10/28/26

COMP 311: *Computer Organization!*

Lecture 19: Computer Performance Metrics



The University
of North Carolina
at Chapel Hill

https://tinyurl.com/comp311-fa26

Logistics Update



- Lab 3 is out
  - Due 11/6
- WA4
  - Due 11/2
- Quiz Corrections!

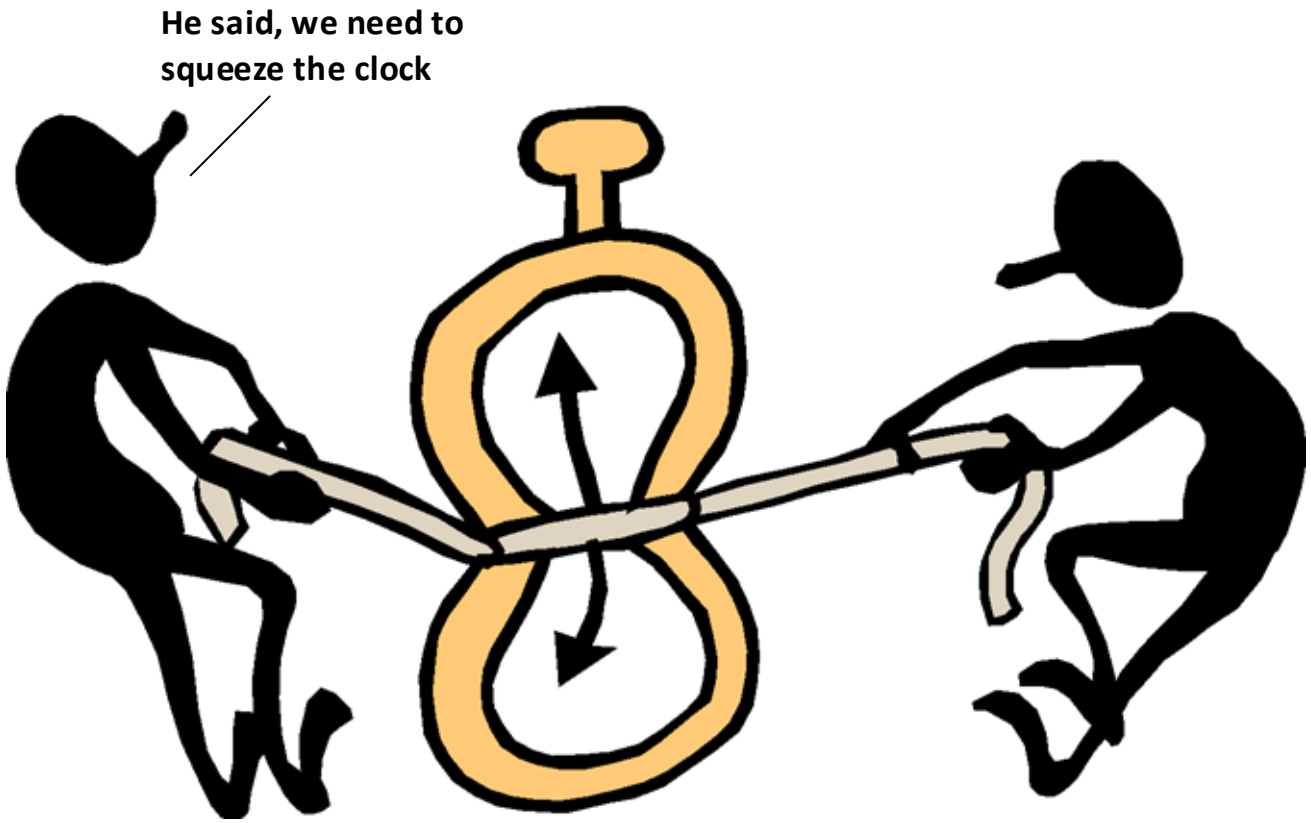
Plan for Class Today



- 3:35-4:00: Computer Performance Terms
- 4:00-4:15: Timing Analysis with Multiple Computations
- 4:15-4:20: BREAK
- 4:20-4:50: Practice

Guiding Question ☺

How does hardware remember information, and how can we predict when its outputs change?



Computer Performance

Why study performance?



- Helps us to make intelligent choices
- Helps us see through marketing hype
- Affects computer organization (pipelining, caches, etc.)
- Why is some hardware faster than others?
- What factors of system performance are hardware related?
  - Do we need a new machine,
  - more memory
  - a better compiler
  - or a new OS?
- How does a machine's instruction set affect its performance?



Performance Metrics



- **Latency**: time from an input to its corresponding output
  - How long does it take for my program to run?
  - How long must I wait after typing `return` for the result?
- **Throughput:** The rate at which new outputs are generated
  - How many calculations per second?
  - What is the average execution rate of my program?
  - How much work is getting done?

By running a program on 20 different input files on the fastest available processor, what performance metric do we improve?

By running our program simultaneously on 20 CPU's, each assigned an input file, what performance metric do we improve?

What airplane has the best performance?

Aircraft	Passengers	Range (miles)	Speed (mph)
Boeing 737-100	132	630	598
Boeing 747	470	4150	610
BAC/Sud Concorde	101	4000	1350
Douglas DC-8-50	146	8720	544



How much faster is the Concorde than the 747? **2.213 X**

How much larger is the 747's capacity than the Concorde? **4.65 X**

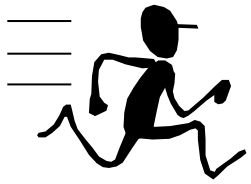
It is roughly 4000 miles from Raleigh to Paris. What is the throughput of the 747 in passengers/hr? The Concorde?

470(610)/4000 = 71.65 pass/hr

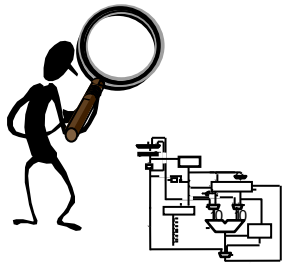
What is the latency of the 747? **4000/610 = 6.56 hr/pass**

The Concorde? **4000/1350 = 2.96 hr/pass**

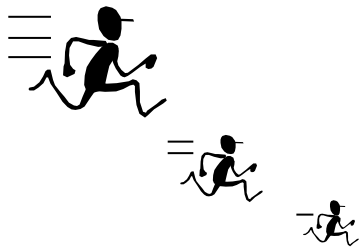
Performance Tradeoffs



Maximum Performance: measured by the "number of instructions executed per second"



Minimum Cost: determined by the size-of-the-circuit/number-of-components-used plus power/cooling costs



Best Price/Performance: measured by the ratio of CPU-cost to number of instructions executed per sec.

Performance/Watt instructions per second per watt

Execution Time

Elapsed Time/Wall Clock Time

counts everything (disk and memory accesses, I/O , etc.)

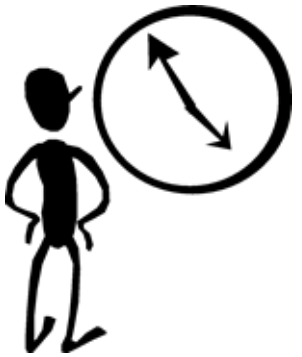
a useful number, but often not good for comparison purposes

CPU time

Doesn't include I/O or time spent running other

Programs can be broken up into system time,

and user time



Our focus: user CPU time

Time spent executing actual instructions of "our" program

Definition of Performance

For some program running on machine X,

$Performance_X$ = Program Executions / $Time_X$ (executions/sec)

"X is N times faster than Y"

$Performance_X$ / $Performance_Y$ = N



Problem:

Machine A runs a program in 20 seconds

Machine B runs the same program in 25 seconds

$Performance_A$ = 1/20 $Performance_B$ = 1/25

Machine A is (1/20)/(1/25) = 1.25 times faster than Machine B

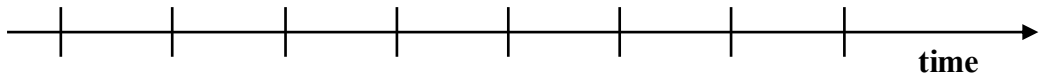
Program Clock Cycles



Instead of reporting execution time in seconds, we can also use cycle counts

(sec/program) * (clocks/sec) = clocks/program

Clocks are when machine-state changes (synchronous abstraction):



cycle time = time between rising edges of the clock = seconds per clock
clock rate (frequency) = clocks per second (1 Hz. = 1 clock/sec)

A 200 Mhz. clock has a $1/(200 * 10^6) = 5.0 \times 10^{-9}$ cycle time = 5 nS

Overclocking: improves performance (seconds/program) by decreasing the cycle time (seconds/cycle), while hoping that the functional blocks continue to operate as specified.

Standard Computer Performance Measures



Millions of Instructions per Second

Frequency in Hz

$$\text{MIPS} = \frac{1}{10^6} \frac{\text{clocks / second}}{\text{clocks / instruction}}$$

CPI (Average Clocks Per Instruction)

CPI = cycles / instruc

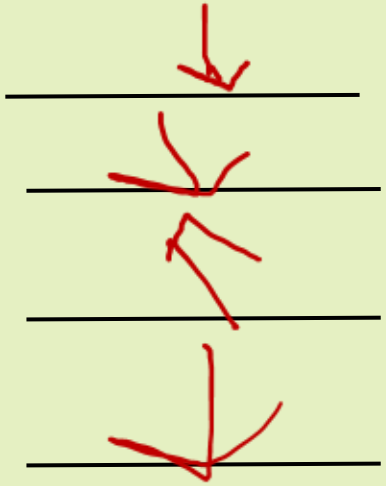
How to improve performance?



(sec/program) (clocks/sec) = clocks/program

$$\textbf{MIPS} = \frac{1}{10^6} \frac{\text{clocks / second}}{\text{clocks / instruction}}$$

So, to improve performance (everything else being equal) you can either



the # of required clocks for a program, or

the clock cycle time or, said another way,

the clock rate.

the CPI (average clocks per instruction)

Clocks == clock cycles

How to improve performance?



(sec/program) (clocks/sec) = clocks/program

$$\textbf{MIPS} = \frac{1}{10^6} \frac{\text{clocks / second}}{\text{clocks / instruction}}$$

So, to improve performance (everything else being equal) you can either

Decrease (improve ISA) the # of required clocks for a program, or

Decrease the clock cycle time or, said another way,

Increase the clock rate.

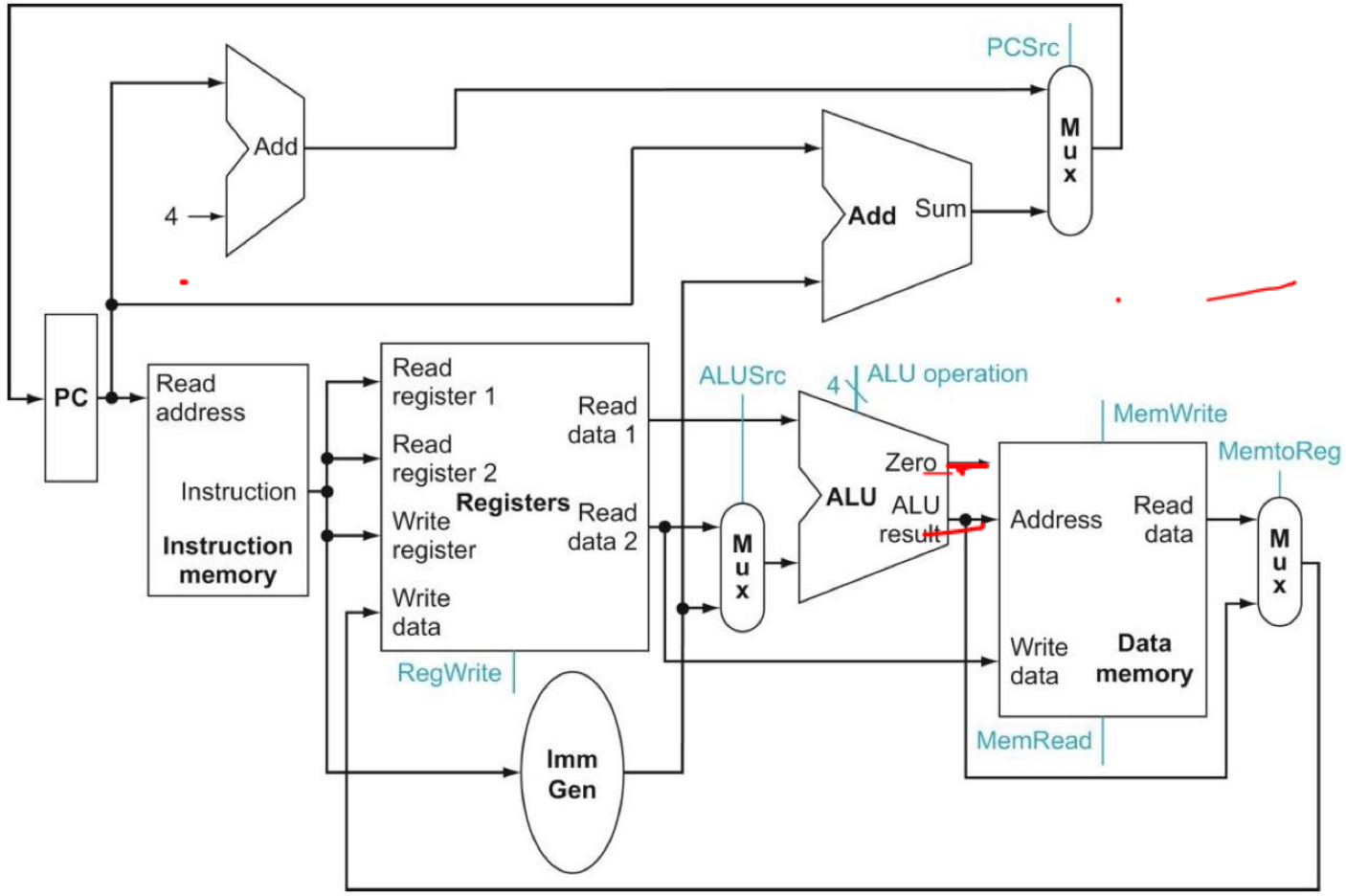
Decrease the CPI (average clocks per instruction)



More Timing Analysis

Where are we headed?

