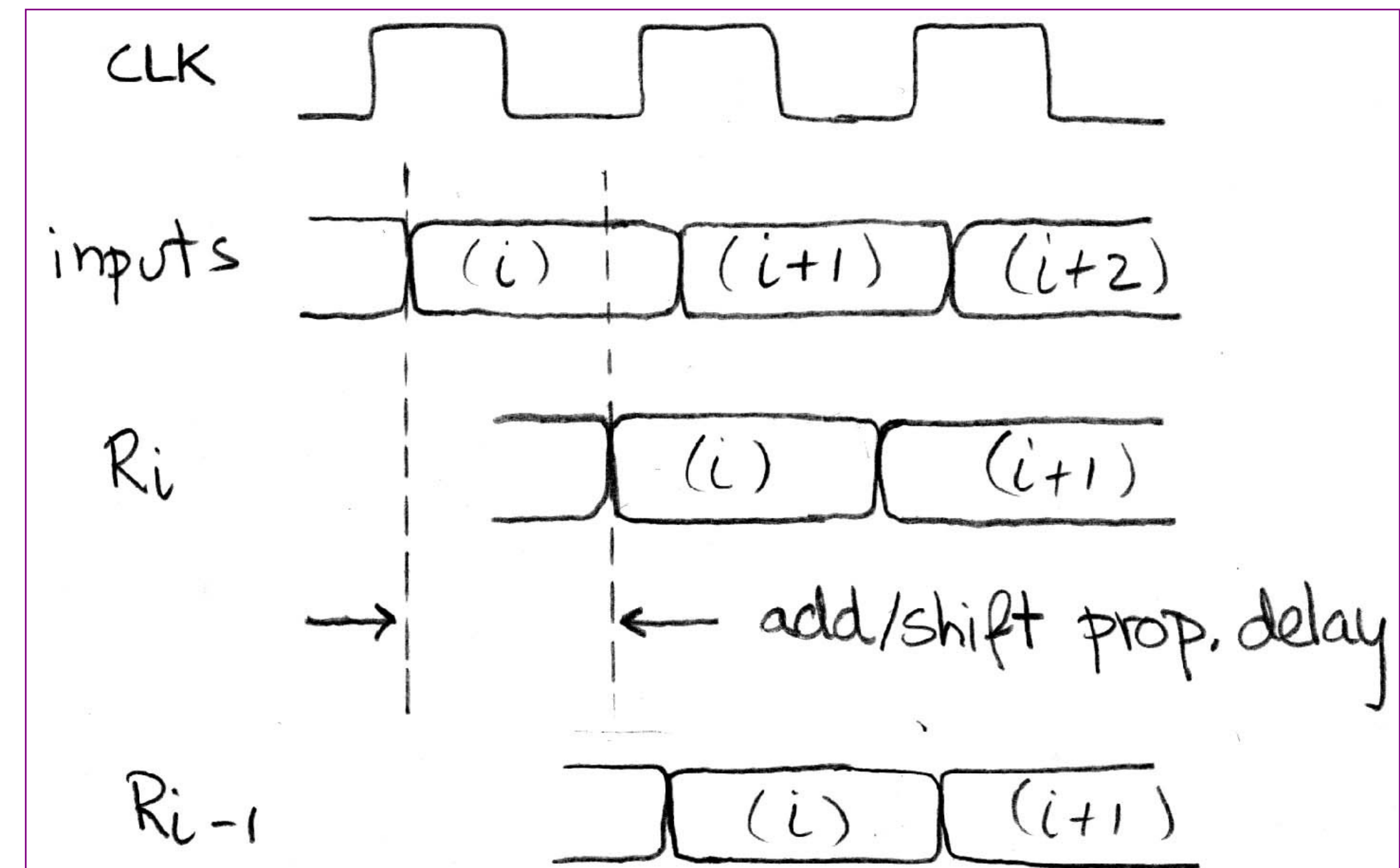
Hint:
Frequency = $1/\text{Period}$

$$\text{Period} = \text{Max Delay} = \text{CLK-to-Q Delay} + \text{CL Delay} + \text{Setup Time}$$

Critical Paths



Timing...



Note: delay of 1 clock cycle from input to output.

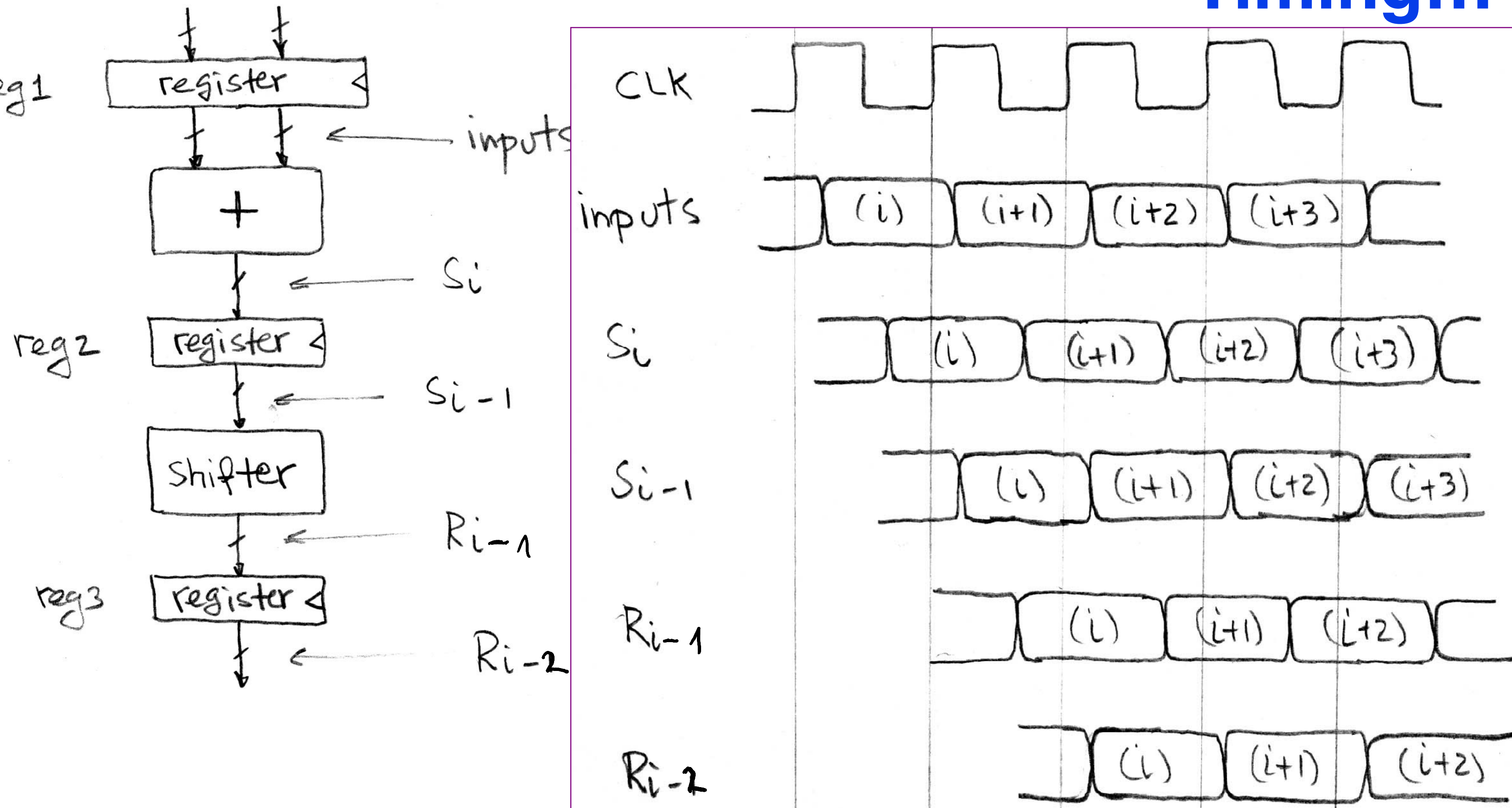
Clock period limited by propagation delay of adder/shifter.

Pipelining to improve performance

Timing...

Computer Science 61C Spring

Kolb and Weaver



- Insertion of register allows higher clock frequency
- More outputs per second (higher bandwidth)
- But each individual result takes longer (greater latency)

Recap of Timing Terms

- **Clock (CLK)** - steady square wave that synchronizes system
- **Setup Time** - when the input must be stable before the rising edge of the CLK
- **Hold Time** - when the input must be stable after the rising edge of the CLK
- **“CLK-to-Q” Delay** - how long it takes the output to change, measured from the rising edge of the CLK

- **Flip-flop** - one bit of state that samples every rising edge of the CLK (positive edge-triggered)
- **Register** - several bits of state that samples on rising edge of CLK or on LOAD (positive edge-triggered)

Problems With Clocking...

