Unit 11: Sequential Circuits

CSE 220: System Fundamentals I

Stony Brook University

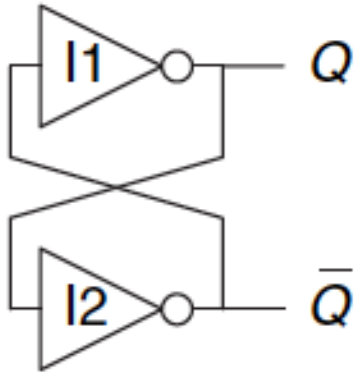
Joydeep Mitra

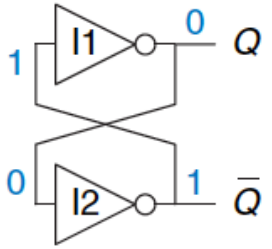
Combination Vs Sequential

- In Combinational logic, outputs depend on current inputs
- In Sequential logic, outputs depend on current and prior inputs
- So, sequential logic has *memory*
- Sequential logic circuits remembers prior inputs in a set of bits called *state variables;* the set itself is called the *state*
- Outputs are a function of current inputs and state
- Sequential logic circuits are based on *bistable elements*

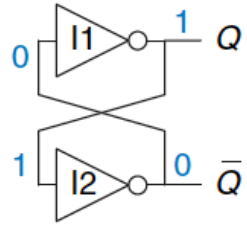
Bistable Element

- The fundamental building block of memory is the *bistable element*
  - Bistable because it has two *stable* states
- The element is designed as two inverters with two outputs, which are cross-coupled as inputs
- There are two cases to consider
  - **Q = 0**; input to I2 is 0 and output is Q', which is 1. Q' is input to I1, which outputs Q, which is 0. Our assumption Q = 0 is correct! Hence, *stable*.
  - **Q = 1**; input to I2 is 1 and output is Q', which is 0. Q' is input to I1, which outputs Q, which is 1. Our assumption Q = 1 is correct! Hence, *stable*.





Q = 0



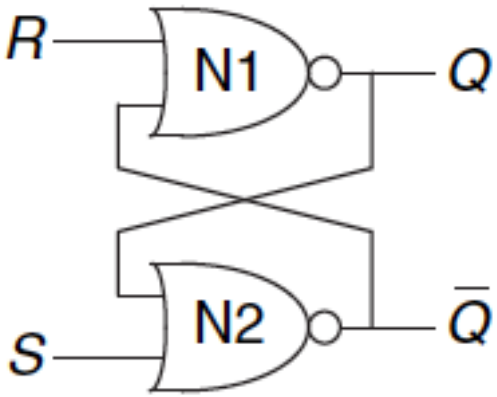
Q = 1

Bistable Element

- A bistable element stores 1 bit. Why?
  - N stable states convey $\log_2 N$ bits of information
  - Bistable element has 2 stable states
- If Q = 0, then it will remain 0 forever
- If Q = 1, then it will remain 1 forever
- This notion of stability underlies the property of memory
- However, the initial state of a bistable element is unknown, differs each time circuit is turned on
- Hence, they cannot be practically used as the user cannot control the inputs to control state
- Consequently, we need bistable elements with inputs (latches and flip-flops)

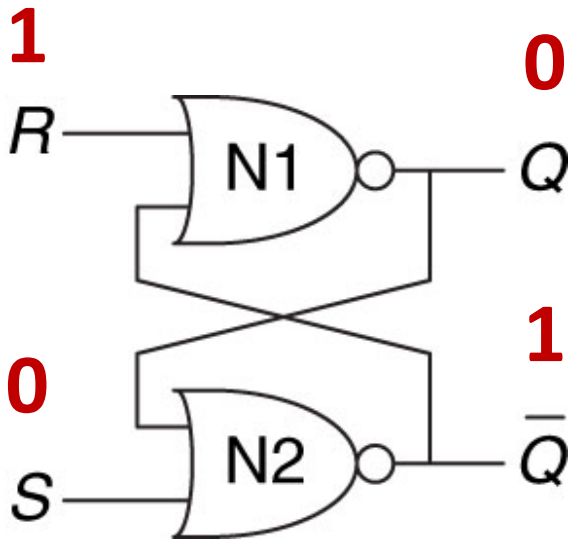
SR Latch

- The SR latch is composed of two cross-coupled NOR gates
- It has two inputs, S and R, and two outputs, Q and Q'
- Let's understand the circuit by exploring each input combination