Figure 4.11 The simple datapath for the core RISC-V architecture

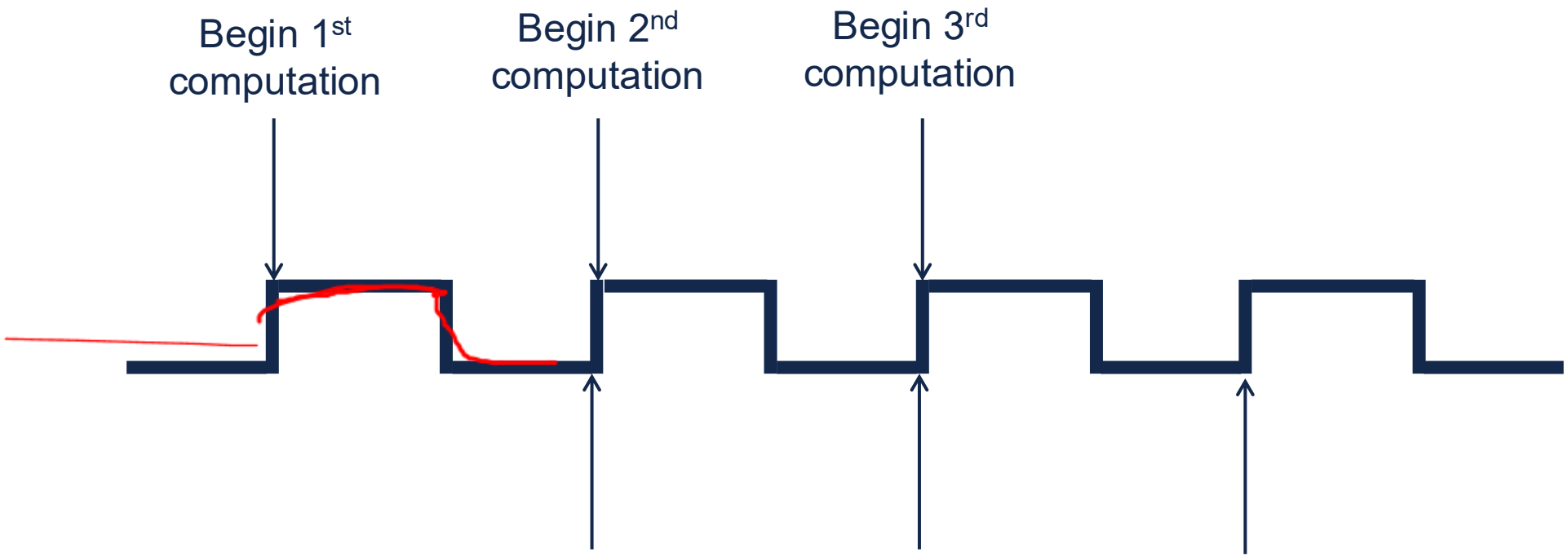


Our ultimate goal: a full CPU

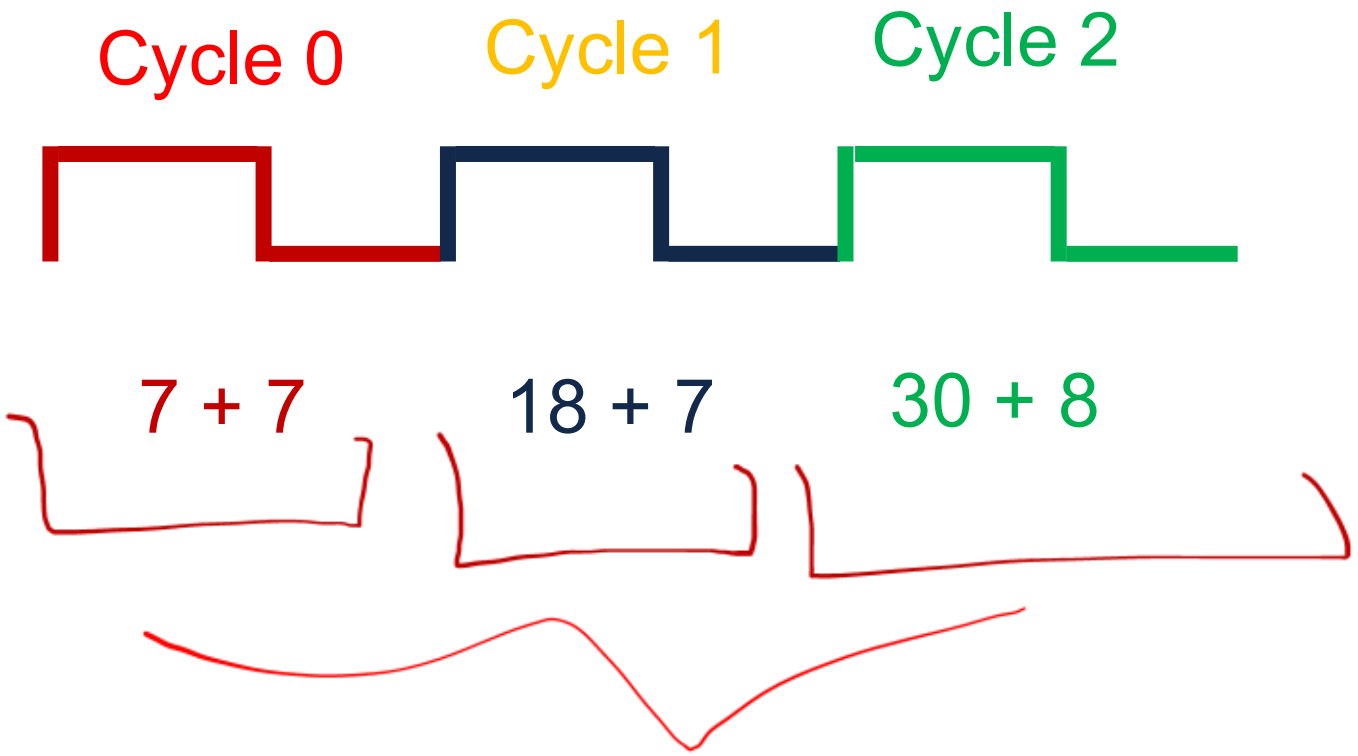


- Our goal: understand how fast we can safely clock a CPU
- That depends on how long signals take to travel through its circuits. We started small. Last time:
  - how individual gate delays determine when a circuit's output is ready
- Today: how registers and computation work together:
  - When inputs change, when does the circuit compute the new result, and on which clock edge is that result stored?

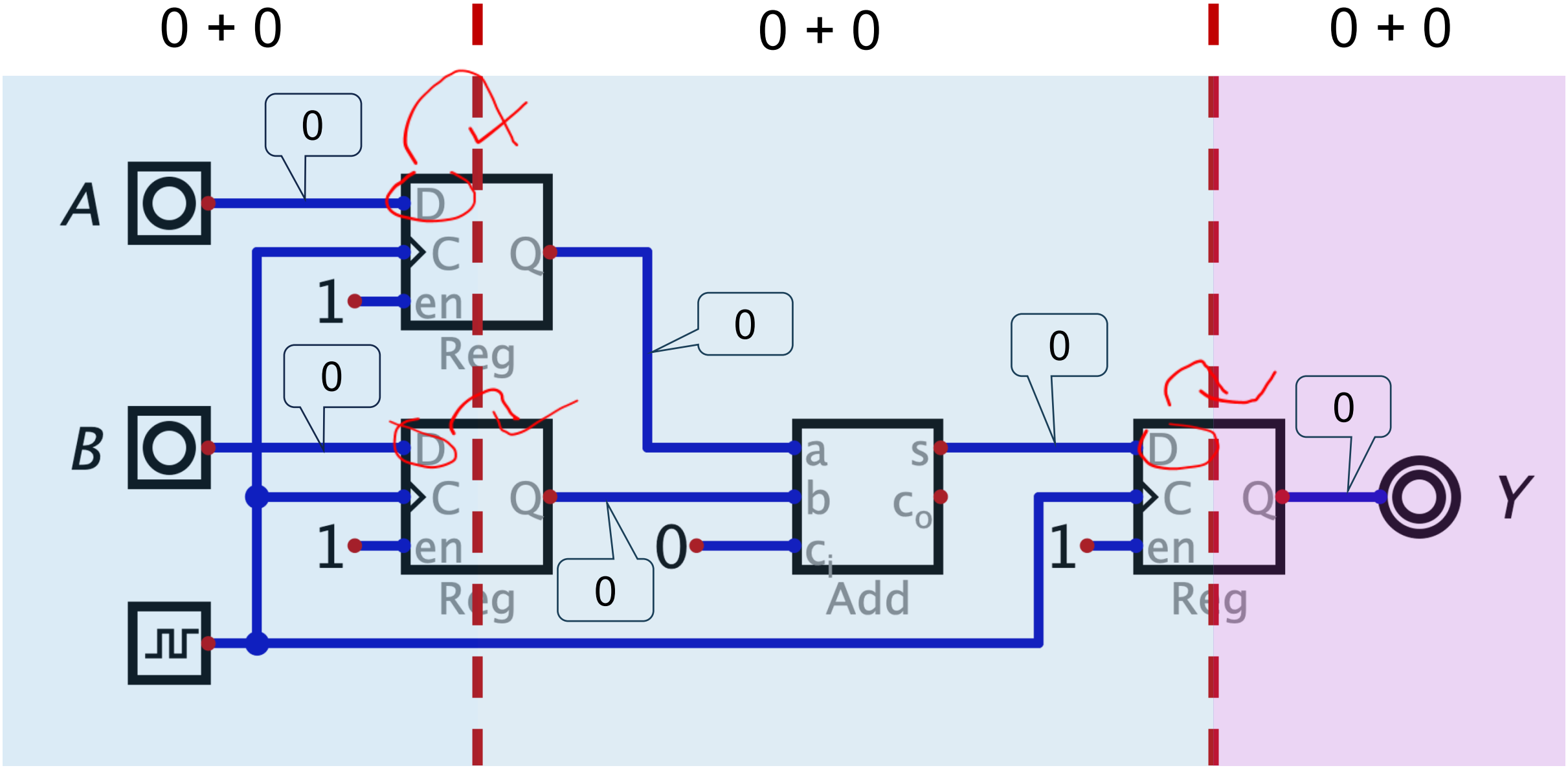
Timing


Begin 1st computation
Begin 2nd computation
Begin 3rd computation
1st computation should be completed by the next rising edge of the clock
2nd computation should be completed by the next rising edge of the clock
3rd computation should be completed by the next rising edge of the clock