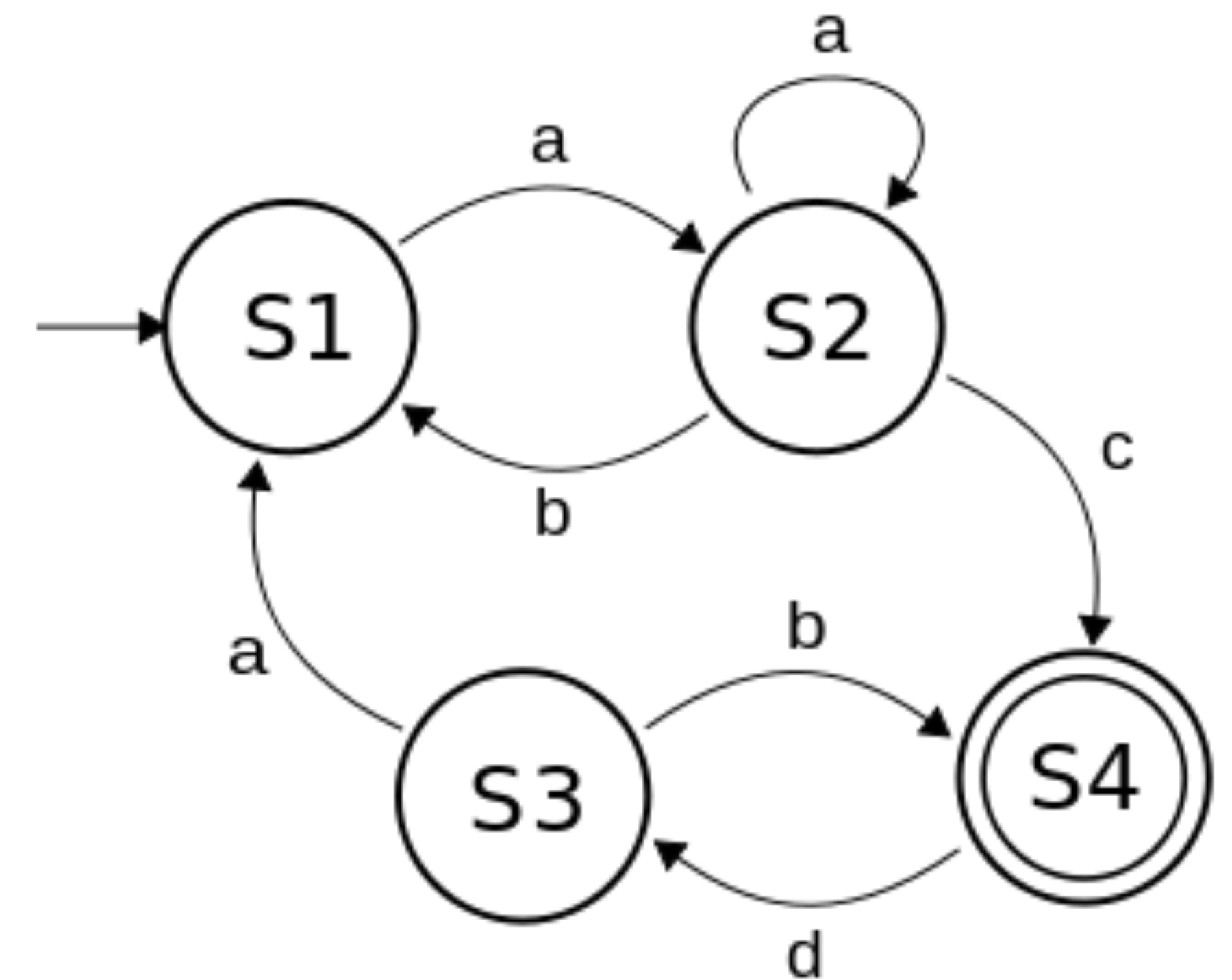
- The clock period ***must be*** longer than the critical path
 - Otherwise, you will get the wrong answers
- Critical path:
 - clk->q time
 - Necessary to get the output of the registers
 - ***worst case*** combinational logic delay
 - ***Setup time*** for the next register
- Must meet all of these to be correct

Hold-Time Violations...

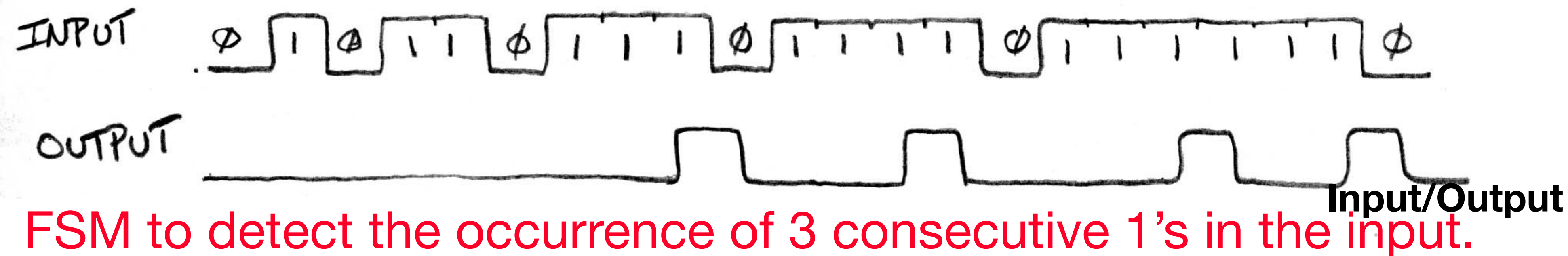
- An alternate problem can occur...
 - $\text{Clk} \rightarrow \text{Q} + \text{best case combinational delay} < \text{Hold time}$...
- What happens?
 - $\text{Clk} \rightarrow \text{Q} + \text{data propagates}$...
 - And now you don't hold the input to the flip flop long enough
- Solution:
 - **Add** delay on the best-case path (e.g. two inverters)

Finite State Machines (FSM) Intro

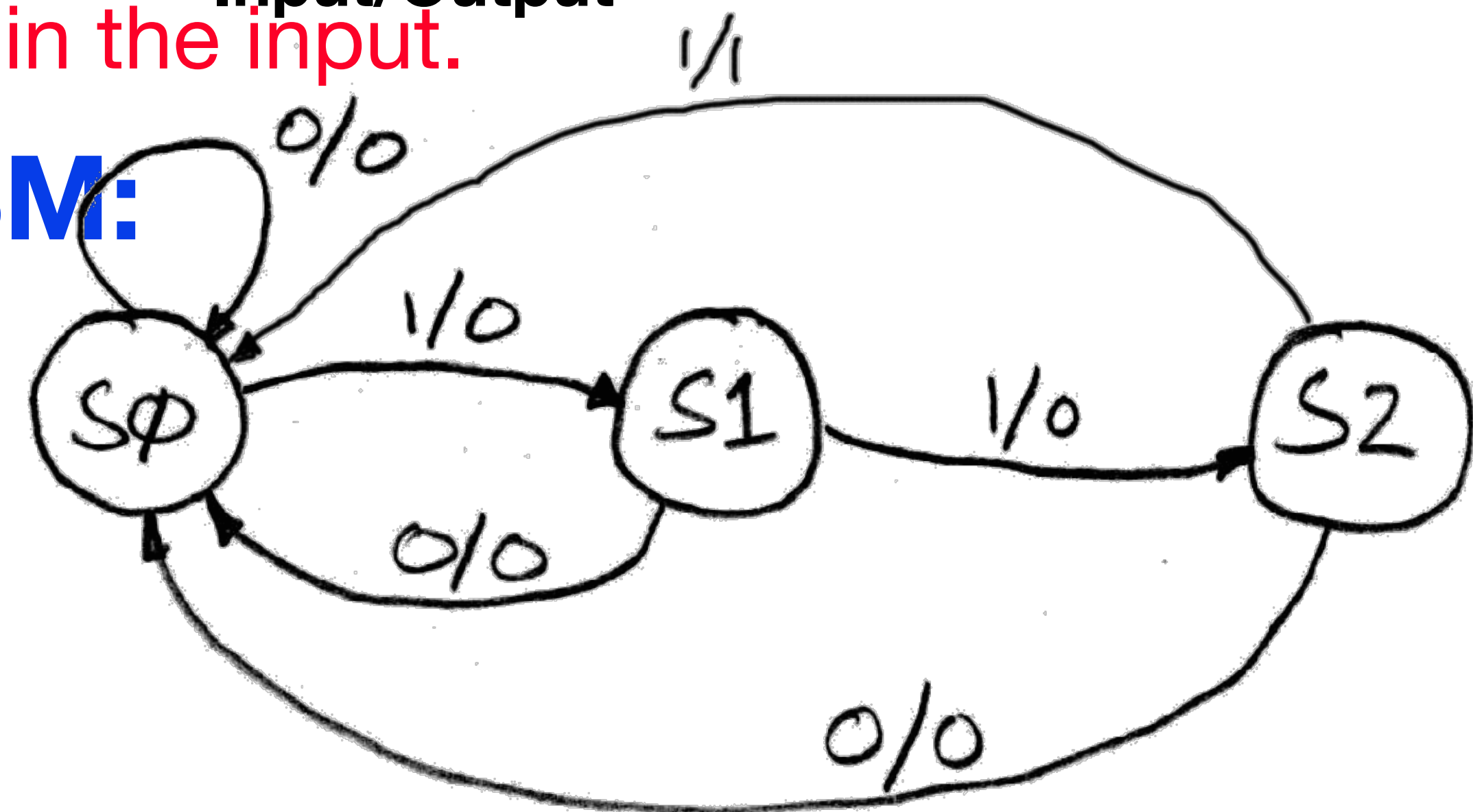
- A convenient way to conceptualize computation over time
- We start at a state and given an input, we follow some edge to another (or the same) state
- The function can be represented with a “state transition diagram”.
- With combinational logic and registers, any FSM can be implemented in hardware.