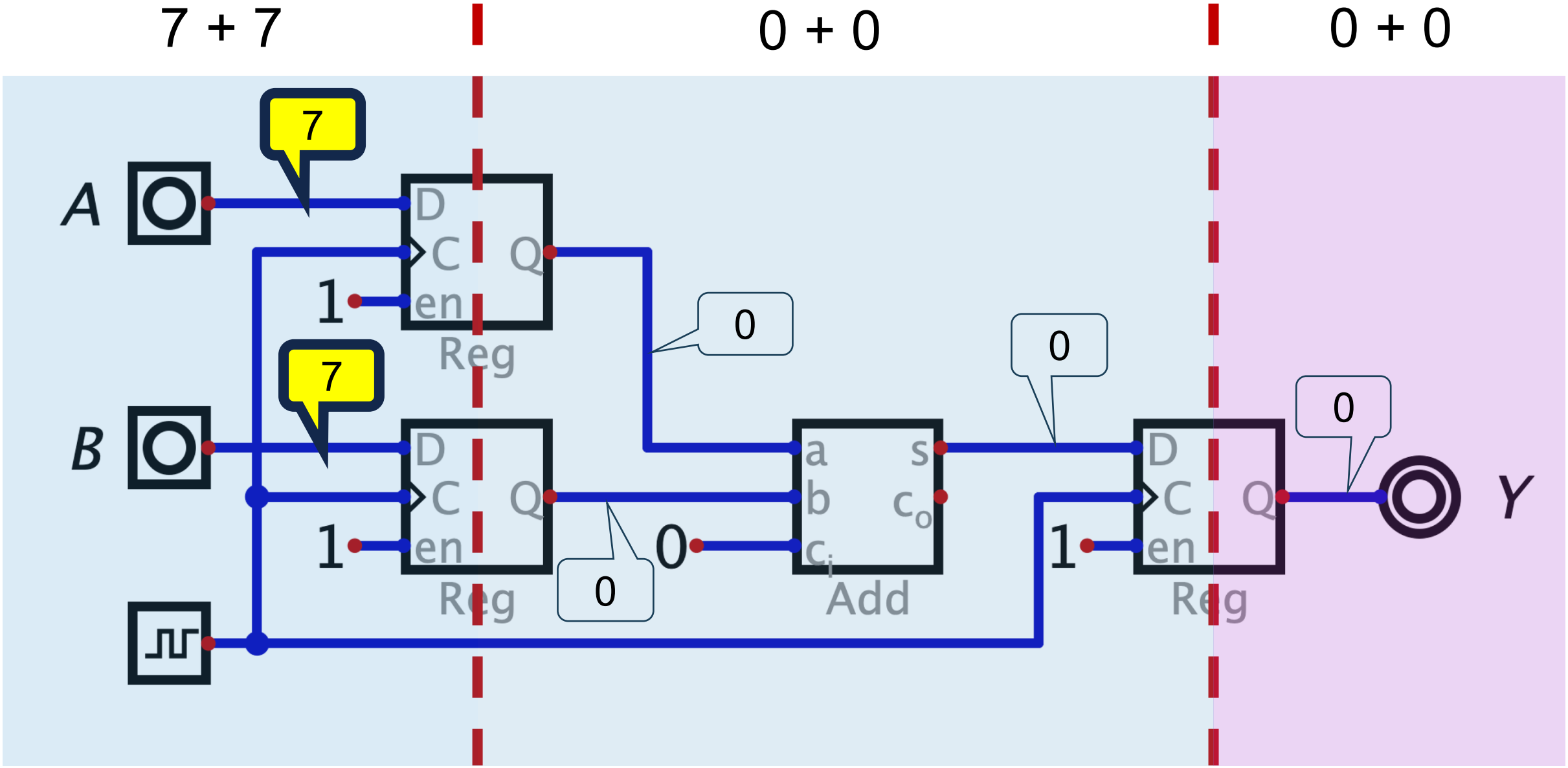
Timing



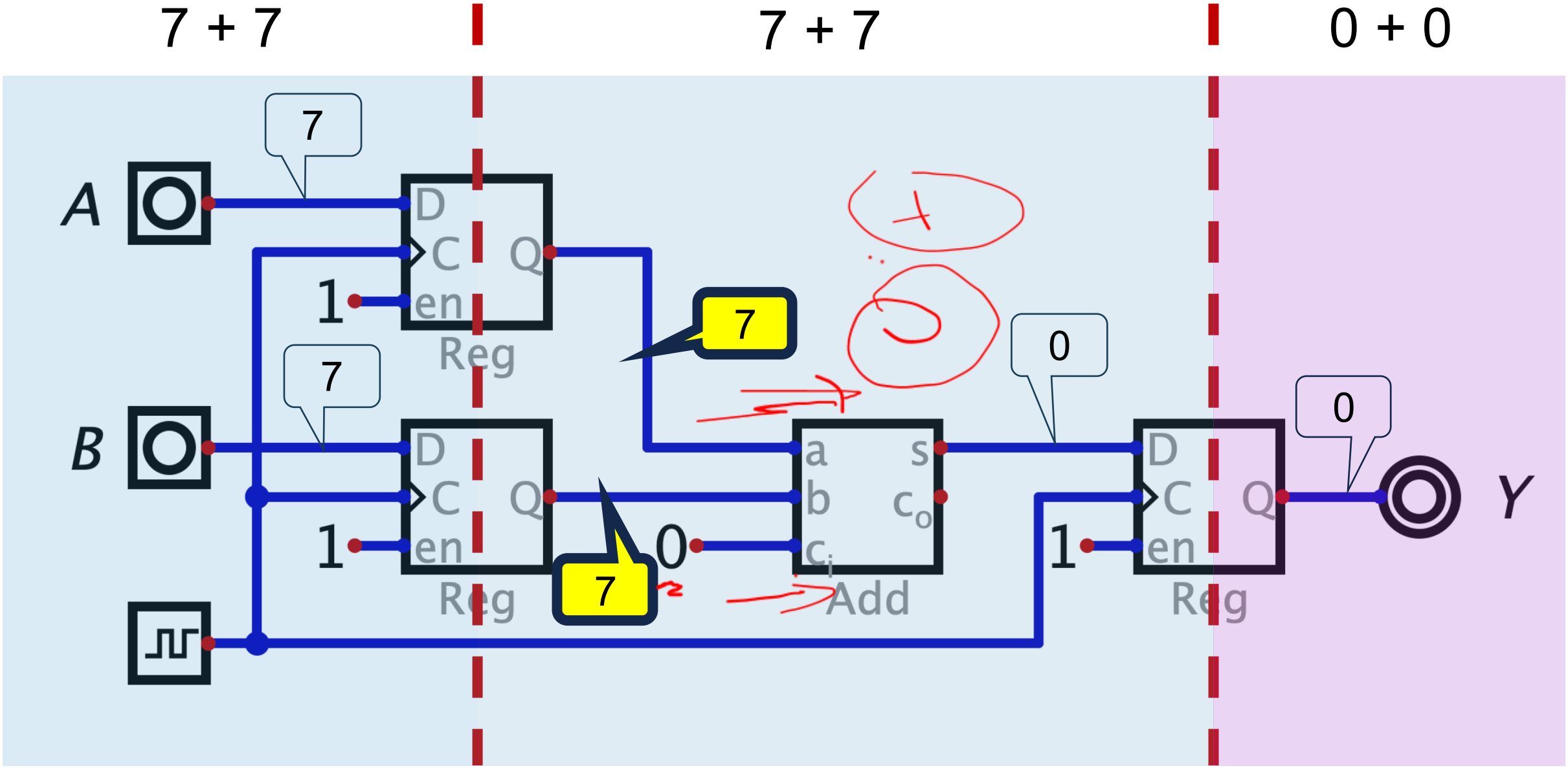
Starting State



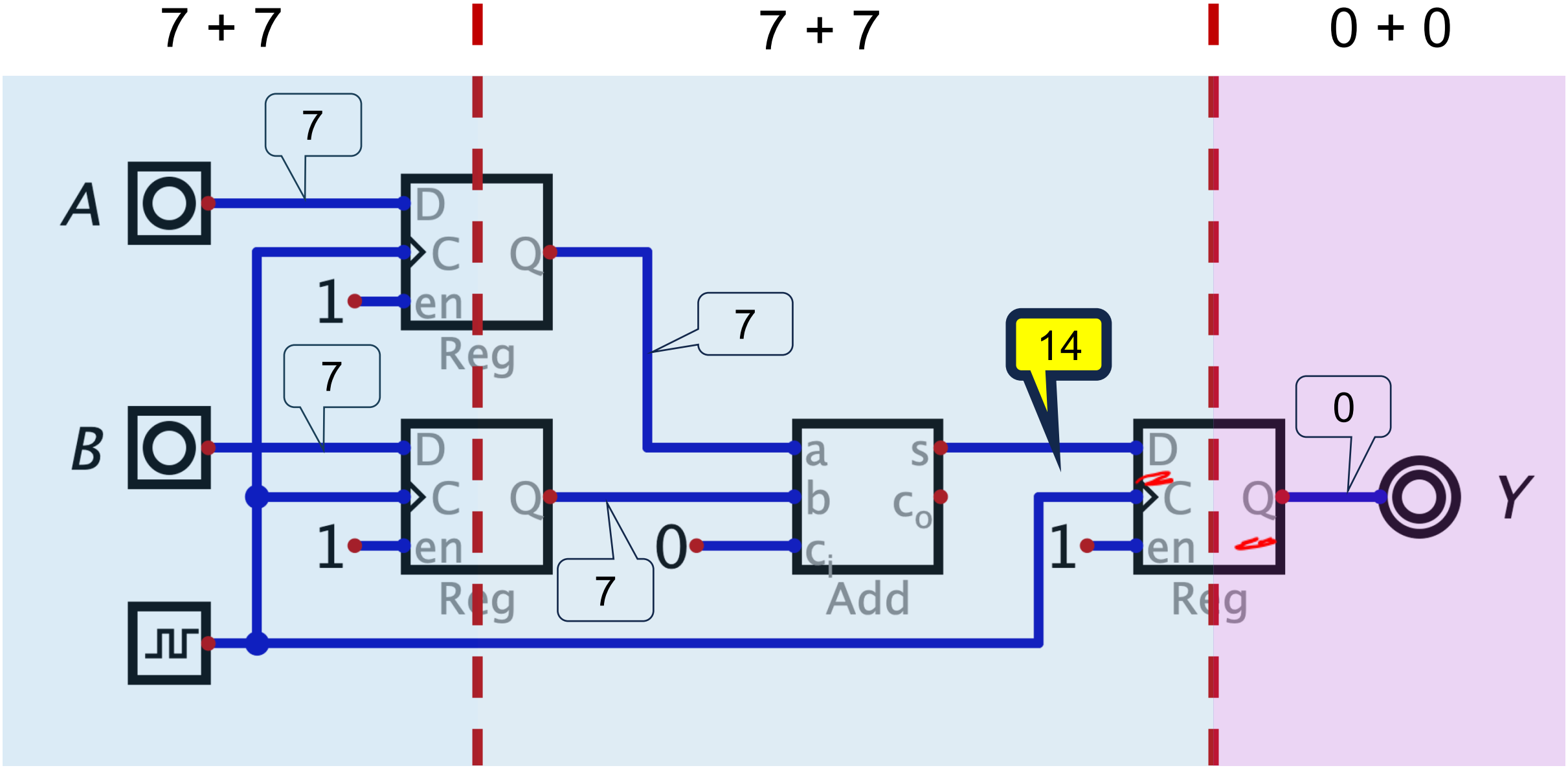
Cycle 0: On rising edge



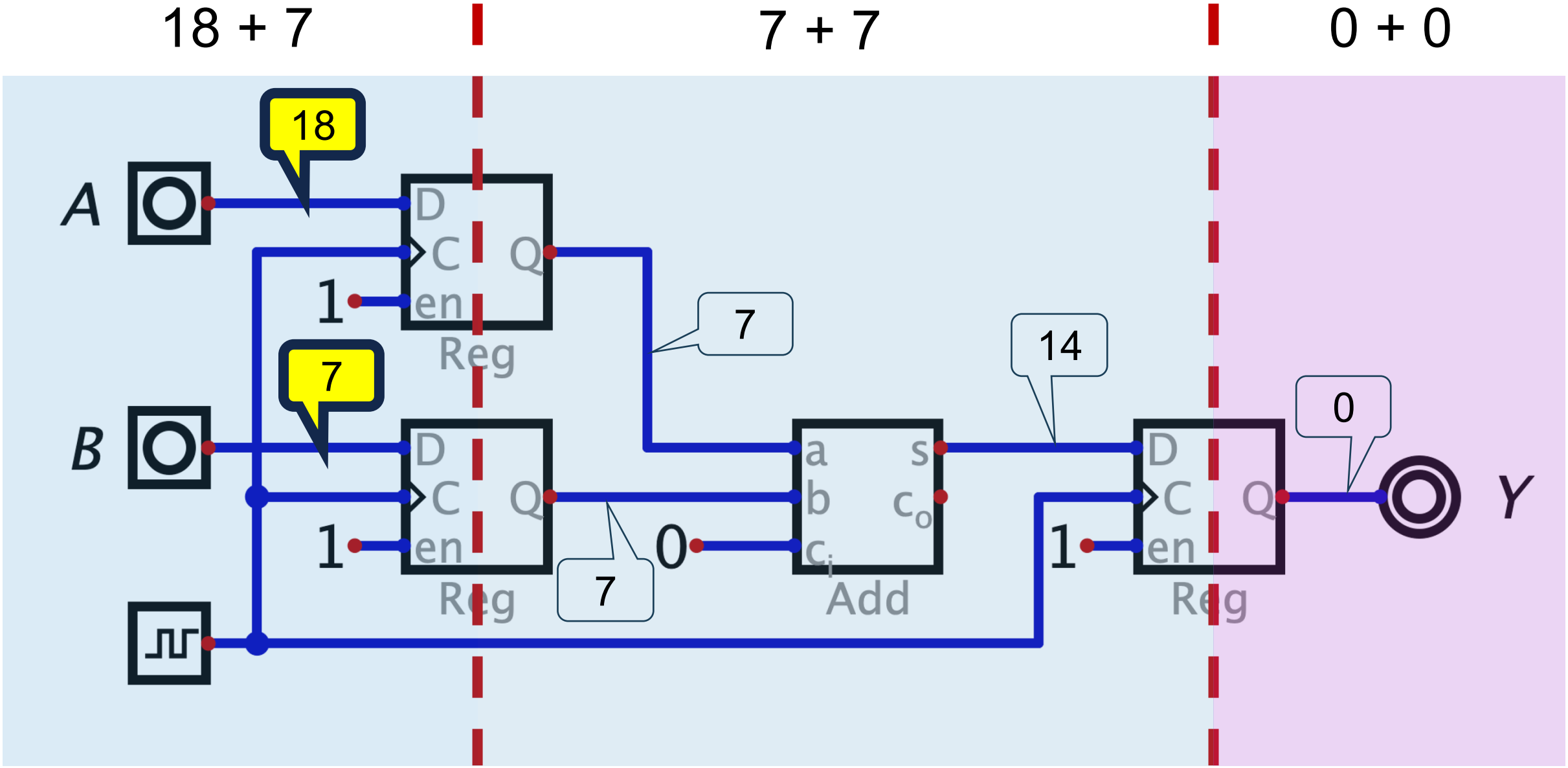
Cycle 0: After clk-to-q delay



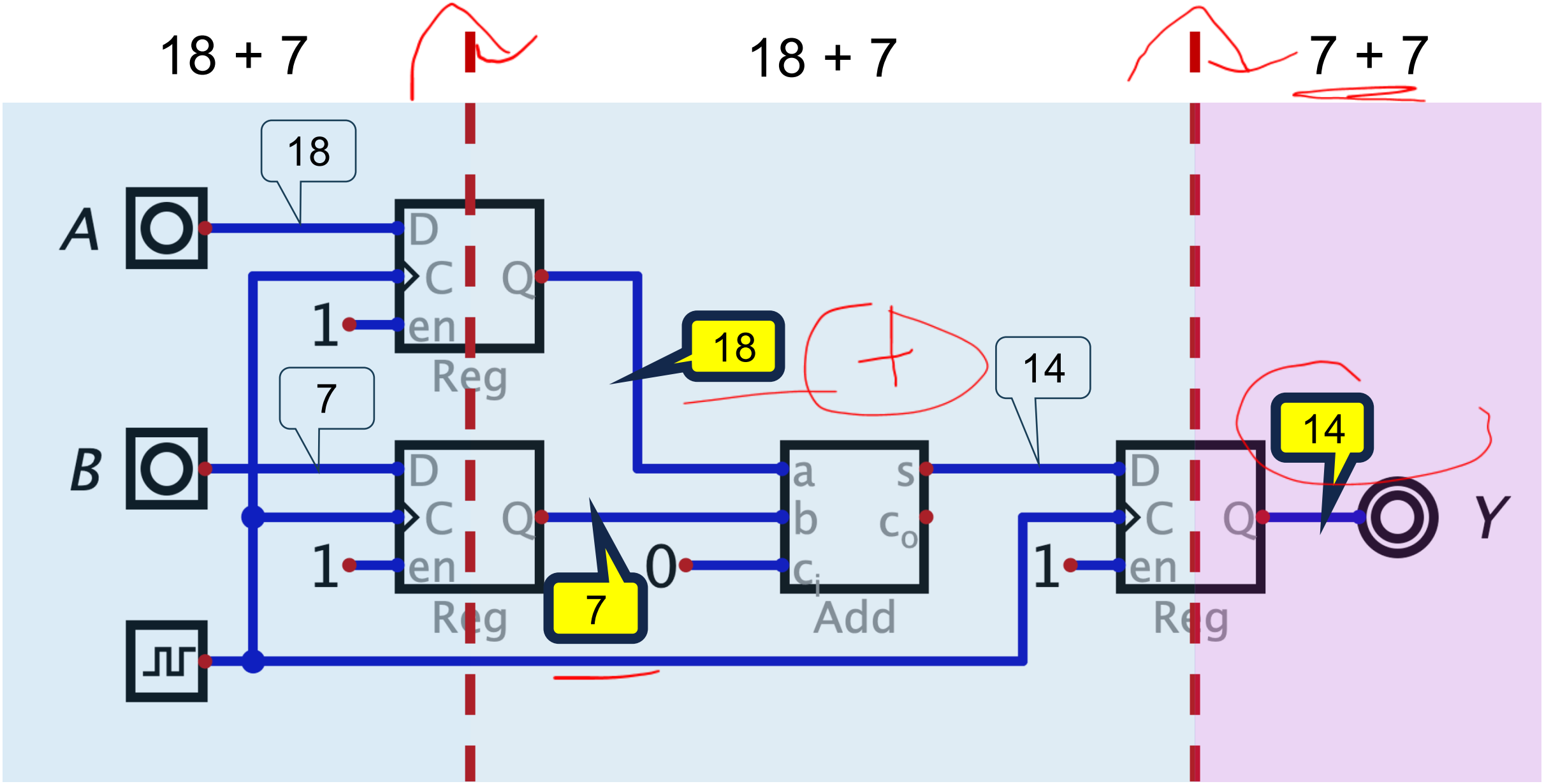
Cycle 0: After adder propagation delay



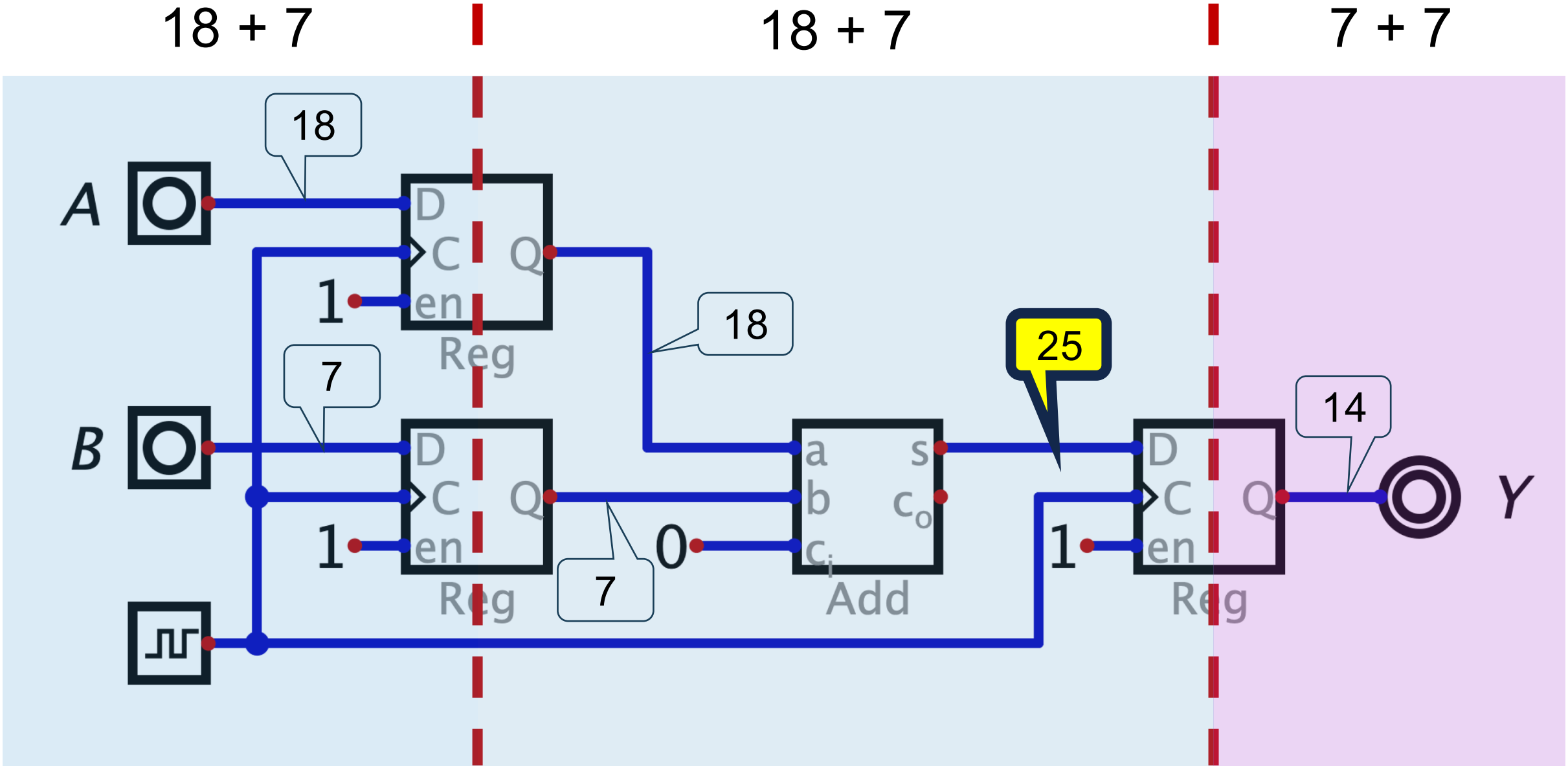
Cycle 1: On rising edge



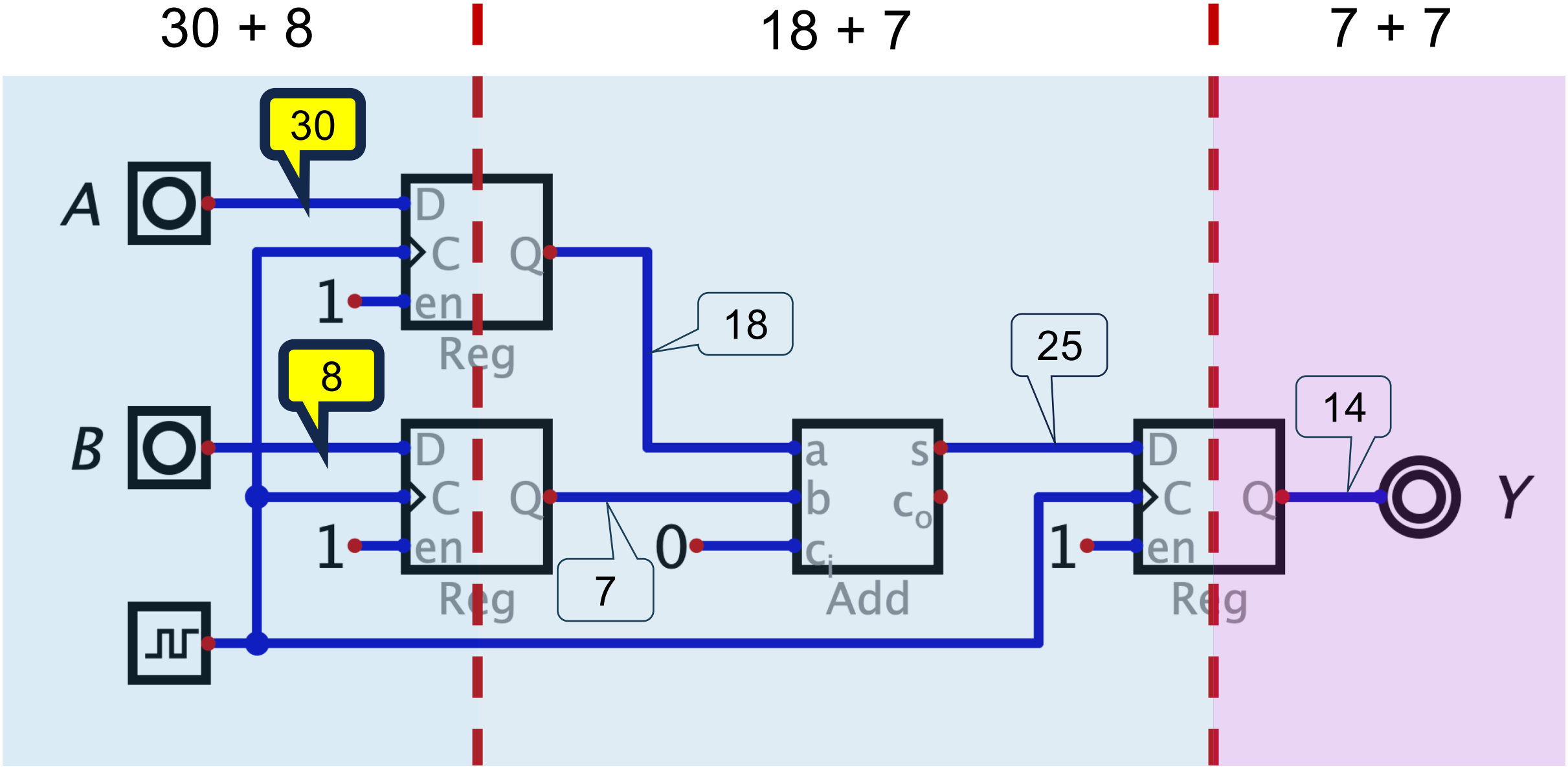
Cycle 1: After clk-to-q delay



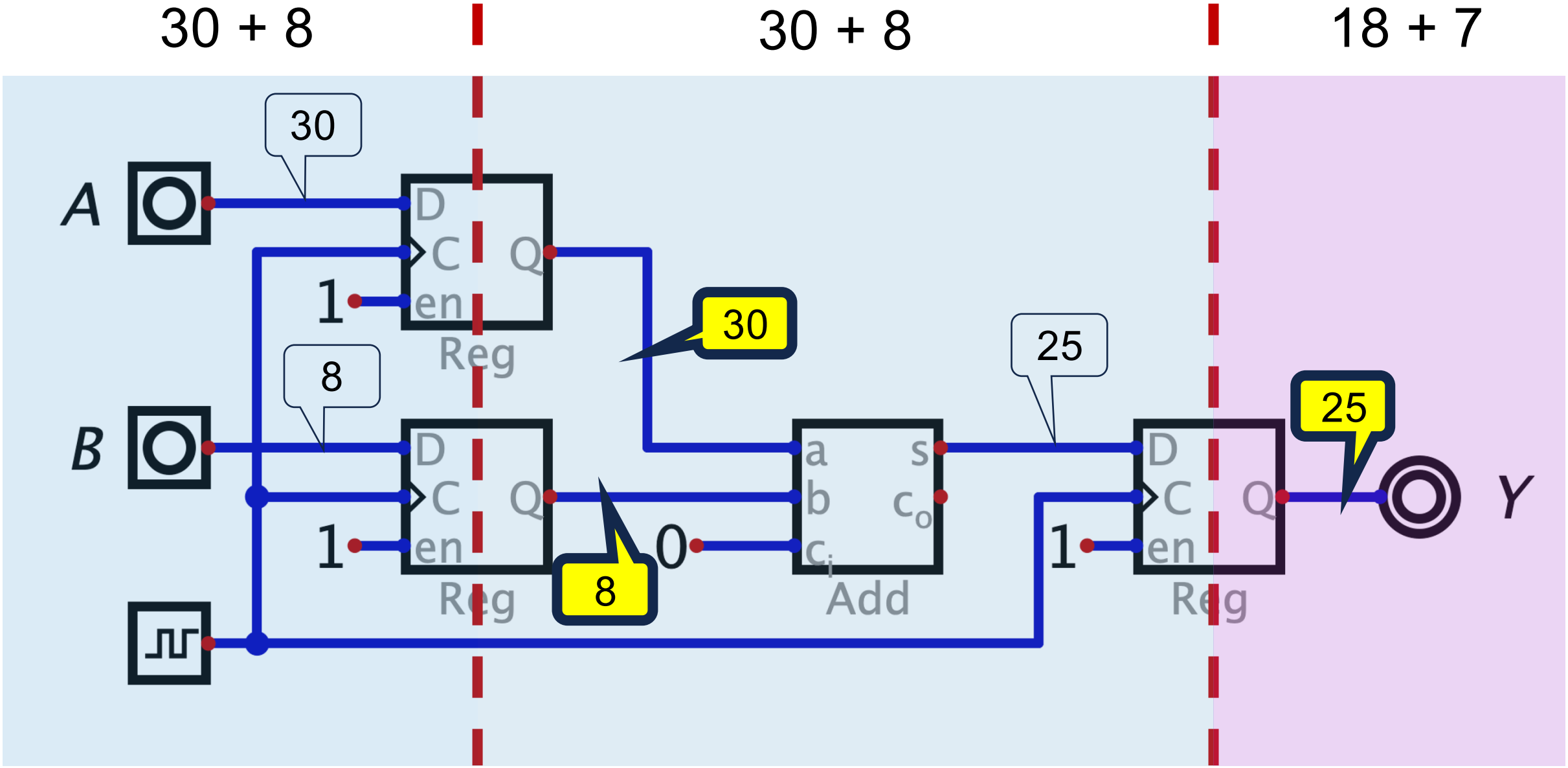
Cycle 1: After adder propagation delay



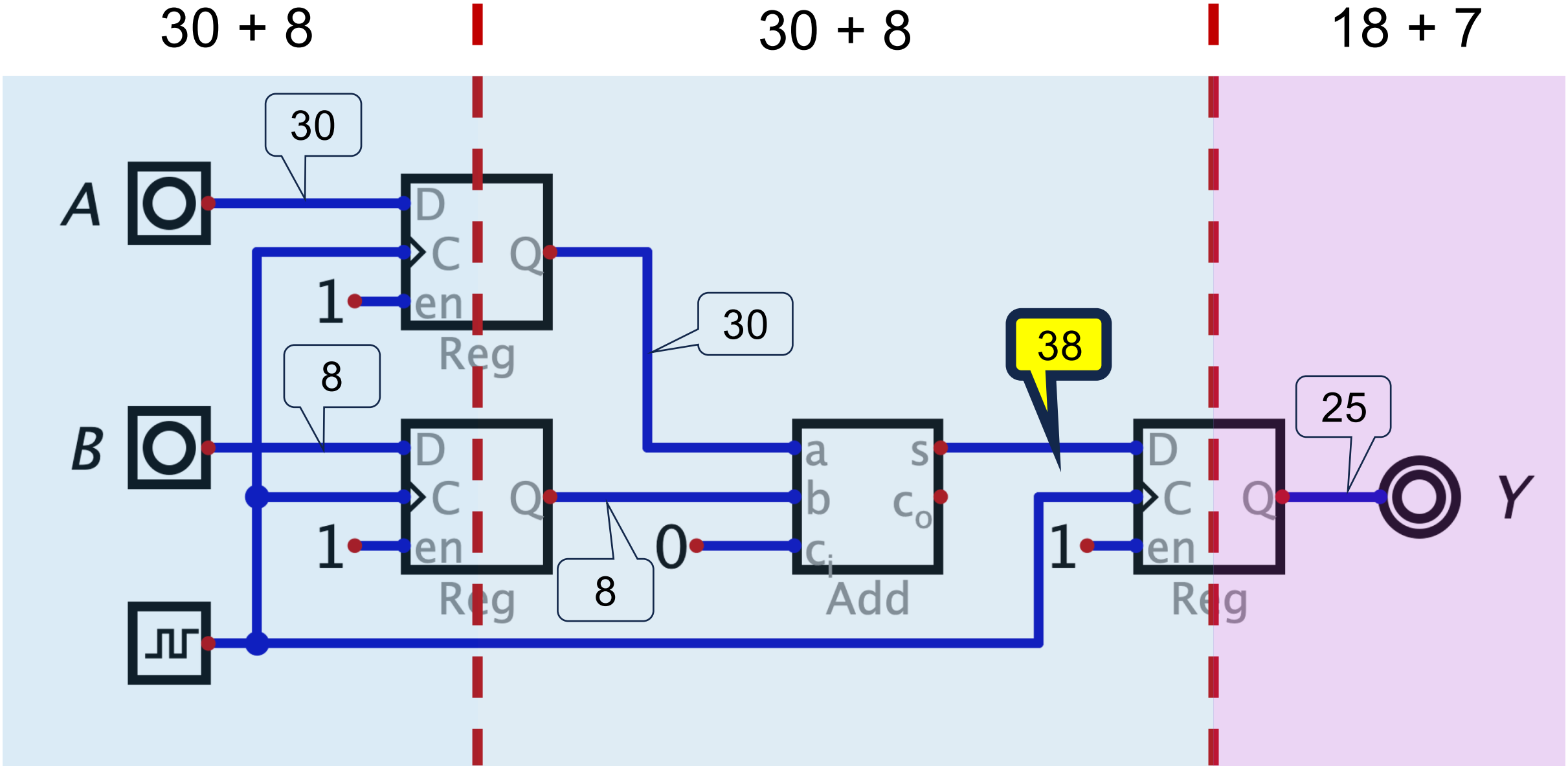
Cycle 2: On Rising Edge



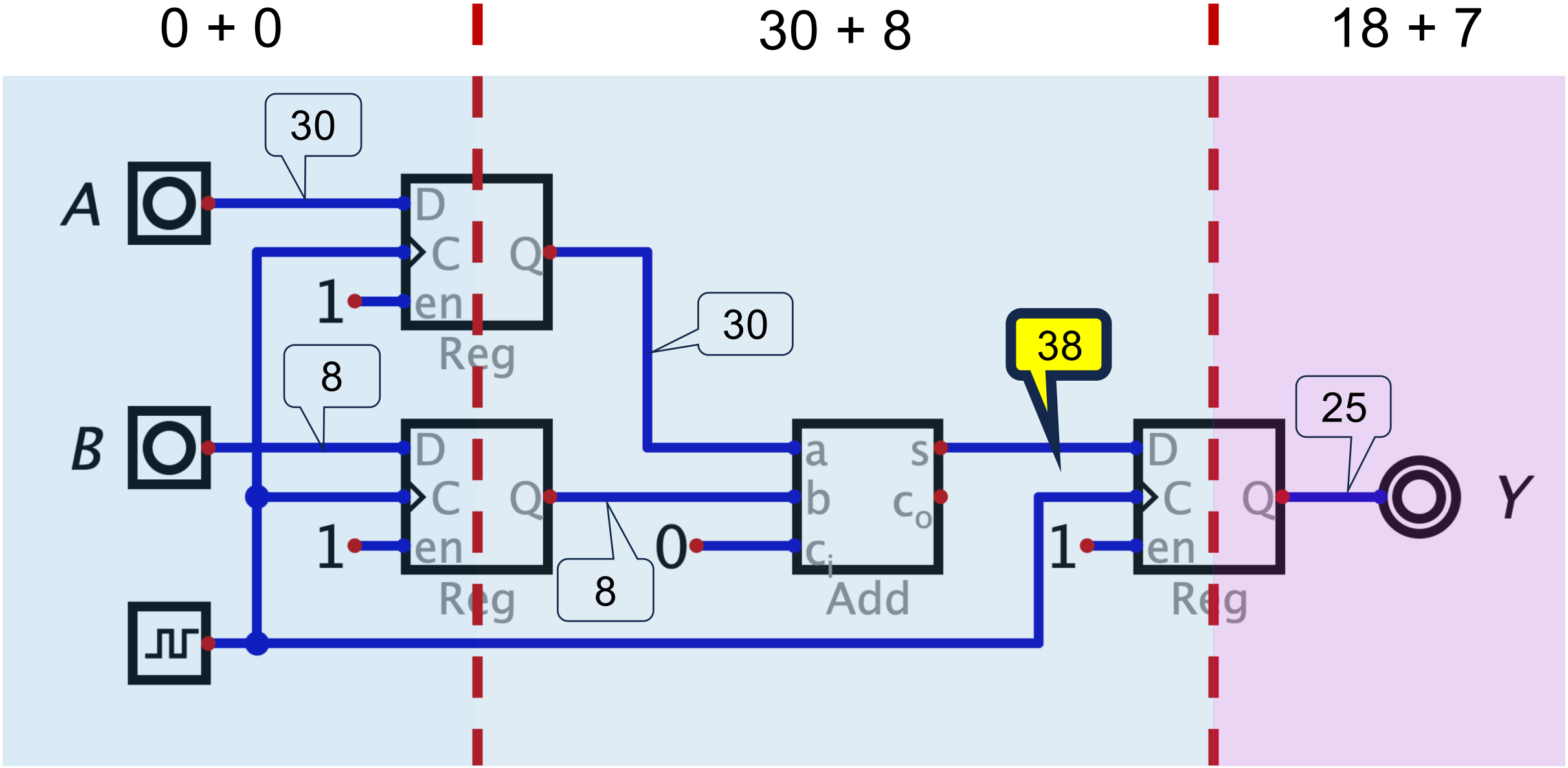
Cycle 2: After clk-to-q Delay



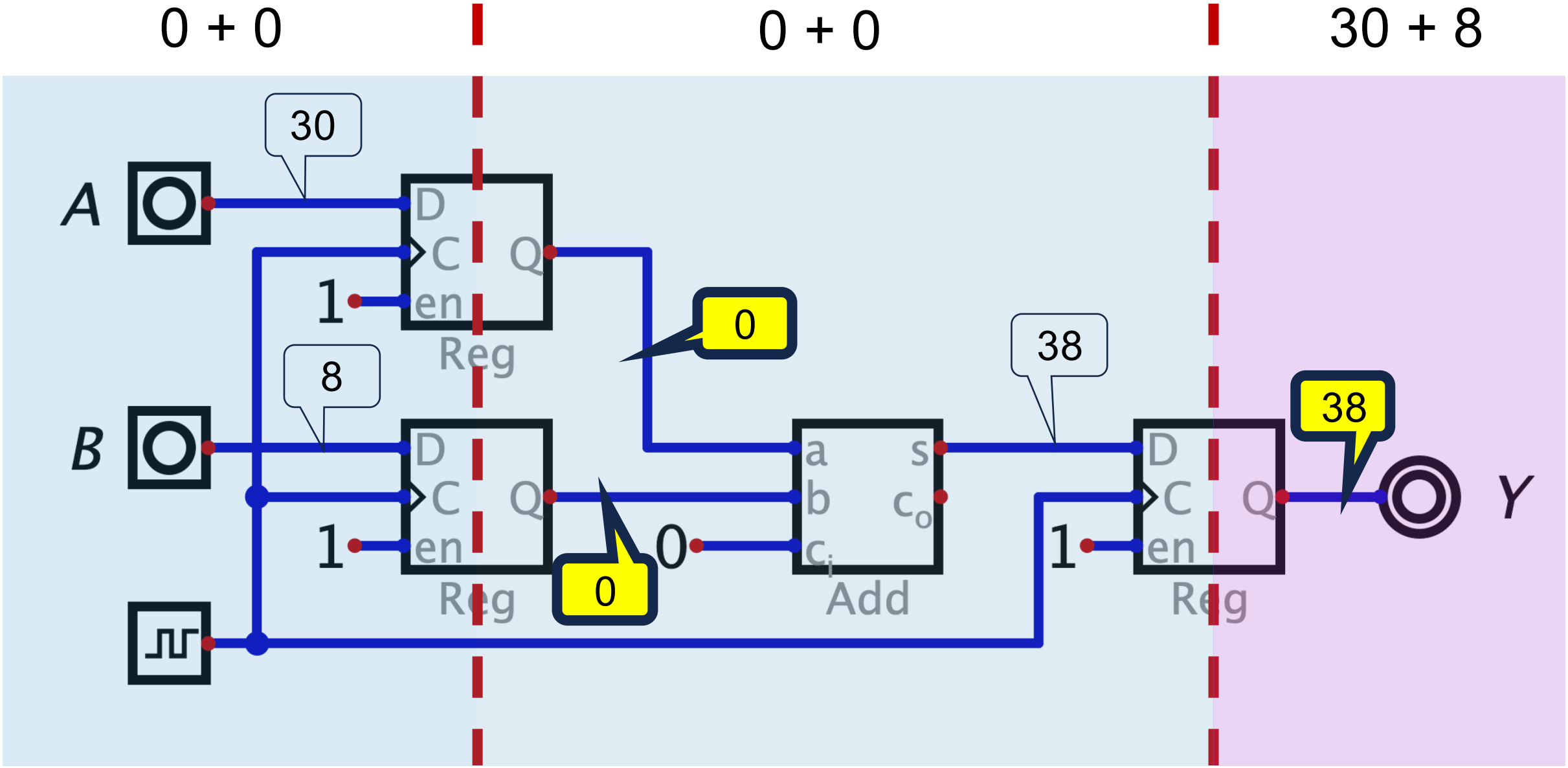
Cycle 2: After Adder Propagation Delay



Cycle 3: On Rising Edge



Cycle 3: After clk-to-q Delay



Delays



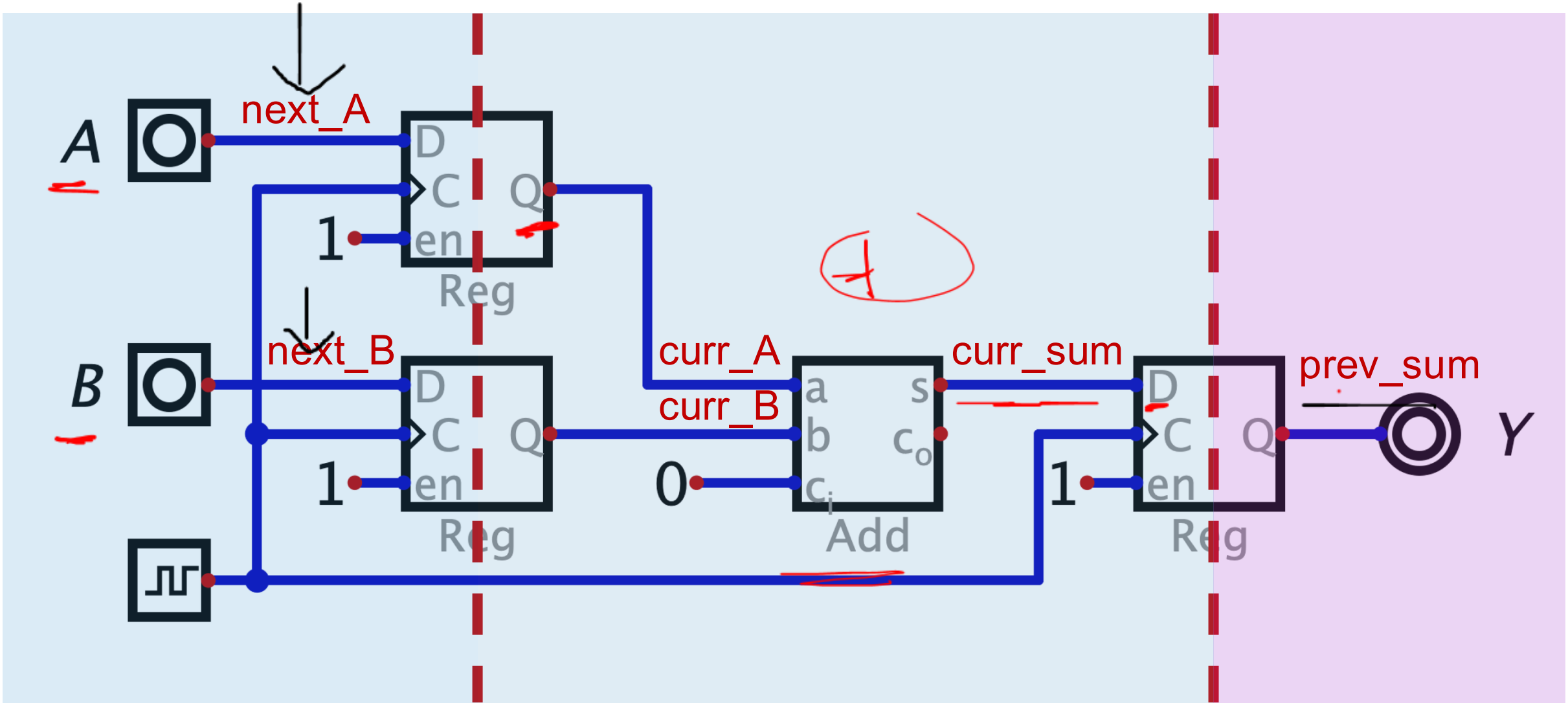
- Adder propagation delay = 600ps
- Register clk-to-q delay = 50ps
- These delays are innate properties of the components
- We can calculate these delays by finding the longest combinational delay
- We cannot change these values without changing the components