FSM Example: 3 ones...



Draw the FSM:



Assume state transitions are controlled by the clock:
On each clock cycle the machine checks the inputs and moves to a new state and produces a new output...

State Machine: Mealy v Moore Machines

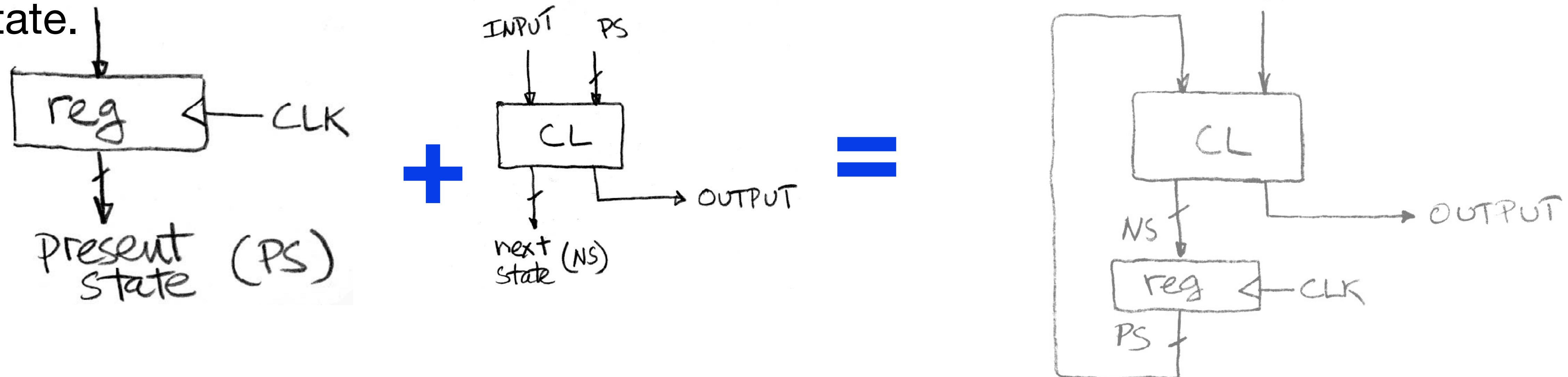
- Mealy machine:
 - Output is a function of both input and current state
 - Draw the outputs on the transitions
- Moore machine:
 - Output is ***only*** a function of the current state
 - Draw the outputs with the state bubbles

Advantages and Disadvantages

- Mealy Machine
 - Smaller/fewer states
 - Potentially slower in terms of clock rate:
 - Critical path of the **system** has to include both the input and output logic and whoever is using it
- Moore Machine
 - Bigger/more states
 - Slower in terms of clock cycles to respond:
 - Going to take an extra cycle to give the output from the input
 - Potentially higher clock rate:
 - Critical path of the **system**

Hardware Implementation of FSM

...therefore a register is needed to hold the a representation of which state the machine is in. Use a unique bit pattern for each state.



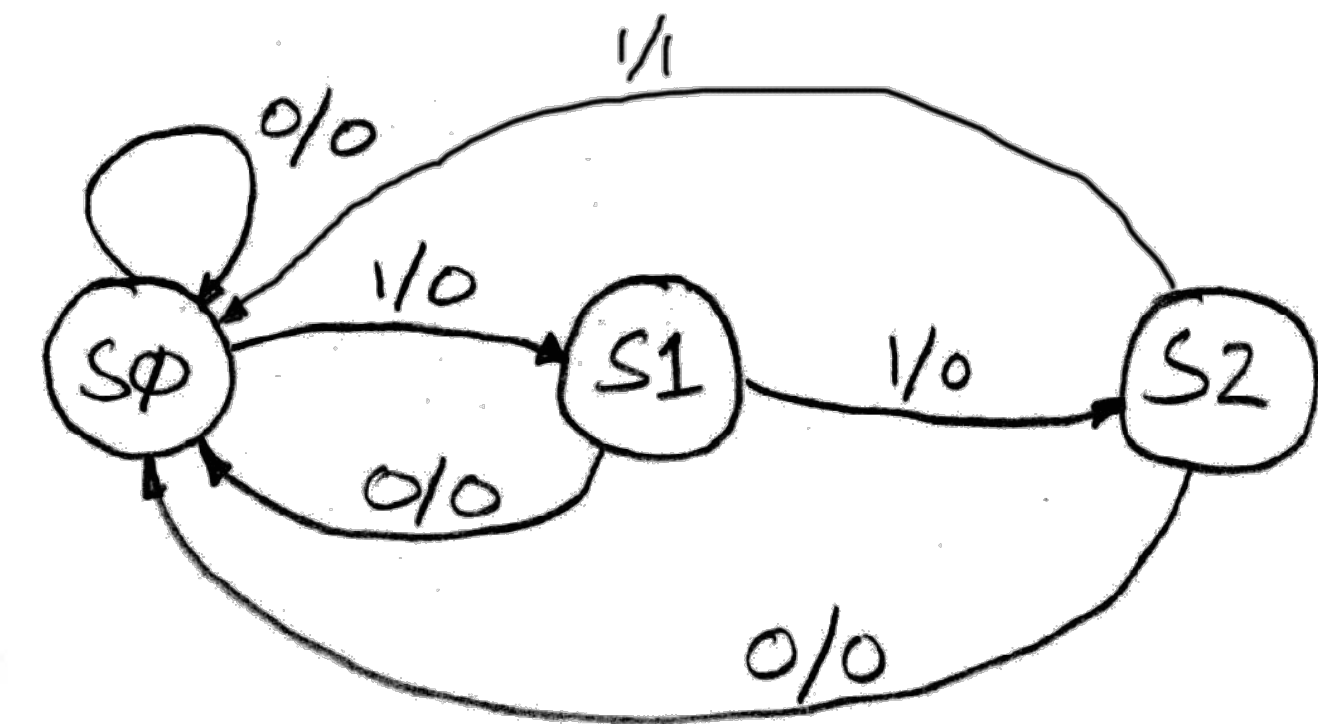
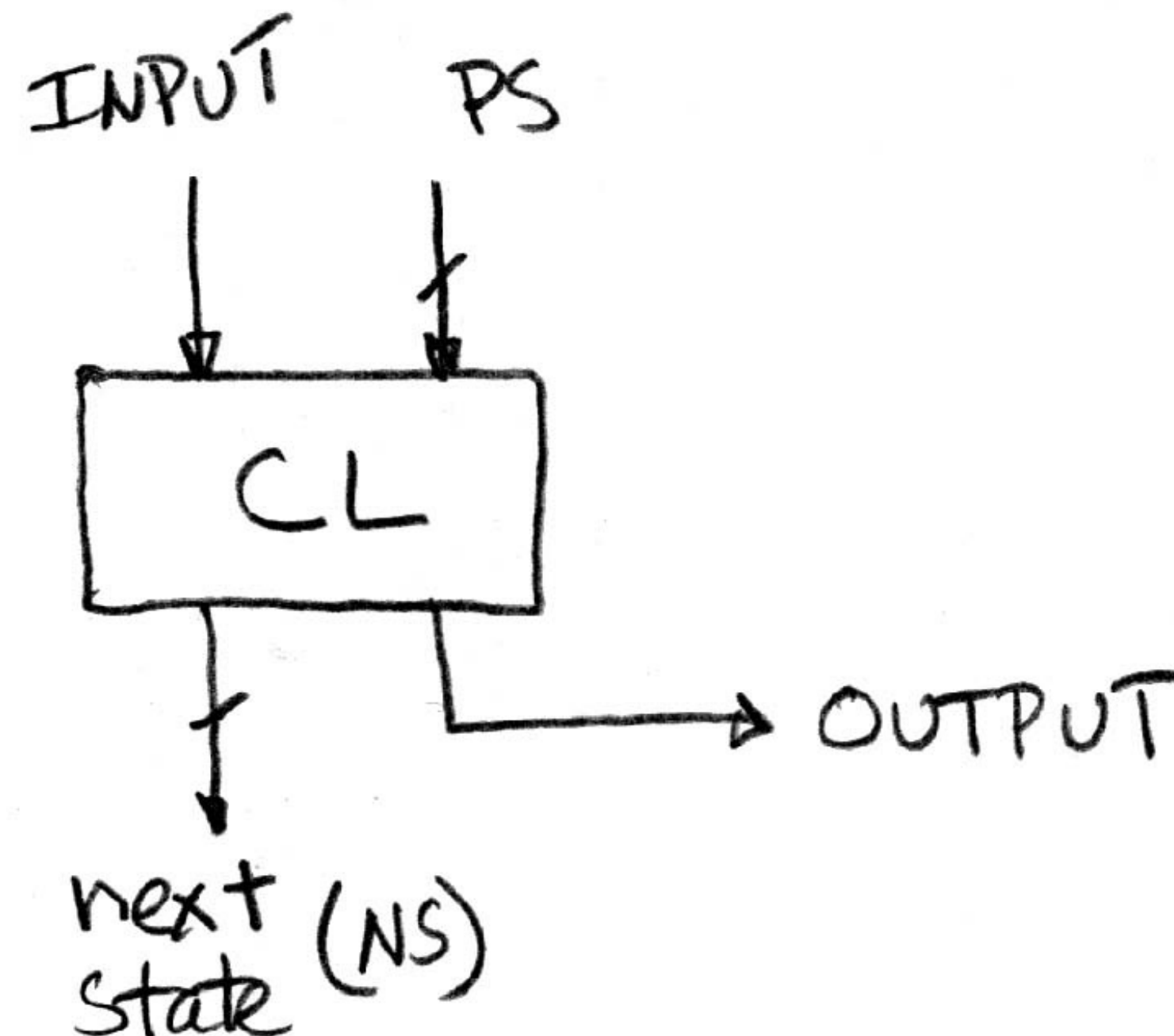
Combinational logic circuit is used to implement a function that maps from *present state and input* to *next state and output*.