Clock Period

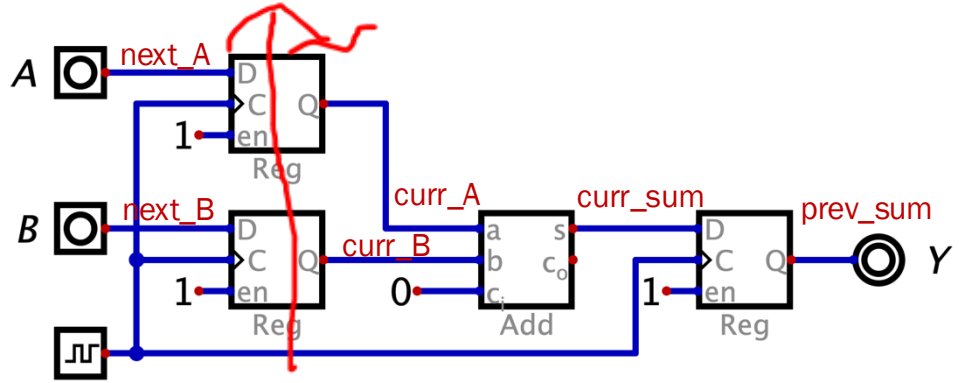


- We can adjust the clock period based on our needs
- There is a minimum clock period at which the circuit will work properly
- We can set our clock period to any value >= min clock period

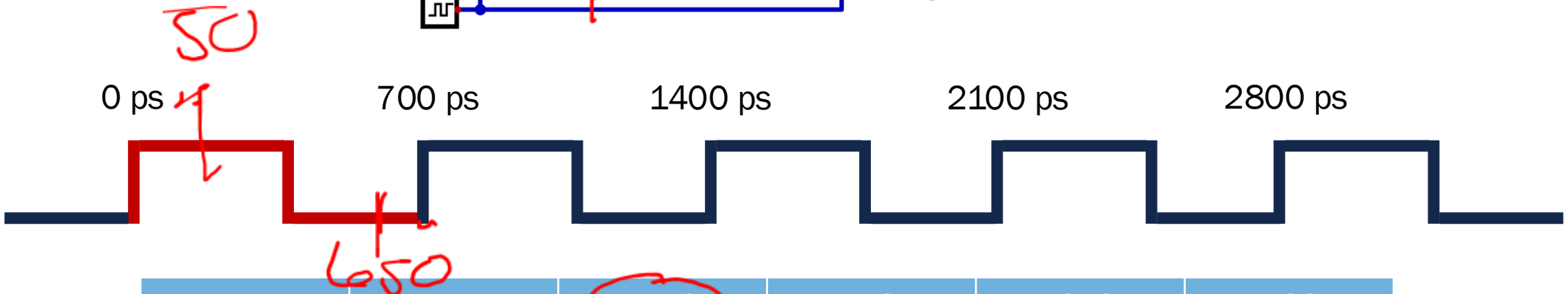
Ex1: Clock period = 700ps, clk-to-q delay = 50ps, adder propagation delay = 600ps



Cycle 0: 7 + 7

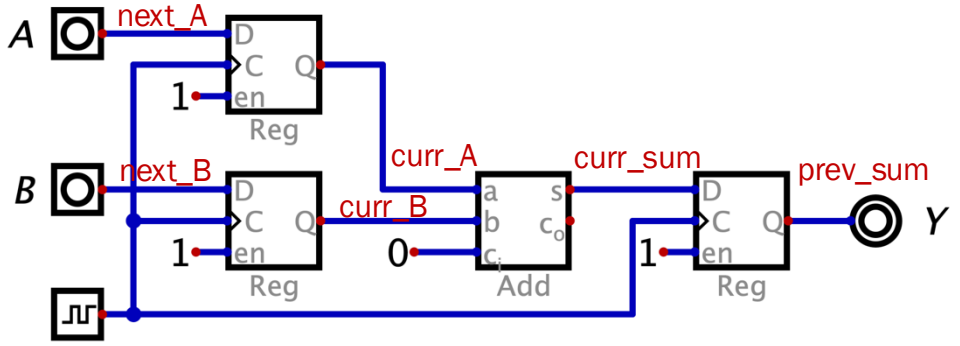


adder propagation delay = 600 ps
clk-to-q delay is 50ps
clock period = 700ps



Signal	Starting State	t = 0 A = 7 B = 7	t = 50ps A = 7 B = 7	t = 650ps A = 7 B = 7	t = 700 ps A = 18 B = 7
next_A	0	7	7	7	18
next_B	0	7	7	7	7
curr_A	0	0	7	7	7
curr_B	0	0	7	7	7
curr_sum	0	0	0	14	14
prev_sum	0	0	0	0	0

Cycle 0: 7 + 7

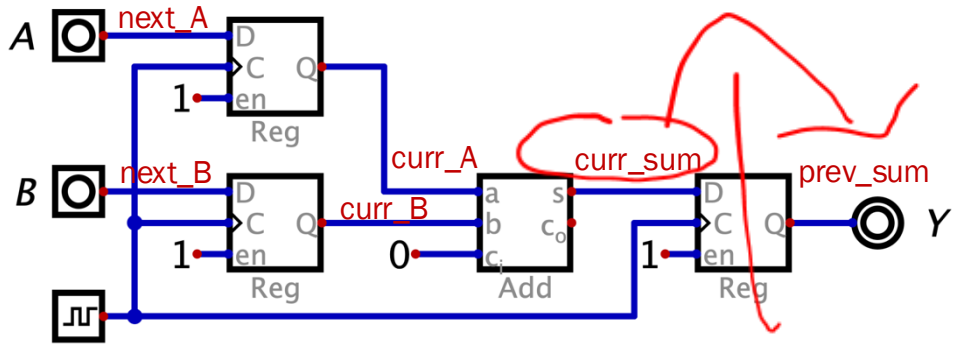


adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 700ps



Signal	Starting State	t = 0 A = 7 B = 7	t = 50ps A = 7 B = 7	t = 650ps A = 7 B = 7	t = 700 ps A = 18 B = 7
next_A	0	7	7	7	18
next_B	0	7	7	7	7
curr_A	0	0	7	7	7
curr_B	0	0	7	7	7
curr_sum	0	0	0	14	14
prev_sum	0	0	0	0	0

Cycle 1: 18 + 7

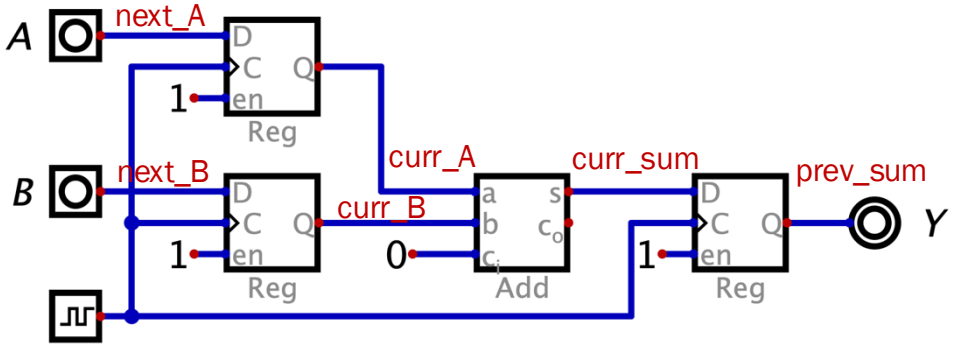


adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 700ps



Signal	t = 700 ps A = 18 B = 7	t = 750 ps ~~A = 18~~ B = 7	t = 1350 ps A = 18 B = 7	t = 1400 ps A = 30 B = 8
next_A	18			30
next_B	7			8
curr_A	7			
curr_B	7			
curr_sum	14			
prev_sum	0			

Cycle 1: 18 + 7

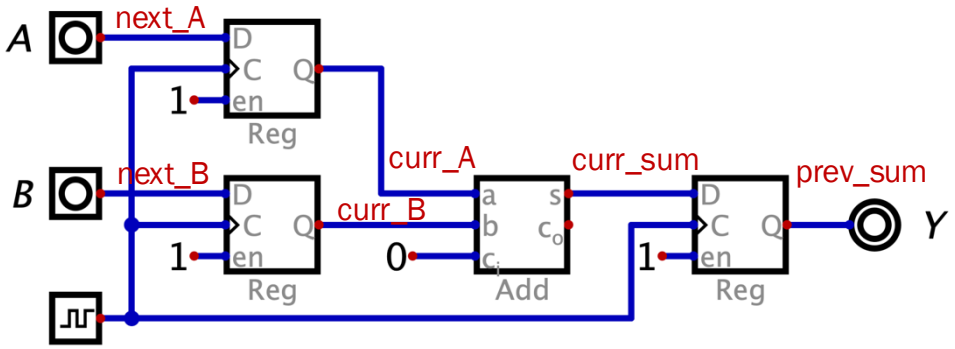


adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 700ps



Signal	t = 700 ps A = 18 B = 7	t = 750 ps A = 18 B = 7	t = 1350 ps A = 18 B = 7	t = 1400 ps A = 30 B = 8
next_A	18	18	18	30
next_B	7	7	7	8
curr_A	7	18	18	18
curr_B	7	7	7	7
curr_sum	14	14	25	25
prev_sum	0	14	14	14

Cycle 2: 30 + 8

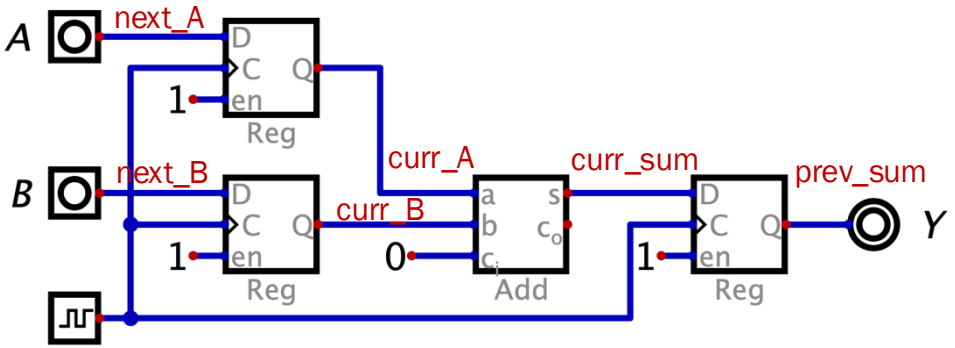


adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 700ps

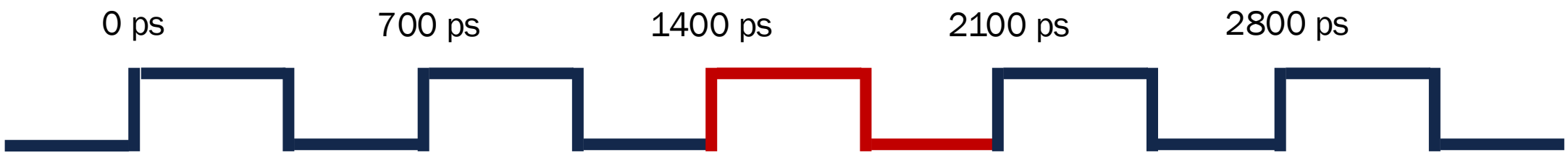


Signal	t = 1400 ps A = 30 B = 8	t = 1450 ps A = 30 B = 8	t = 2050 ps A = 30 B = 8	t = 2100 ps A = 15 B = 30
next_A	30			
next_B	8			
curr_A	18			
curr_B	7			
curr_sum	25			
prev_sum	14			

Cycle 2: 30 + 8

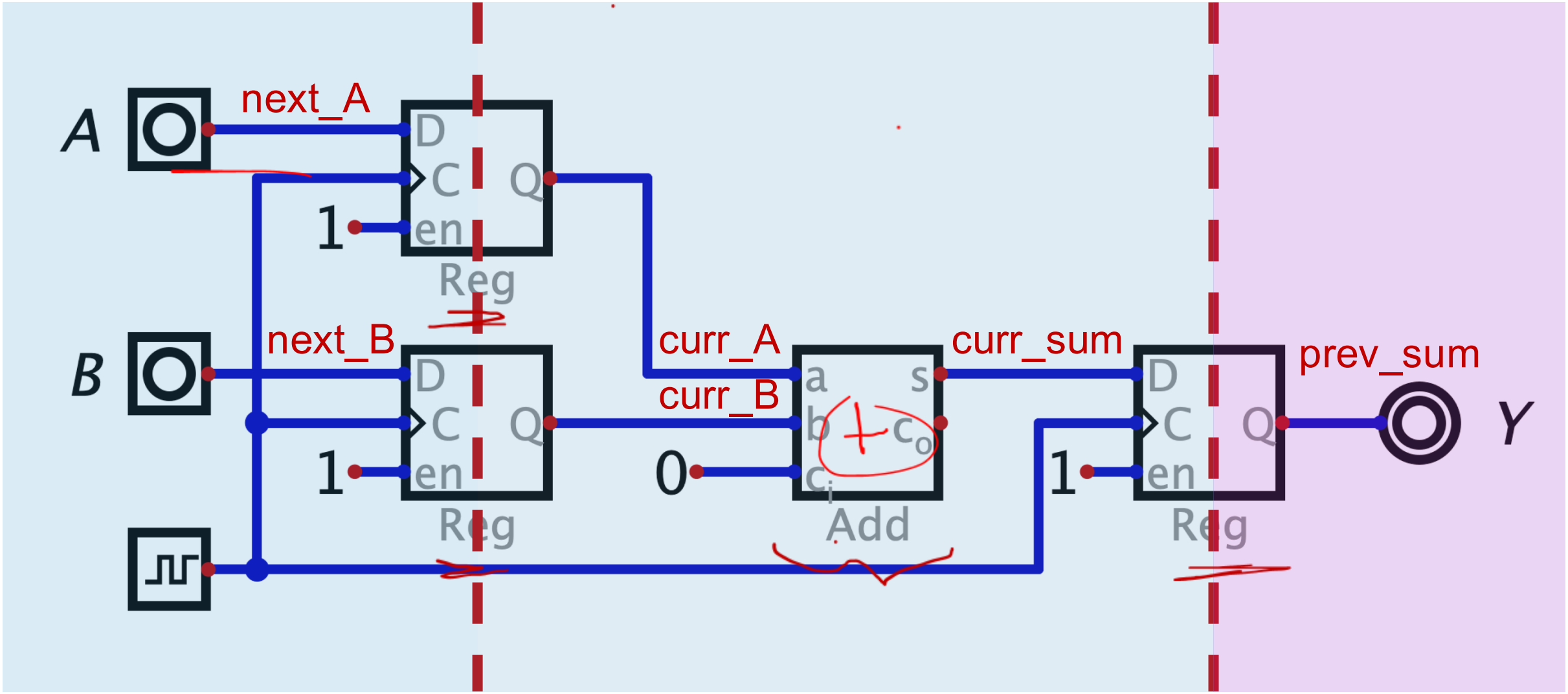


adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 700ps

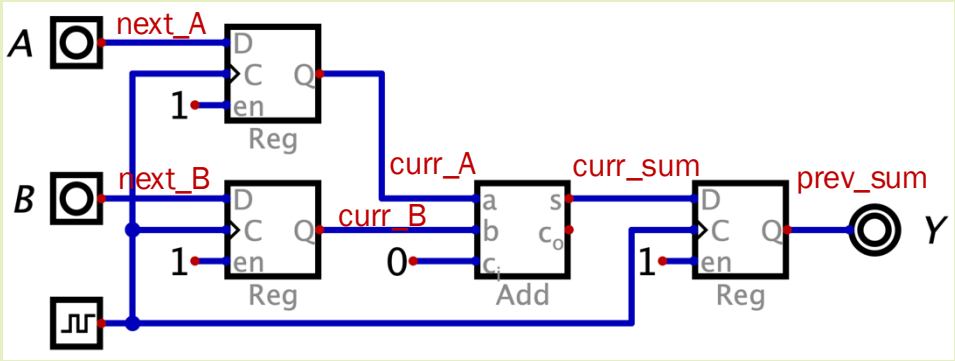


Signal	t = 1400 ps A = 30 B = 8	t = 1450 ps A = 30 B = 8	t = 2050 ps A = 30 B = 8	t = 2100 ps A = 15 B = 30
next_A	30	30	30	15
next_B	8	8	8	30
curr_A	18	30	30	30
curr_B	7	8	8	8
curr_sum	25	25	38	38
prev_sum	14	25	25	25

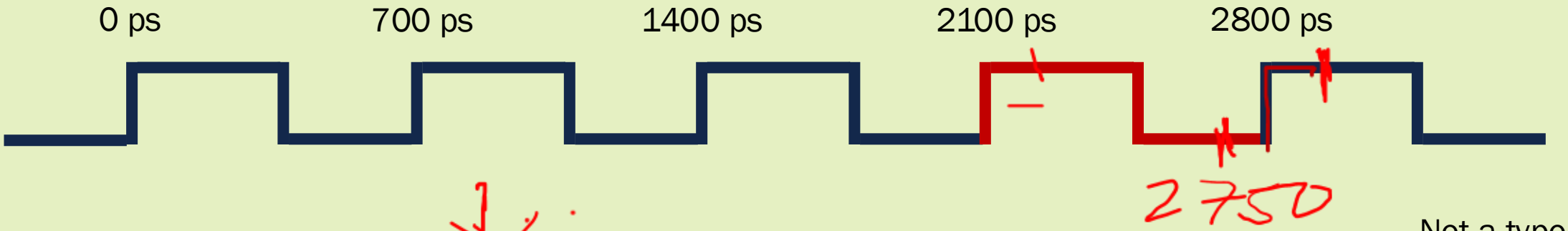
Ex1: Clock period = 700ps, clk-to-q delay = 50ps, adder propagation delay = 600ps



Cycle 3: 15 + 30



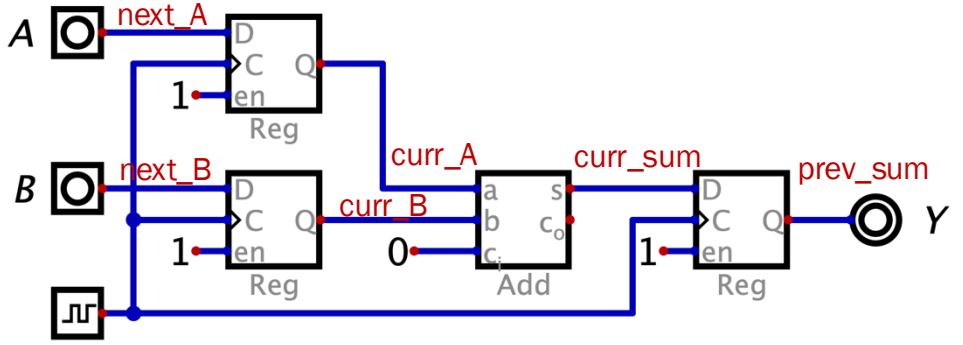
adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 700ps



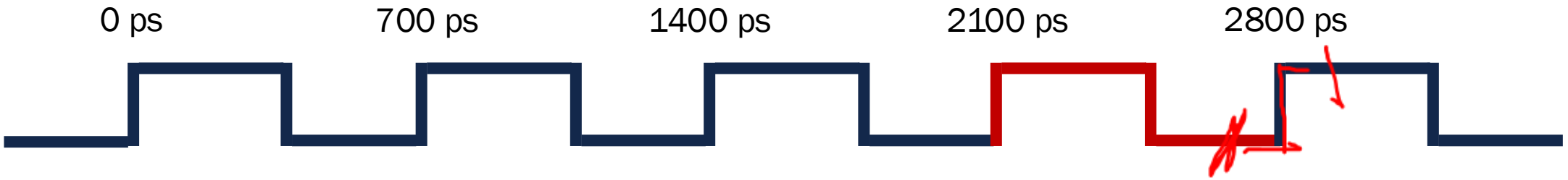
Signal	t = 2100 ps A = 15 B = 30	t = 2150 ps A = 15 B = 30	t = 2750 ps A = 20 B = 30	t = 2800 ps A = 20 B = 30
next_A	15		20	
next_B	30			
curr_A	30			
curr_B	8			
curr_sum	38			
prev_sum	25			

Not a typo! What will happen in this case?

Cycle 3: 15 + 30



adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 700ps



Signal	t = 2100 ps A = 15 B = 30	t = 2150 ps A = 15 B = 30	t = 2750 ps A = 20 B = 30	t = 2800 ps A = 20 B = 30
next_A	15	15	20	20
next_B	30	30	30	30
curr_A	30	15	15	15
curr_B	8	30	30	30
curr_sum	38	38	45	45
prev_sum	25	38	38	38

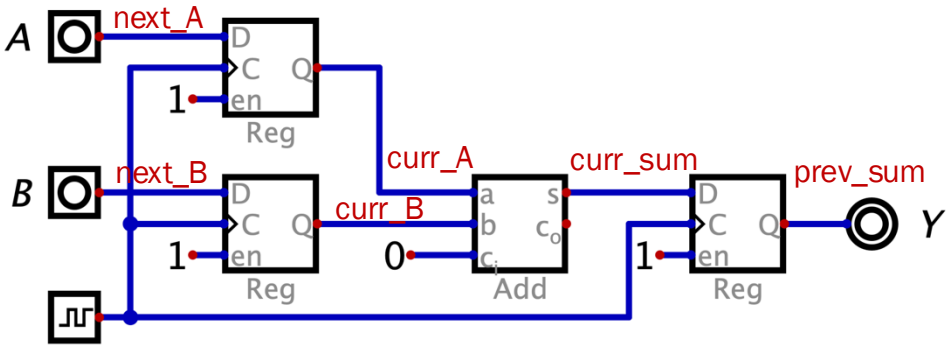
Not a typo! What will happen in this case?

Input A updating at t = 2750ps



- Our addition will work the same as before!
- The register prevents the value of A from moving forward until the rising edge of the clock
- We can update the inputs anytime from t = 2150ps to t = 2800ps
  - (aka until the next rising edge)
- The value will simply just stay at the input of the register until the rising edge of the clock

Cycle 4: 20 + 30

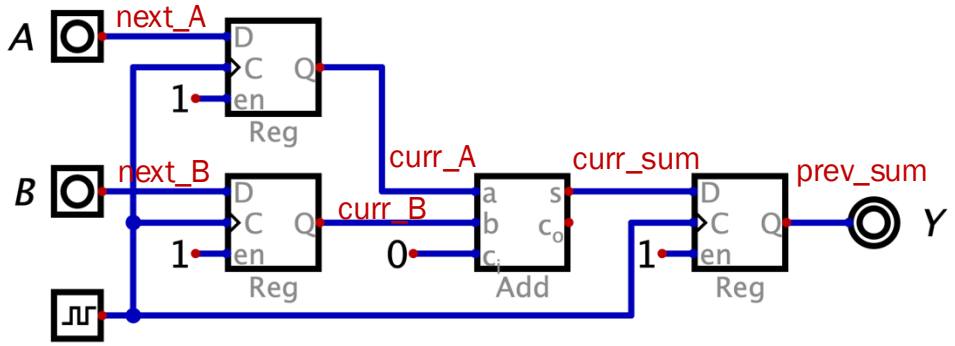


adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 700ps



Signal	t = 2800 ps A = 20 B = 30	t = 2850 ps A = 20 B = 30	t = 3450 ps A = 20 B = 30	t = 3500 ps A = 0 B = 0
next_A	20			
next_B	30			
curr_A	15			
curr_B	30			
curr_sum	45			
prev_sum	38			

Cycle 4: 20 + 30

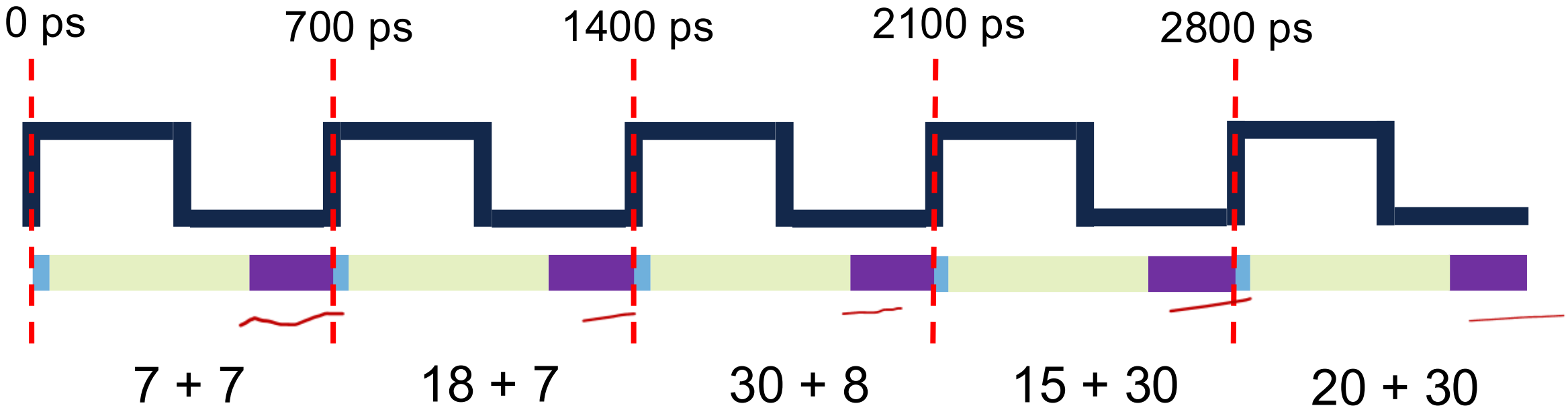


adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 700ps



Signal	t = 2800 ps A = 20 B = 30	t = 2850 ps A = 20 B = 30	t = 3450 ps A = 20 B = 30	t = 3500 ps A = 0 B = 0
next_A	20	20	20	0
next_B	30	30	30	0
curr_A	15	20	20	20
curr_B	30	30	30	30
curr_sum	45	45	50	50
prev_sum	38	45	45	45

Clock Period = 700ps

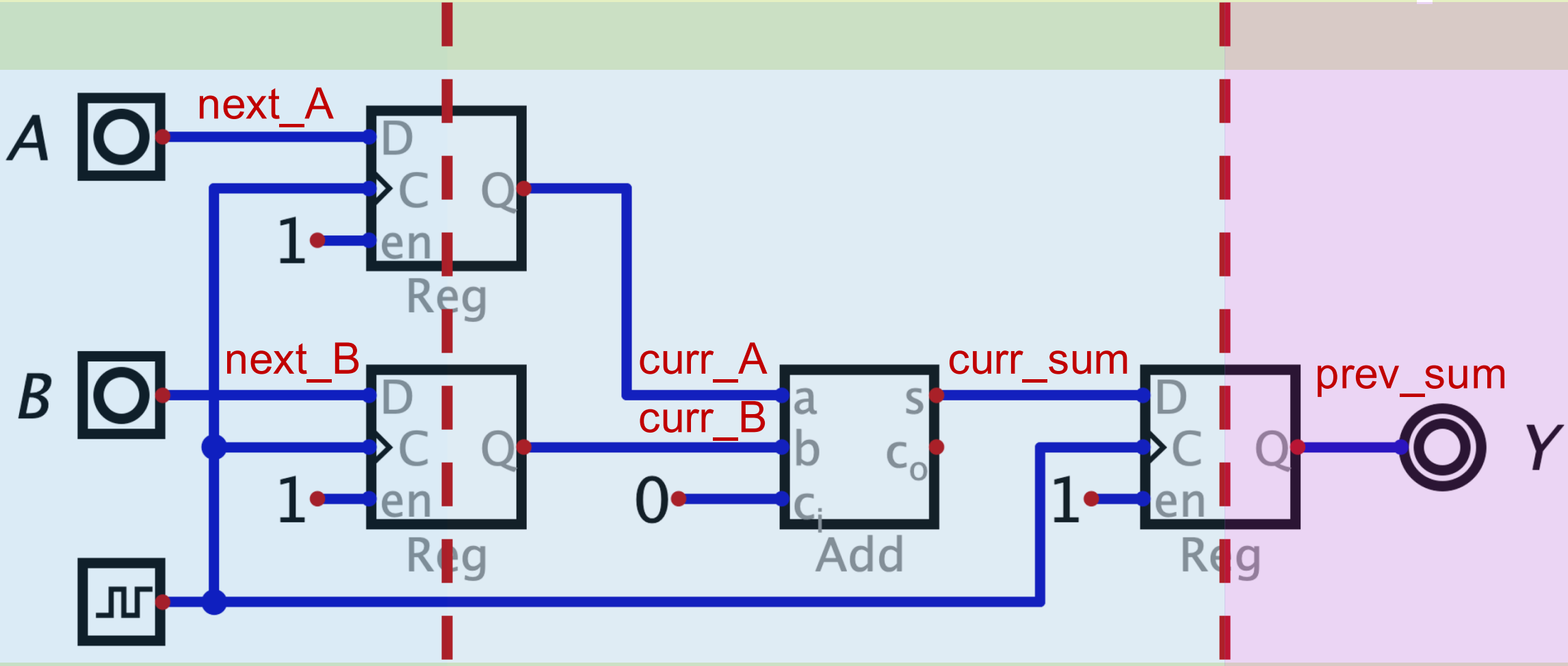


clk-to-q delay = 50ps

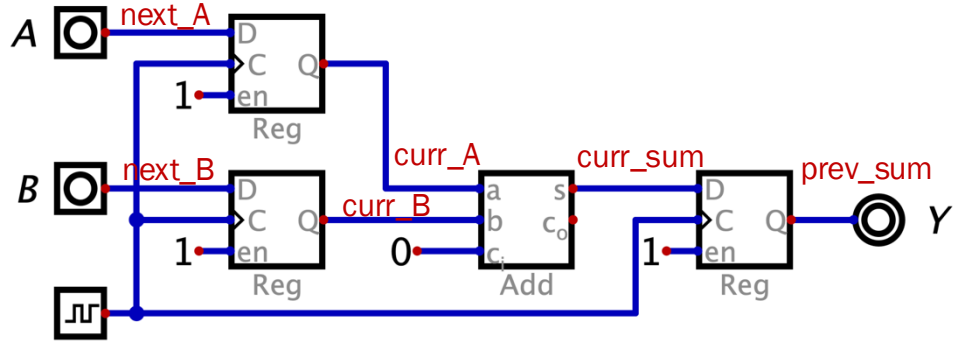
adder propagation delay = 600ps

idle time = 50ps

Ex2: Clock period = 650ps, clk-to-q delay = 50ps, adder propagation delay = 600ps. Fill out the whole table on the WS!



Cycle 0: 7 + 7

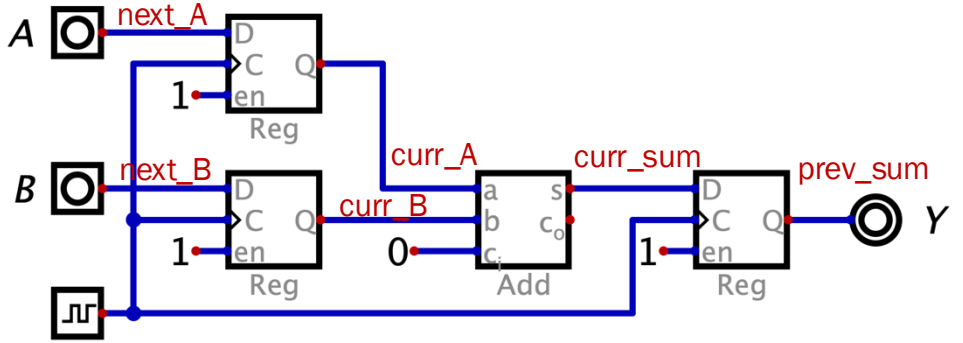


adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 650ps



Signal	Starting State	t = 0 A = 7 B = 7	t = 50ps A = 7 B = 7	t = 650 ps A = 18 B = 7
next_A	0			
next_B	0			
curr_A	0			
curr_B	0			
curr_sum	0			
prev_sum	0			

Cycle 0: 7 + 7

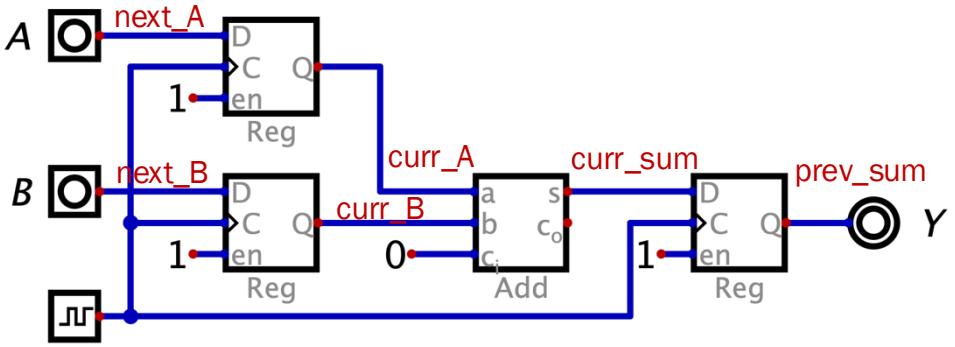


adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 650ps



Signal	Starting State	t = 0 A = 7 B = 7	t = 50ps A = 7 B = 7	t = 650 ps A = 18 B = 7
next_A	0	7	7	18
next_B	0	7	7	7
curr_A	0	0	7	7
curr_B	0	0	7	7
curr_sum	0	0	0	14
prev_sum	0	0	0	0

Cycle 1: 18 + 7

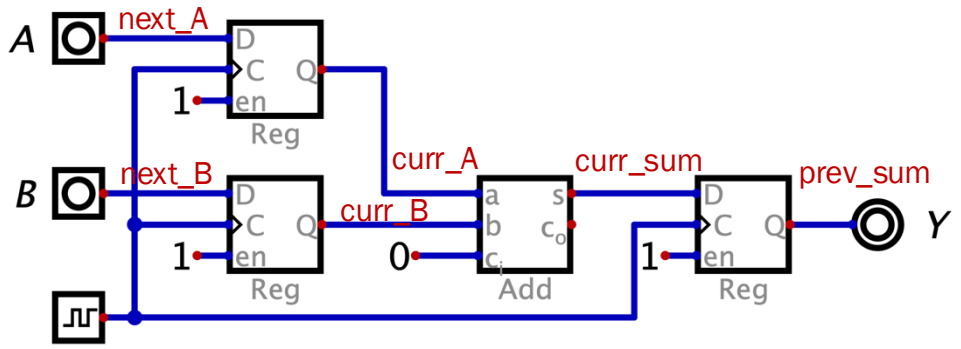


adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 650ps



Signal	t = 650 ps A = 18 B = 7	t = 700 ps A = 18 B = 7	t = 1300 ps A = 30 B = 8
next_A	18		
next_B	7		
curr_A	7		
curr_B	7		
curr_sum	14		
prev_sum	0		