FSM Combinational Logic

Specify CL using a truth table

Truth table...

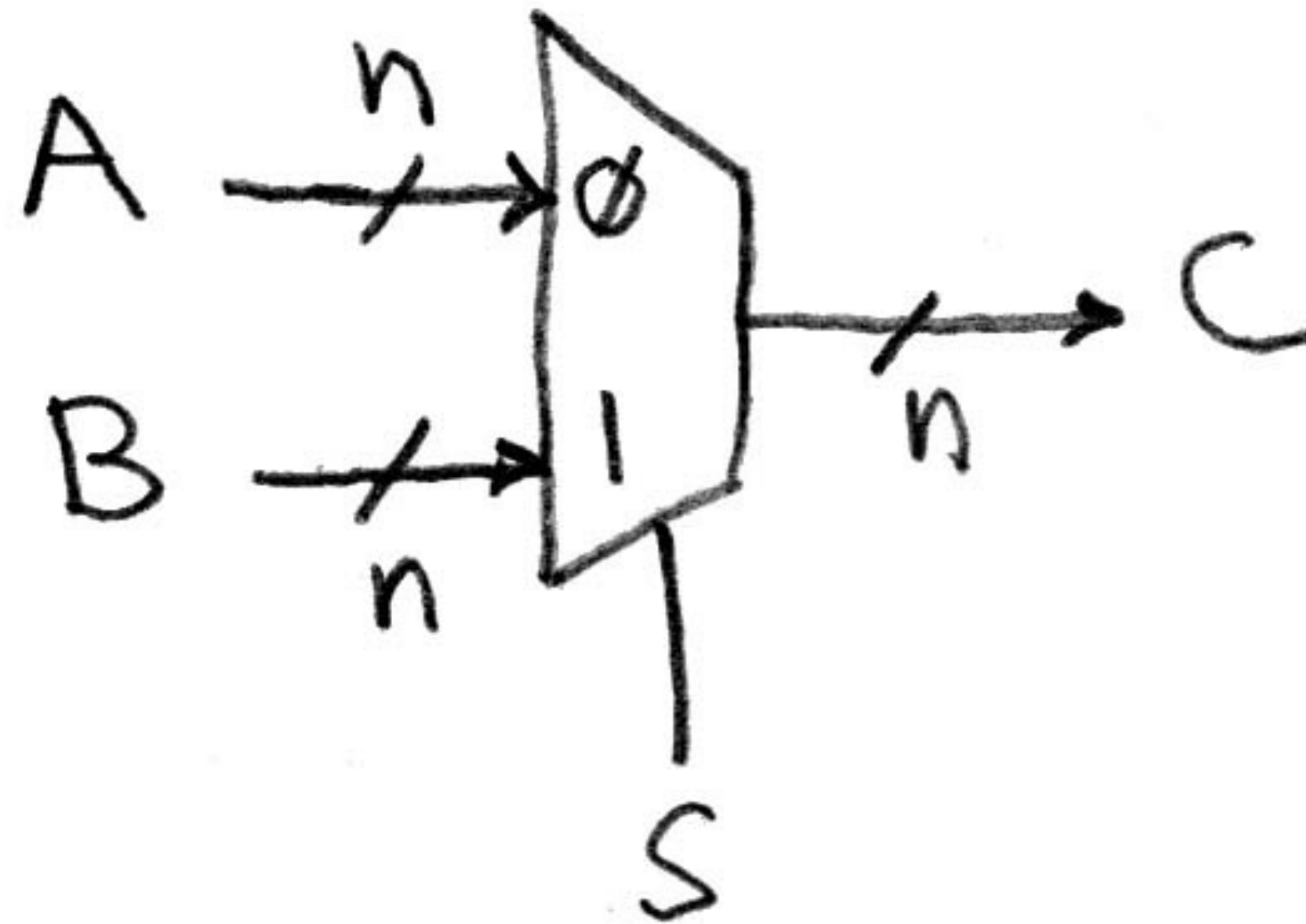
PS	Input	NS	Output
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	00	1



Building Standard Functional Units

- Data multiplexers
- Arithmetic and Logic Unit
- Adder/Subtractor

Data Multiplexer (“Mux”) (here 2-to-1, n-bit-wide)

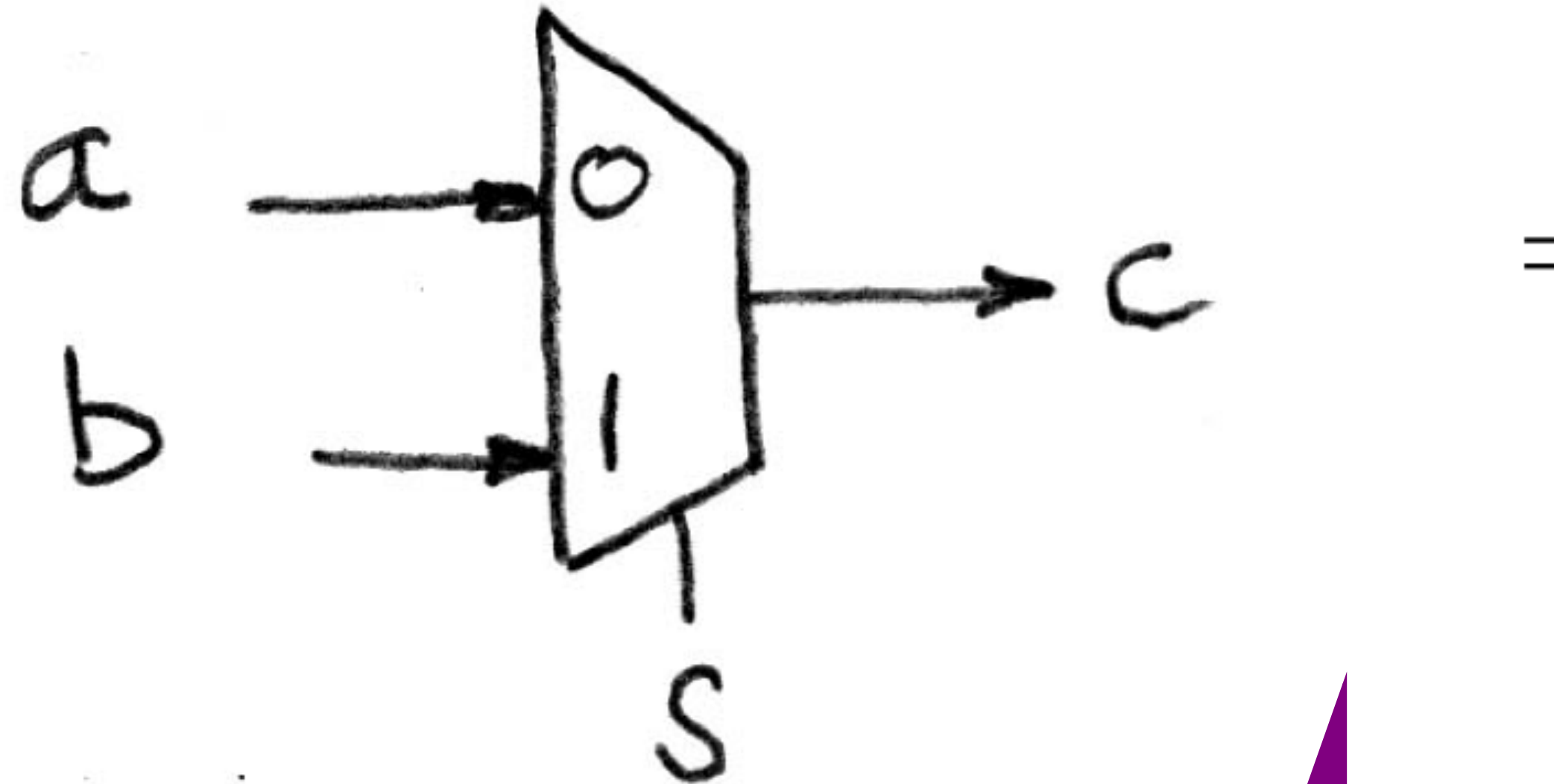


N instances of 1-bit-wide mux

How many rows in TT?

Computer Science 61C Spring 2021

Kolb and Weaver

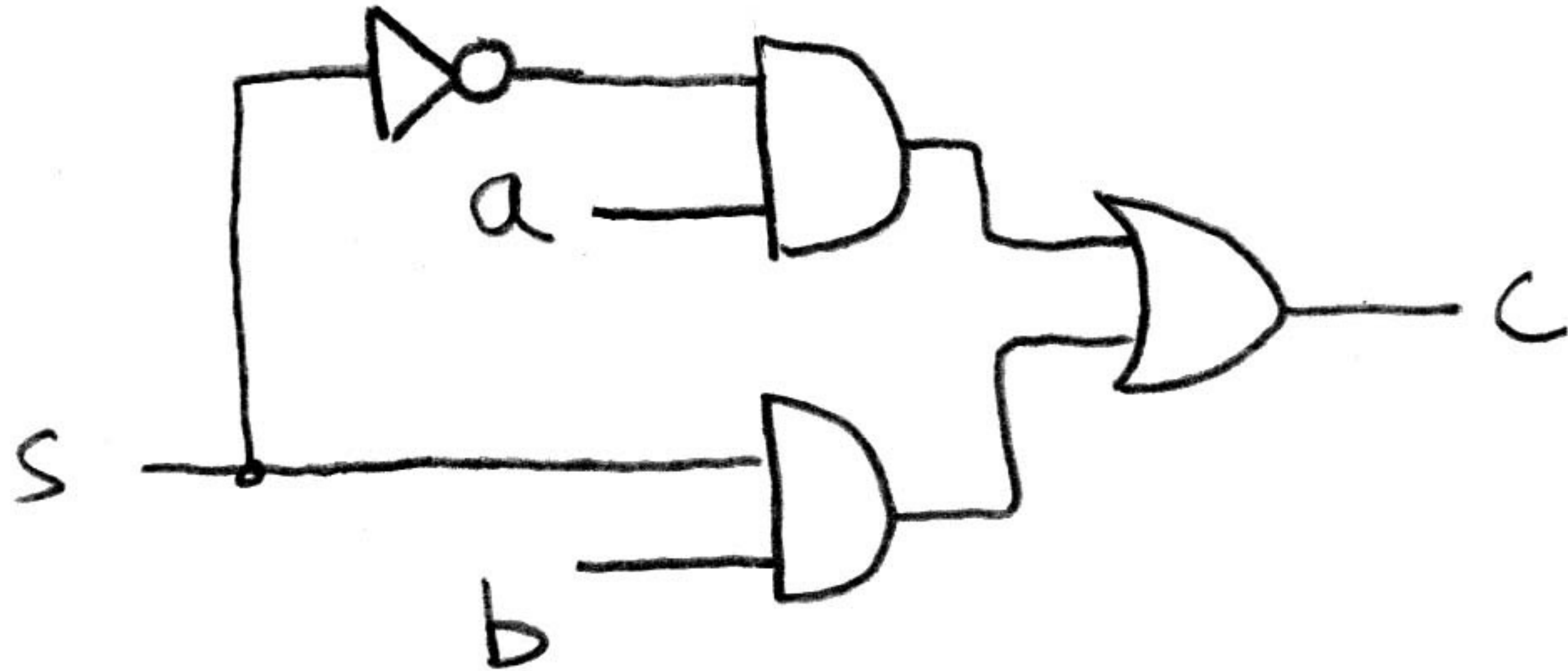


$$\begin{aligned}c &= \bar{s}a\bar{b} + \bar{s}ab + s\bar{a}b + sab \\&= \bar{s}(a\bar{b} + ab) + s(\bar{a}b + ab) \\&= \bar{s}(a(\bar{b} + b)) + s((\bar{a} + a)b) \\&= \bar{s}(a(1) + s((1)b) \\&= \bar{s}a + sb\end{aligned}$$

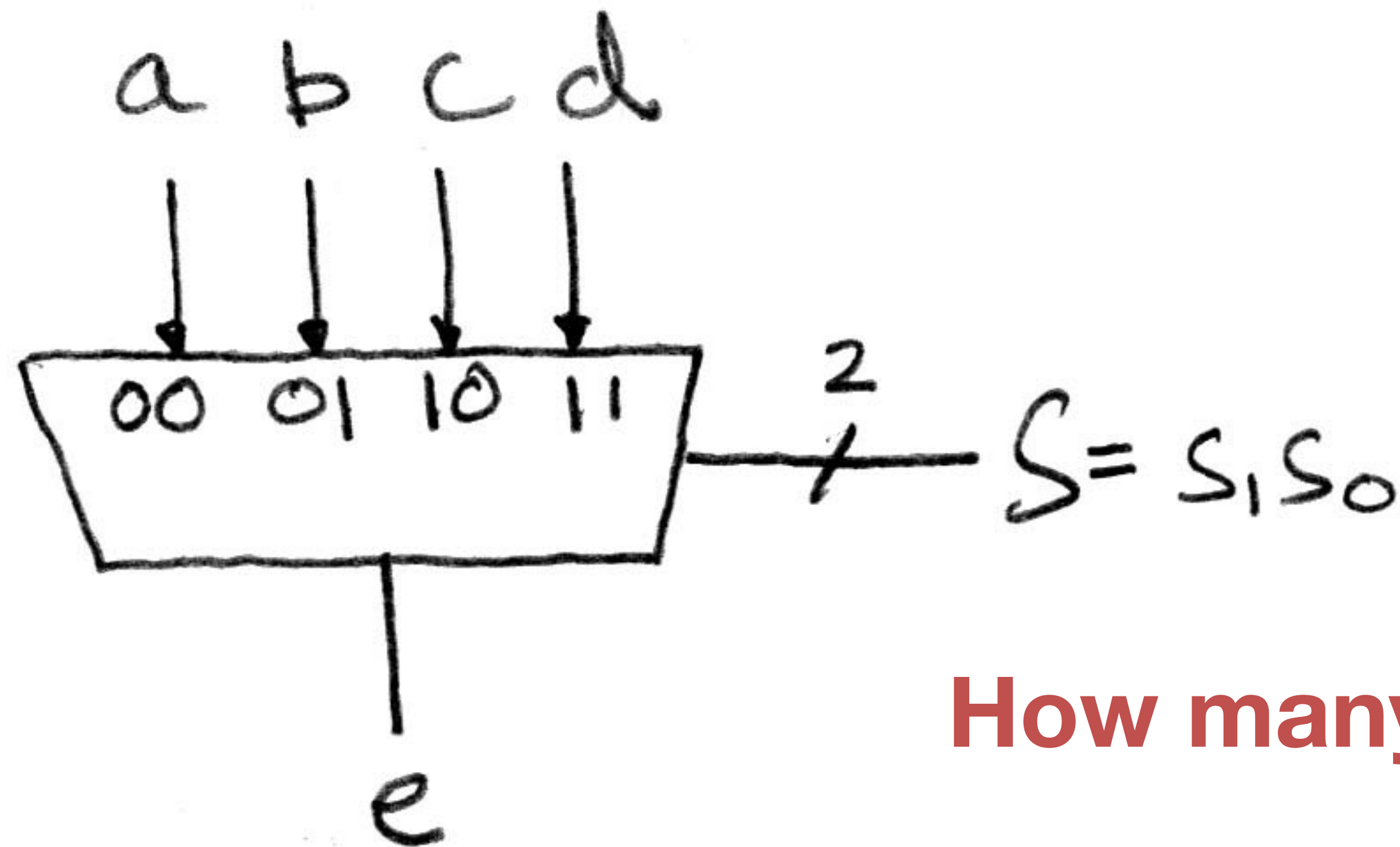


How do we build a 1-bit-wide mux?