Cycle 1: 18 + 7

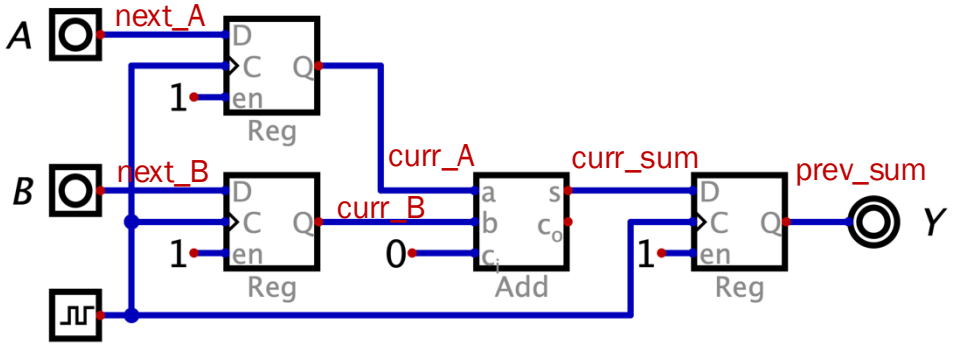


adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 650ps



Signal	t = 650 ps A = 18 B = 7	t = 700 ps A = 18 B = 7	t = 1300 ps A = 30 B = 8
next_A	18	18	30
next_B	7	7	8
curr_A	7	18	18
curr_B	7	7	7
curr_sum	14	14	25
prev_sum	0	14	14

Cycle 2: 30 + 8

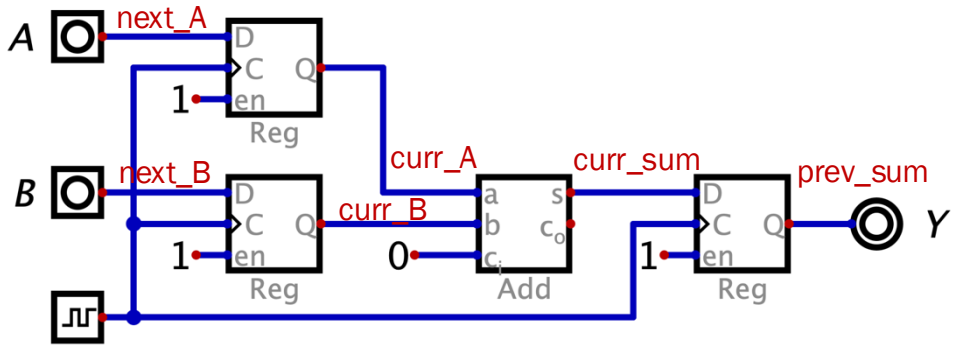


adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 650ps



Signal	t = 1300 ps A = 30 B = 8	t = 1350 ps A = 30 B = 8	t = 1950 ps A = 15 B = 30
next_A	30		
next_B	8		
curr_A	18		
curr_B	7		
curr_sum	25		
prev_sum	14		

Cycle 2: 30 + 8



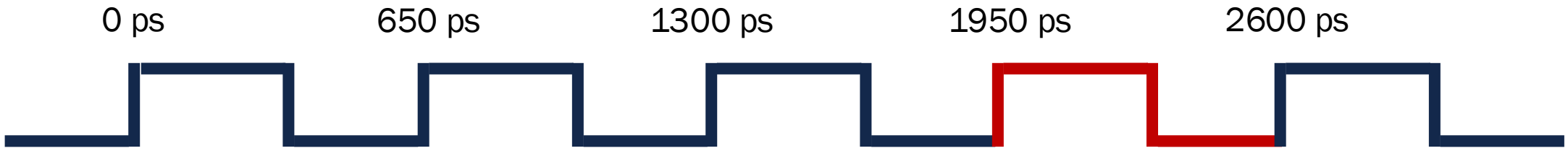
adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 650ps



Signal	t = 1300 ps A = 30 B = 8	t = 1350 ps A = 30 B = 8	t = 1950 ps A = 15 B = 30
next_A	30	30	15
next_B	8	8	30
curr_A	18	30	30
curr_B	7	8	8
curr_sum	25	25	38
prev_sum	14	25	25

adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 650ps

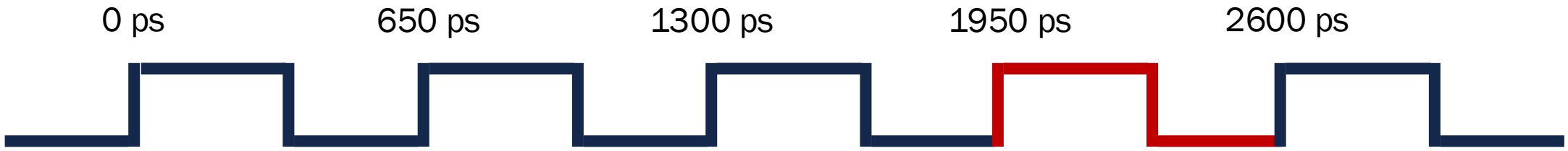
Cycle 3: 15 + 30



Signal	t = 1950 ps A = 15 B = 30	t = 2000 ps A = 15 B = 30	t = 2600 ps A = 20 B = 30
next_A	15		
next_B	30		
curr_A	30		
curr_B	8		
curr_sum	38		
prev_sum	25		

adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 650ps

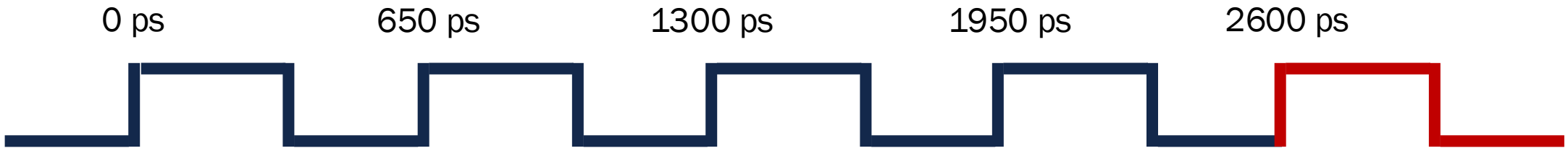
Cycle 3: 15 + 30



Signal	t = 1950 ps A = 15 B = 30	t = 2000 ps A = 15 B = 30	t = 2600 ps A = 20 B = 30
next_A	15	15	20
next_B	30	30	30
curr_A	30	15	15
curr_B	8	30	30
curr_sum	38	38	45
prev_sum	25	38	38

adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 650ps

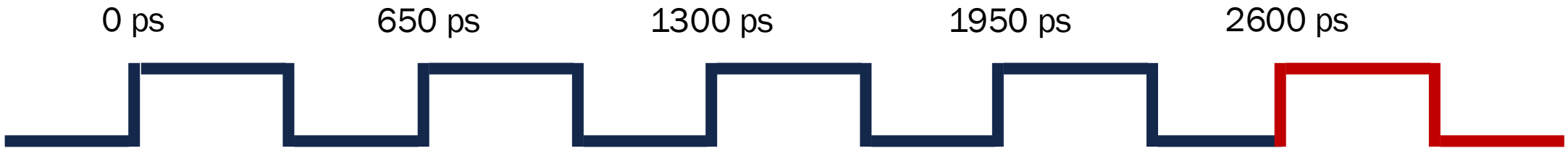
Cycle 4: 20 + 30



Signal	t = 2600 ps A = 20 B = 30	t = 2650 ps A = 20 B = 30	t = 3250 ps A = 0 B = 0	t = 3300 ps A = 0 B = 0
next_A	20			
next_B	30			
curr_A	15			
curr_B	30			
curr_sum	45			
prev_sum	38			

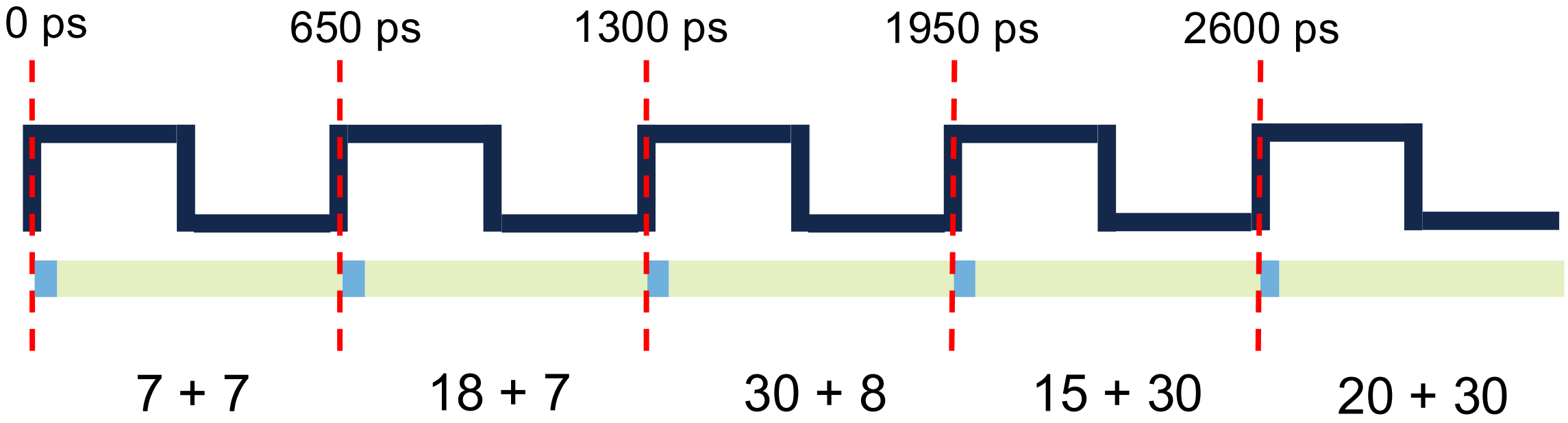
adder propagation delay = 600 ps
clk-to-q delay = 50ps
clock period = 650ps

Cycle 4: 20 + 30



Signal	t = 2600 ps A = 20 B = 30	t = 2650 ps A = 20 B = 30	t = 3250 ps A = 0 B = 0	t = 3300 ps A = 0 B = 0
next_A	20	20	0	0
next_B	30	30	0	0
curr_A	15	20	20	0
curr_B	30	30	30	0
curr_sum	45	45	50	50
prev_sum	38	45	45	50

Clock Period = 650ps



clk-to-q delay = 50ps

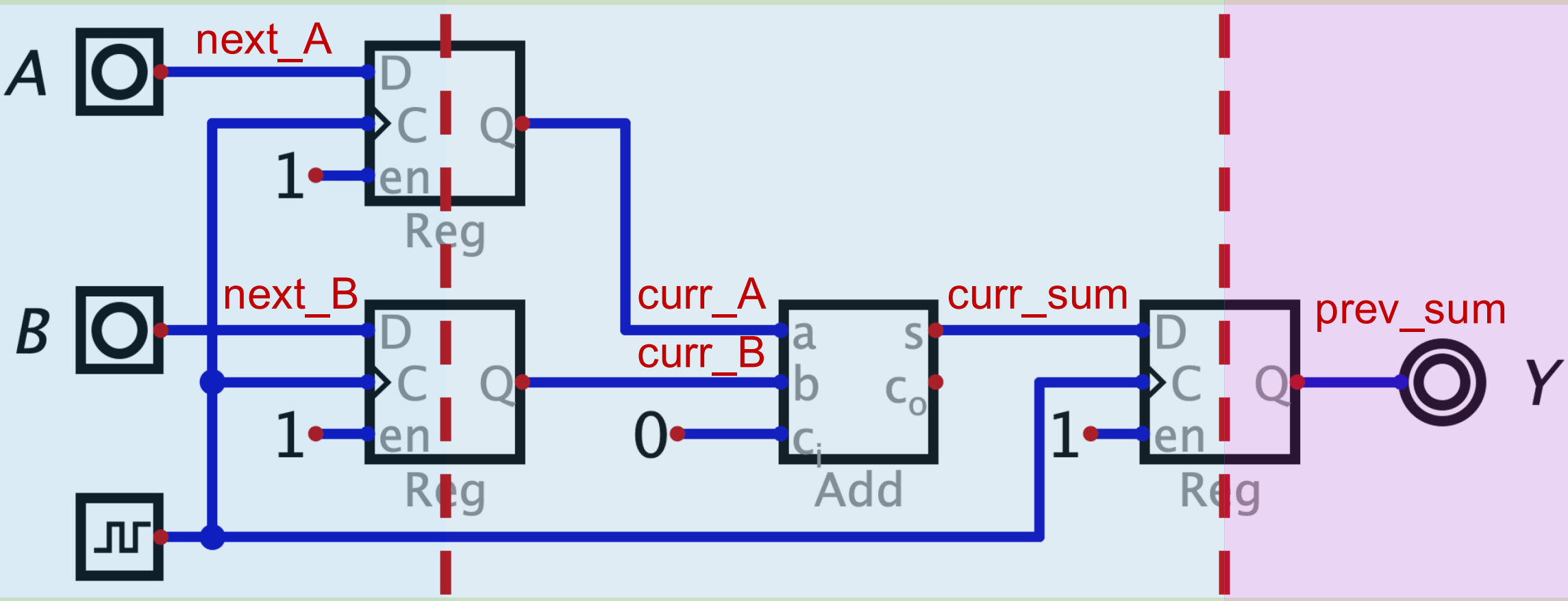
adder propagation delay = 600ps

Comparing Clock Periods

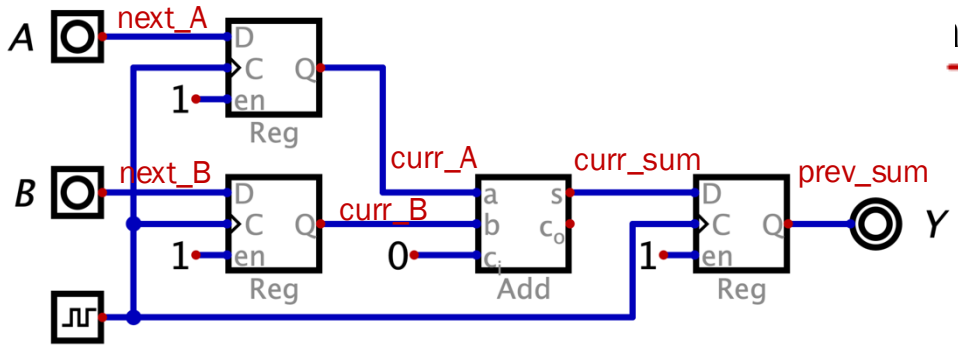


- Our circuit functioned correctly at both of the clock periods we tried
  - 650ps and 700ps
- Our five addition operations finished faster when we set our clock period to 650ps compared to 700ps

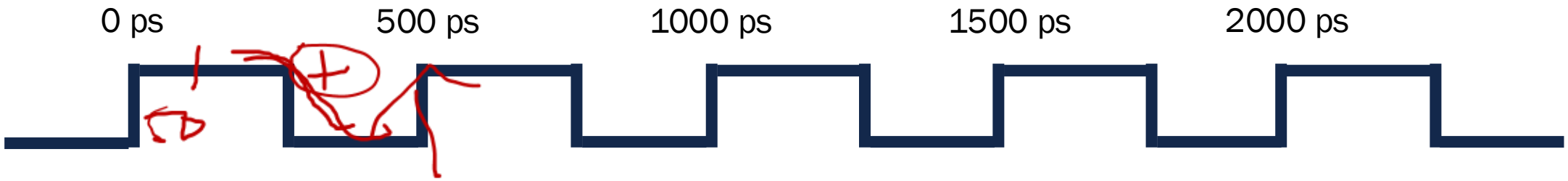
Ex3: Clock period = 500ps, clk-to-q delay = 50ps, adder propagation delay = 600ps. On WS, do cycle 0 and 1



Cycle 0 & 1: 7 + 7

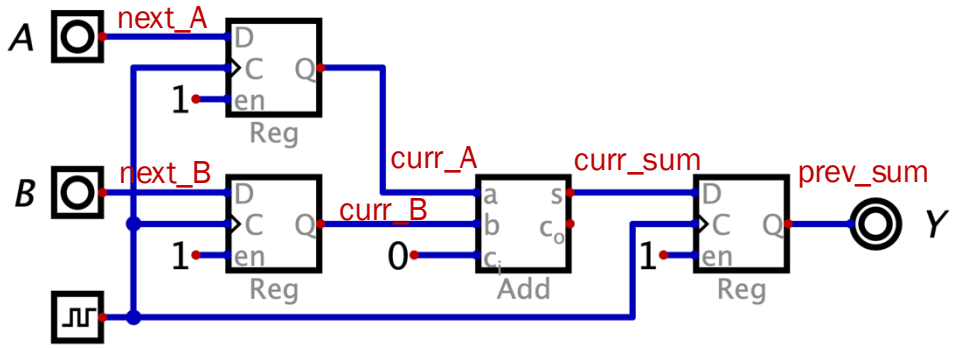


dder propagation delay = 600 ps
clk-to-q delay is 50ps
clock period = 500ps

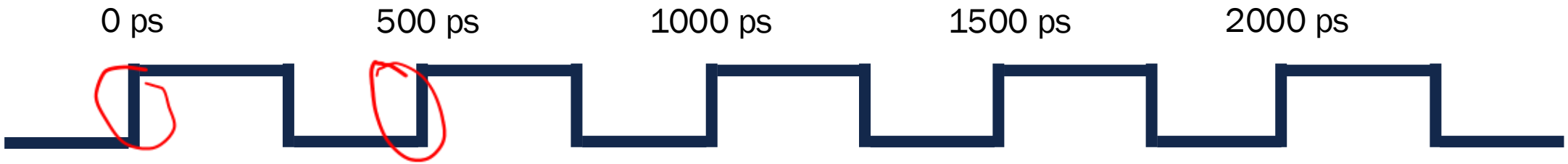


Signal	Starting State	t = 0 A = 7 B = 7	t = 50ps A = 7 B = 7	t = 500ps A = 18 B = 7	t = 550ps A = 18 B = 7
next_A	0	7	7	18	18
next_B	0	7	7	7	18
curr_A	0	0	7	7	18
curr_B	0	0	7	7	7
curr_sum	0	0	0	14	
prev_sum	0	0	0	0	