$$\bar{s}a + sb$$



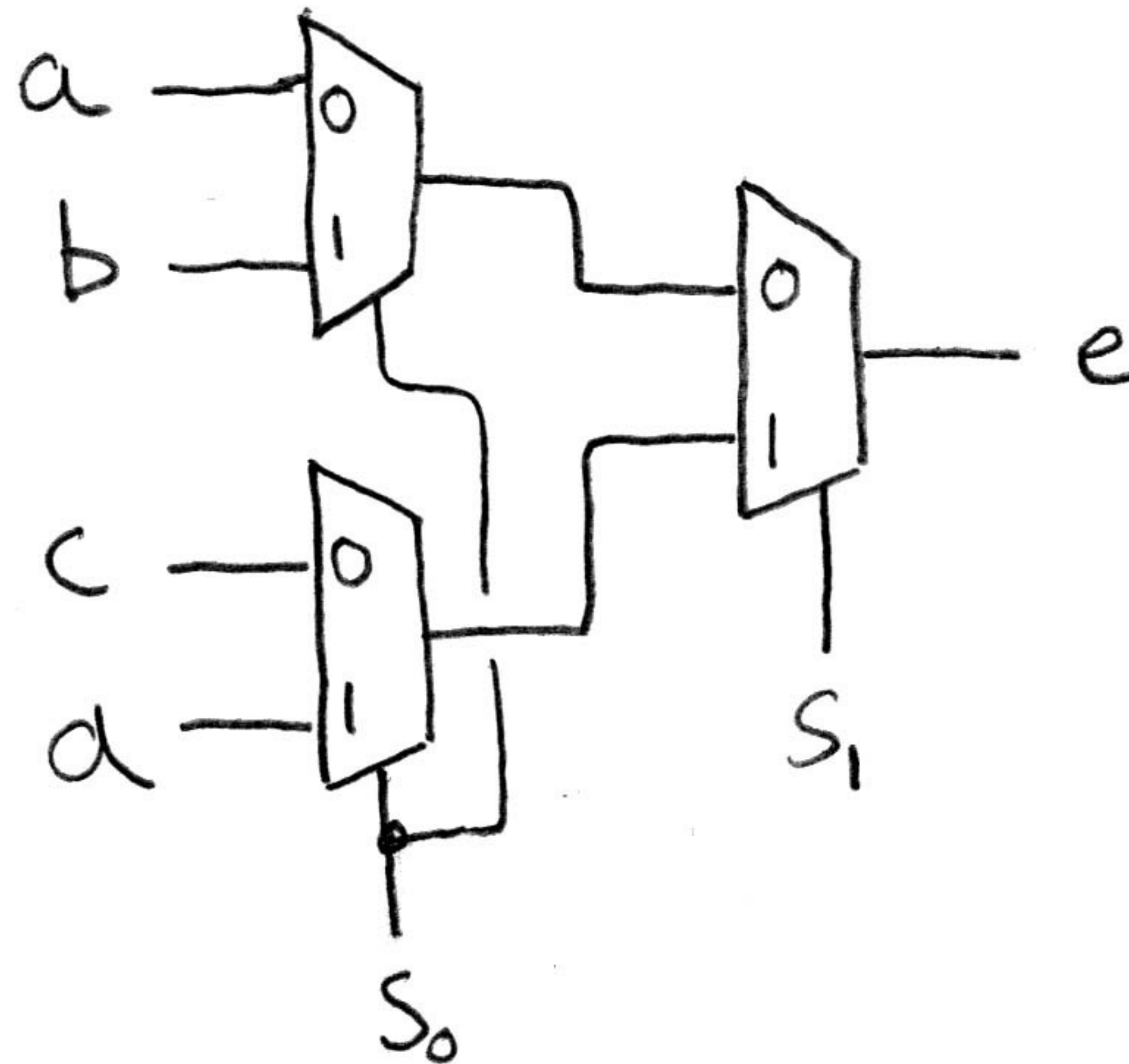
4-to-1 multiplexer?



How many rows in TT?

$$e = \bar{s}_1 \bar{s}_0 a + \bar{s}_1 s_0 b + s_1 \bar{s}_0 c + s_1 s_0 d$$

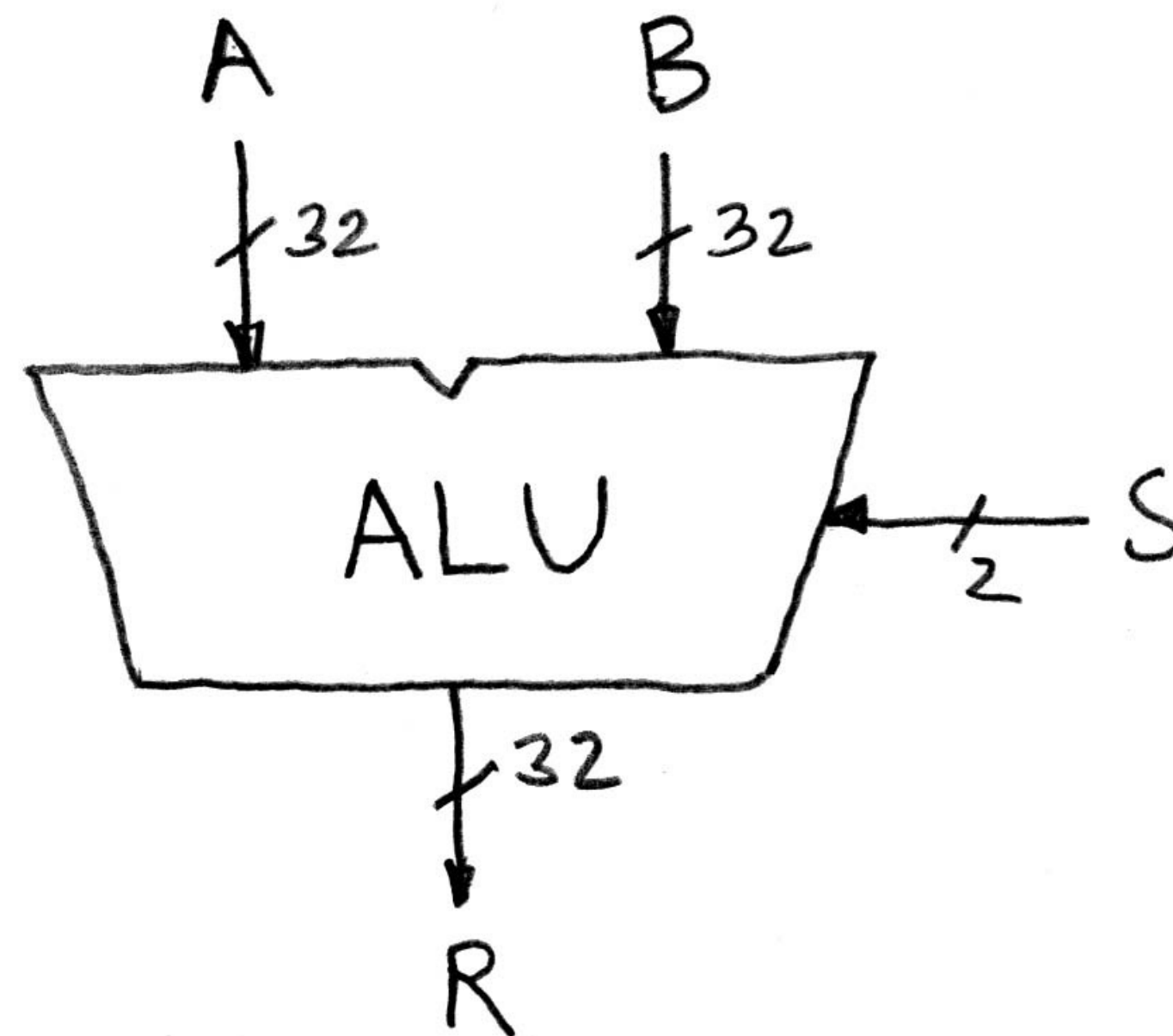
Another way to build 4-1 mux?



Ans: Hierarchically!
Hint: NCAA tourney!

Arithmetic and Logic Unit

- Most processors contain a special logic block called the “Arithmetic and Logic Unit” (ALU)
- We’ll show you an easy one that does ADD, SUB, bitwise AND, bitwise OR



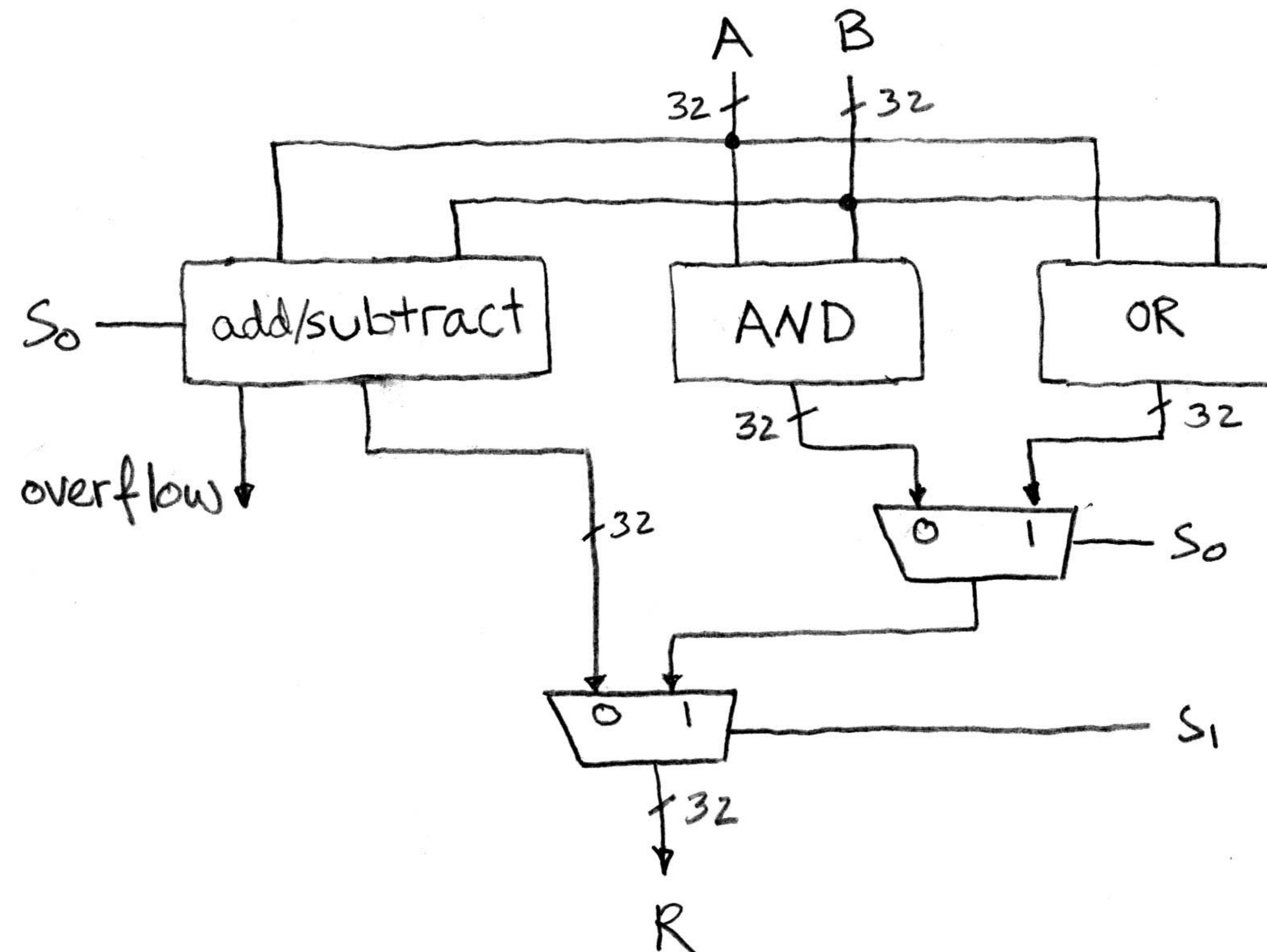
when $S=00$, $R=A+B$

when $S=01$, $R=A-B$

when $S=10$, $R=A \text{ AND } B$

when $S=11$, $R=A \text{ OR } B$

Our simple ALU



How to design Adder/Subtractor?

- Truth-table, then determine canonical form, then minimize and implement as we've seen before
- Look at breaking the problem down into smaller pieces that we can cascade or hierarchically layer

Adder/Subtractor – One-bit adder LSB...

$$\begin{array}{rcccc} & a_3 & a_2 & a_1 & a_0 \\ + & b_3 & b_2 & b_1 & b_0 \\ \hline s_3 & s_2 & s_1 & s_0 \end{array}$$

a ₀	b ₀	s ₀	c ₁
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$s_0 =$$

$$c_1 =$$

Adder/Subtractor – One-bit adder (1/2)...

				a_i	b_i	c_i	s_i	c_{i+1}
				0	0	0	0	0
				0	0	1	1	0
				0	1	0	1	0
				0	1	1	0	1
				1	0	0	1	0
				1	0	1	0	1
				1	1	0	0	1
				1	1	1	1	1

a_3

a_2

a_1

a_0

b_3

b_2

b_1

b_0

s_3

s_2

s_1

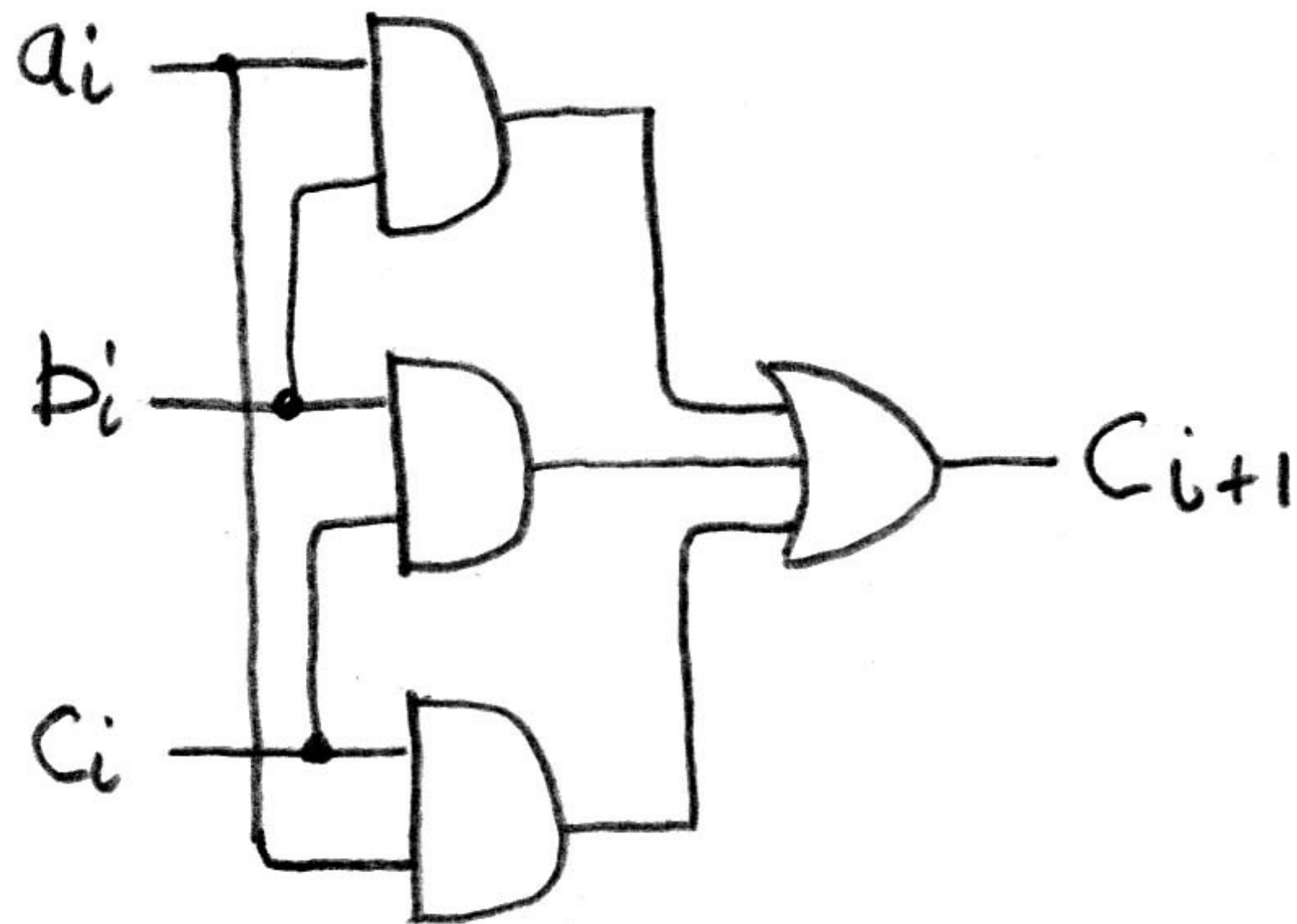
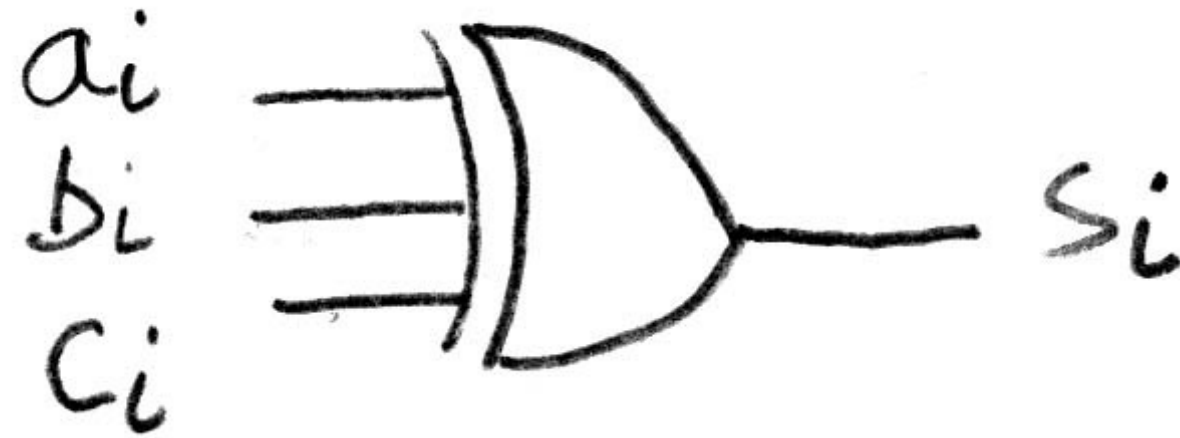
s_0