Cycle 0 & 1: 7 + 7

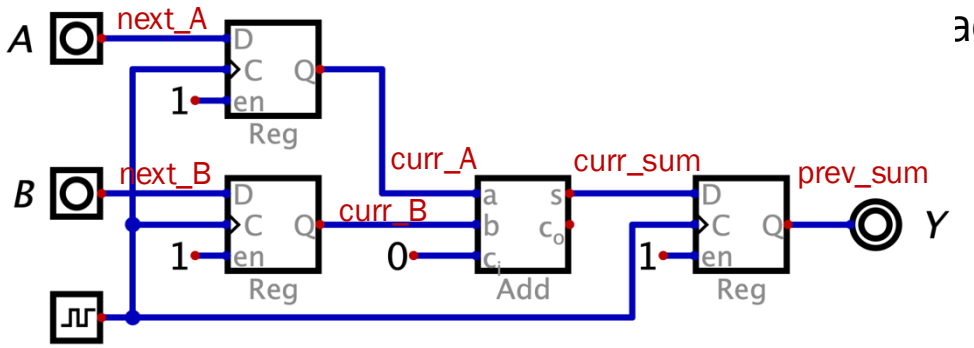


ıdder propagation delay = 600 ps
clk-to-q delay is 50ps
clock period = 500ps



Signal	Starting State	t = 0 A = 7 B = 7	t = 50ps A = 7 B = 7	t = 500ps A = 18 B = 7	t = 550ps A = 18 B = 7
next_A	0	7	7	18	
next_B	0	7	7	7	
curr_A	0	0	7		
curr_B	0		7		
curr_sum	0		0	? ?	
prev_sum	0				

Cycle 0 & 1: 7 + 7

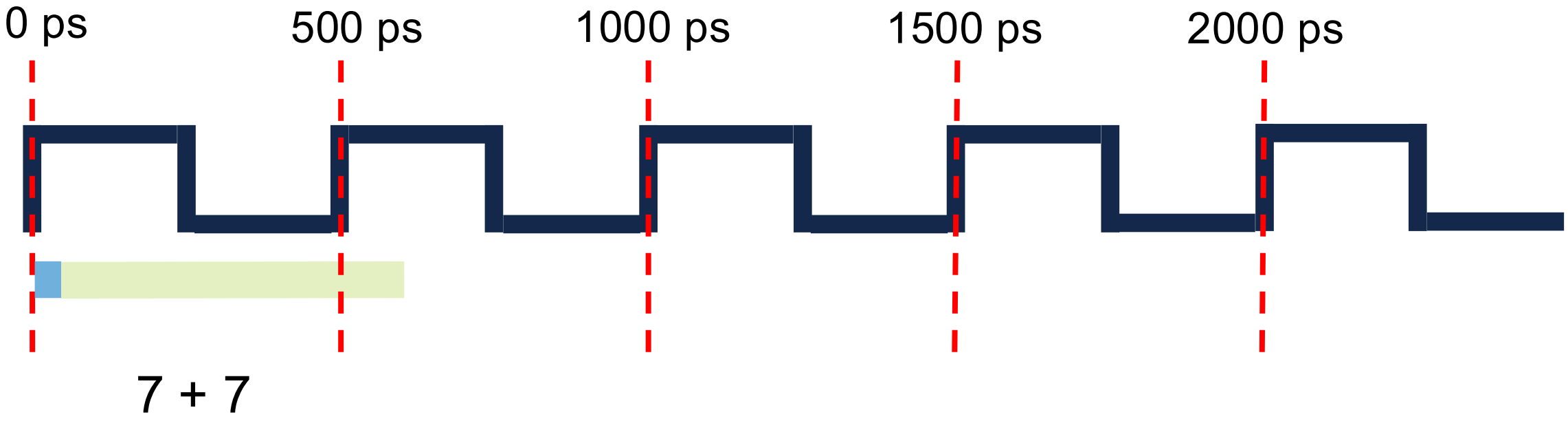


adder propagation delay = 600 ps
clk-to-q delay is 50ps
clock period = 500ps



Signal	Starting State	t = 0 A = 7 B = 7	t = 50ps A = 7 B = 7	t = 500ps A = 18 B = 7	t = 550ps A = 18 B = 7
next_A	0	7	7	18	18
next_B	0	7	7	7	7
curr_A	0	0	7	7	18
curr_B	0	0	7	7	7
curr_sum	0	0	0	??	??
prev_sum	0	0	0	0	??

Clock Period = 500ps



clk-to-q delay = 50ps

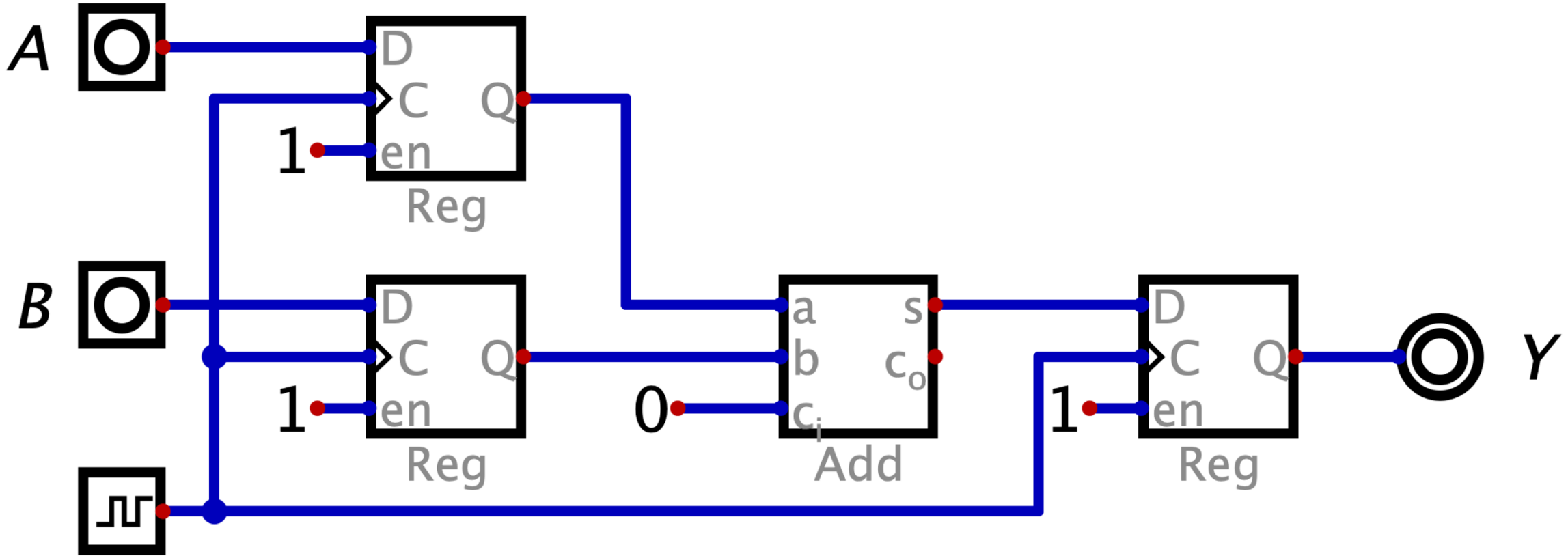
adder propagation delay = 600ps

Clock Period = 500ps



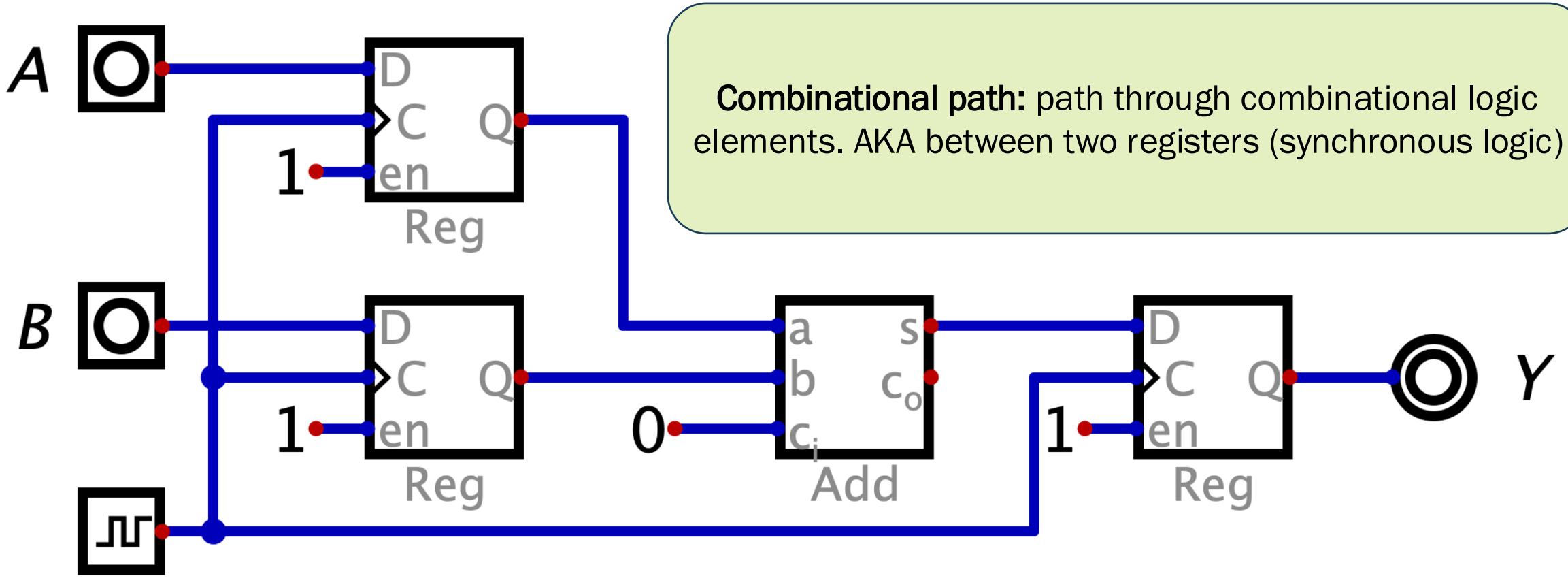
- Our circuit does not operate correctly with a period of 500ps
- Why?
  - The adder cannot finish its operation before the next rising edge of the clock
  - The output register ends up reading garbage

Take Away: How to determine the minimum clock period?



min clock period = clk-to-q delay of the input registers + adder propagation delay

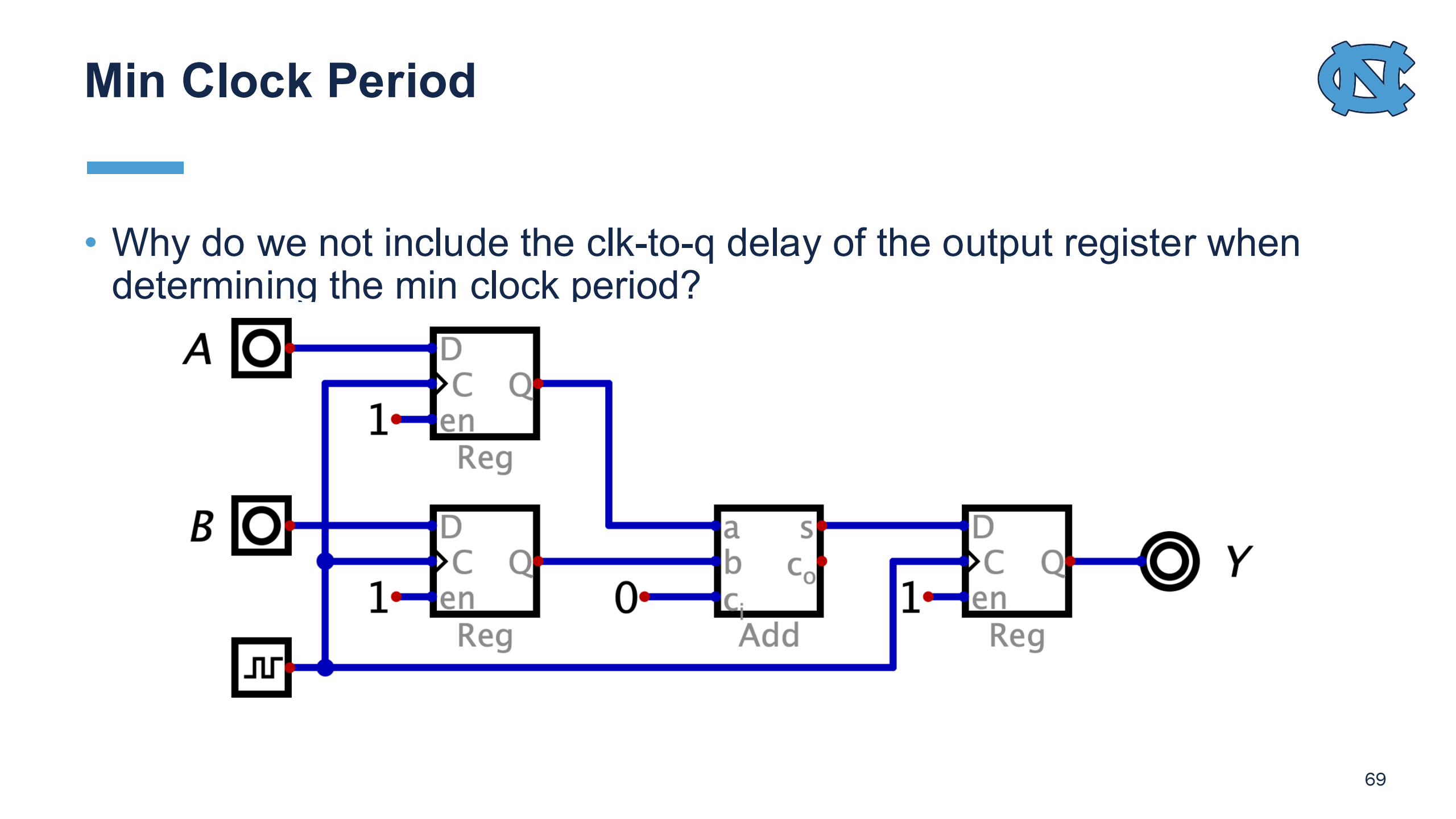
In General…



Combinational path: path through combinational logic elements. AKA between two registers (synchronous logic)

min clock period = clk-to-q delay of the input registers + ***longest combinational path***

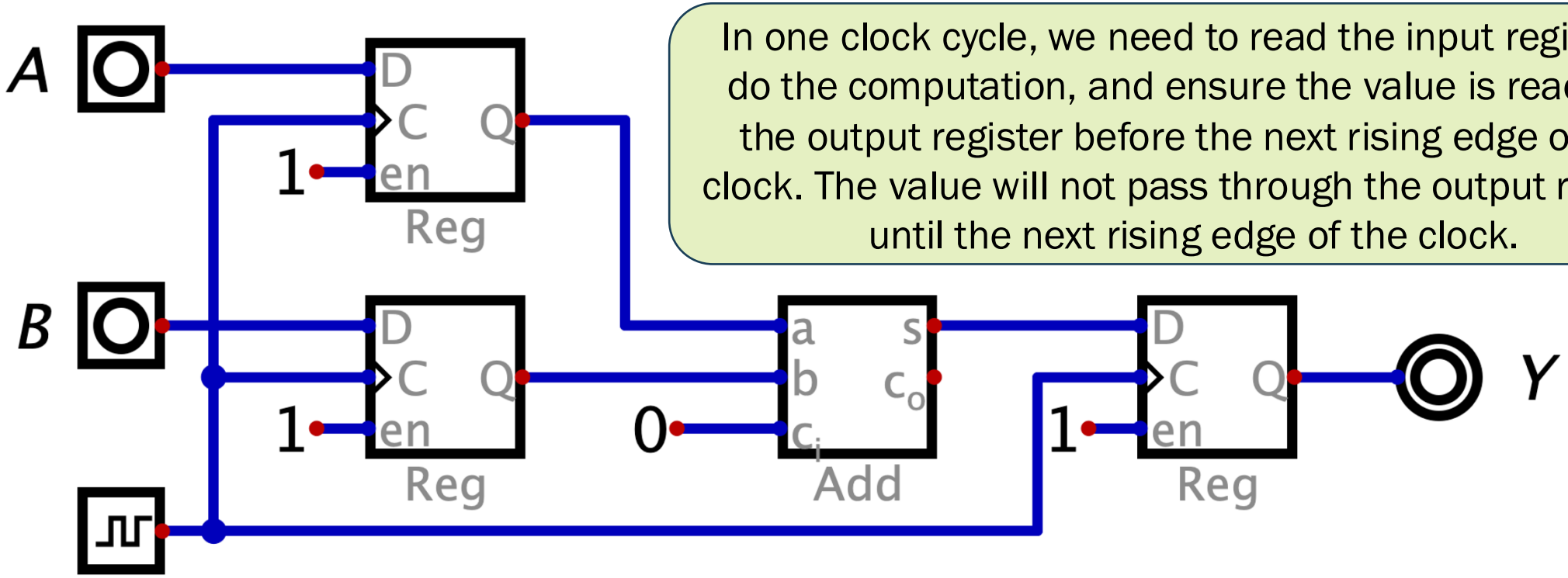
Min Clock Period



- Why do we not include the clk-to-q delay of the output register when determining the min clock period?

Min Clock Period

- Why do we not include the clk-to-q delay of the output register when determining the min clock period?



In one clock cycle, we need to read the input registers, do the computation, and ensure the value is ready for the output register before the next rising edge of the clock. The value will not pass through the output register until the next rising edge of the clock.