$+$

s_i $=$

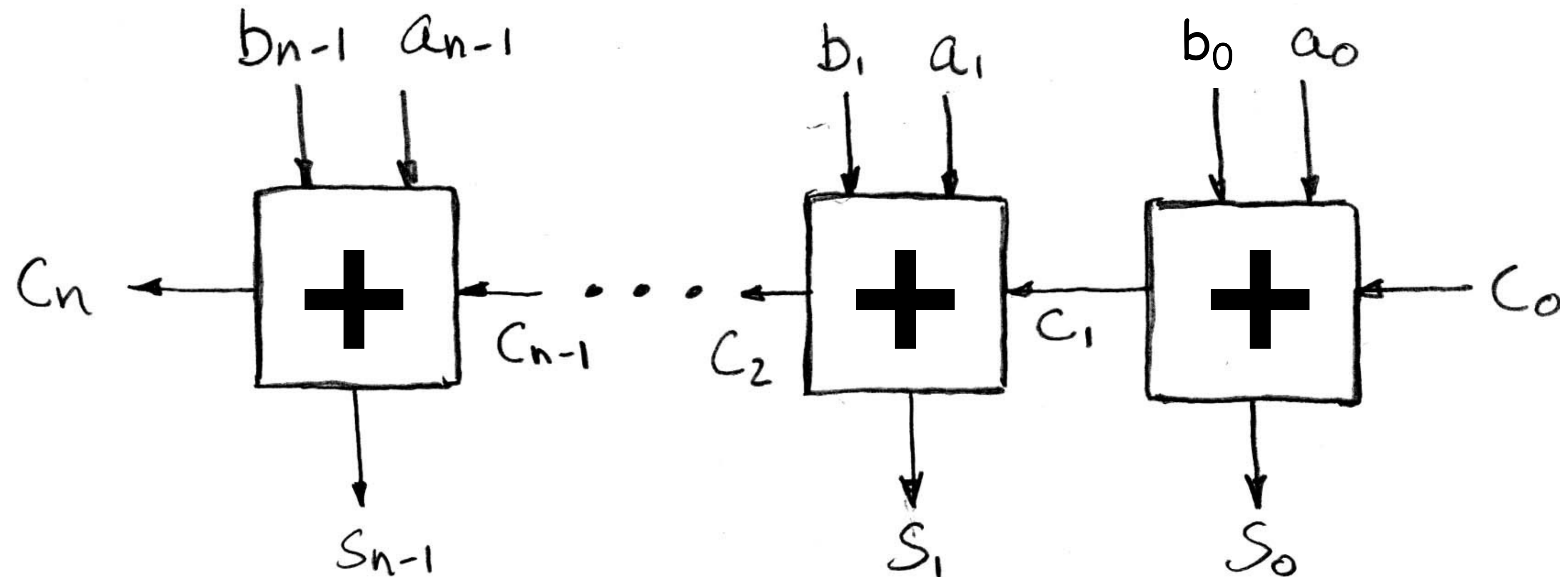
c_{i+1} $=$

Adder/Subtractor – One-bit adder (2/2)



$$S_i = \text{XOR}(a_i, b_i, c_i)$$
$$C_{i+1} = \text{MAJ}(a_i, b_i, c_i) = a_i b_i + a_i c_i + b_i c_i$$

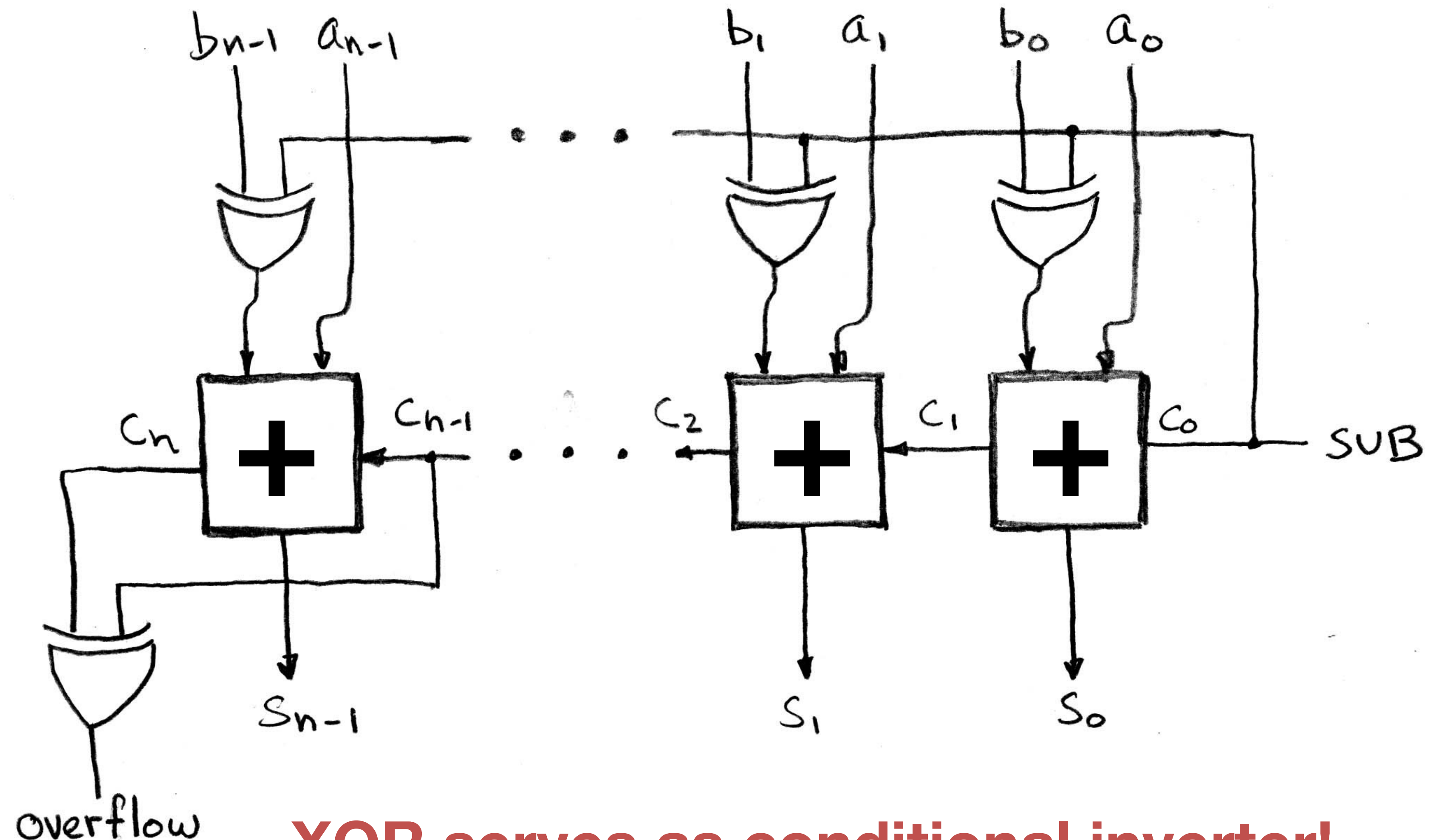
N 1-bit adders \Rightarrow 1 N-bit adder



What about overflow?
Overflow = c_n ?

Extremely Clever Adder/Subtractor: "Invert and add one"

x	y	XOR(x,y)
0	0	0
0	1	1
1	0	1
1	1	0



XOR serves as conditional inverter!

In Conclusion

- Finite State Machines have clocked state elements plus combinational logic to describe transition between states
 - Clocks synchronize D-FF change (Setup and Hold times important!)
- Standard combinational functional unit blocks built hierarchically from subcomponents