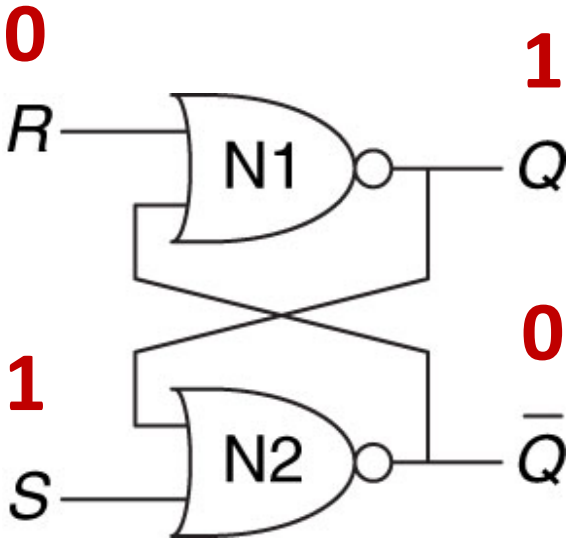


The SR Latch

- Case 1: $S = 0, R = 1$
- Then, Q = 0, Q' = 1

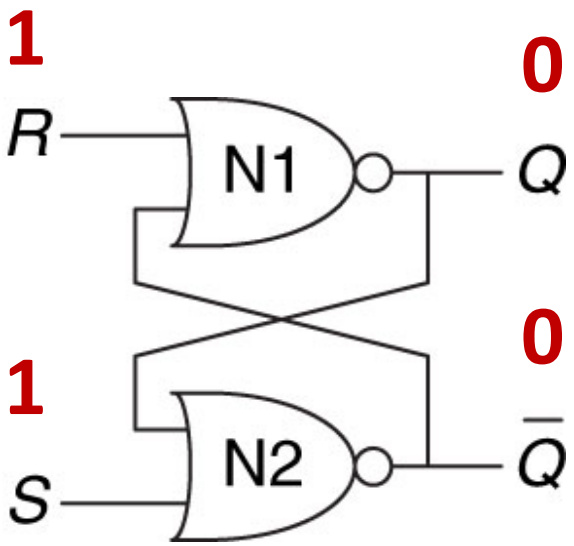


- Case 2: $S = 1, R = 0$
- Then, Q = 1, Q' = 0



The SR Latch

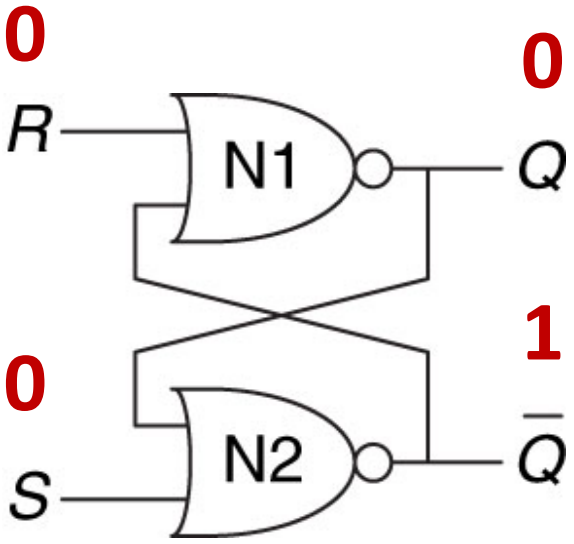
- Case 3: $S = 1, R = 1$
- Then, $Q = 0, Q' = 0$



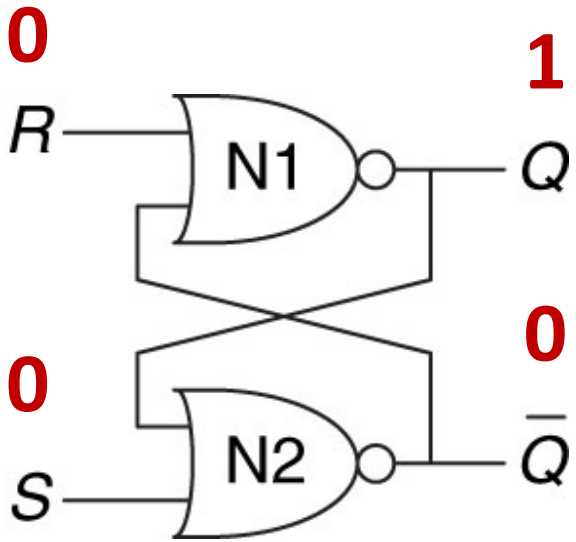
- So, when $S = R = 1$, both outputs are 0 (nonsense inputs!)

The SR Latch

- Case 4A: $S = 0, R = 0, Q = 0$
- Then, $Q = 0, Q' = 1$



- Case 4B: $S = 0, R = 0, Q = 1$
- Then, $Q = 1, Q' = 0$



Putting It Together

- Suppose Q has a known prior value (Q_{prev})
- When S = 0 and R = 0, circuit remembers Q_{prev}
- When S = 1, circuit "sets" Q to 1
- When R = 1, circuit "resets" Q to 0
- *The circuit has memory!*

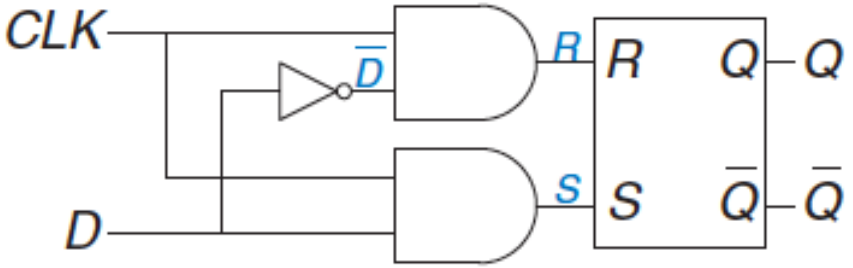
S	R	Q	$\overline{Q}$
0	0	Q_{prev}	$\overline{Q}_{prev}$
0	1	0	1
1	0	1	0
1	1	0	0

Limitation Of SR Latch

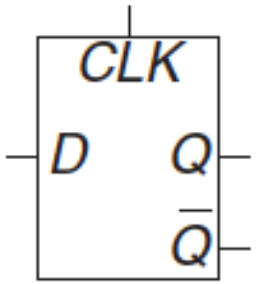
- The S and the R inputs conflate the *what* and the *when*
- Specifically, asserting one of the inputs not only determines Q but also when Q should change!
- This is why we get nonsense outputs when both S = 1 and R = 1
- This is not a good design principle; the *what* and the *when* need to be separated
- **D latch** to the rescue! It has two inputs
  - A *data* input, D, controls what the next state should be
  - A *clock* input, CLK, controls when the state should change

D Latch

- When CLK = 0, S = 0 and R = 0 (due to AND gate)
  - Hence, previous Q will be retained
- When CLK = 1, S = D and R = D’
  - So, if D is set to 1, Q is set to 1
  - If D is reset to 0, Q is reset to 0



Circuit



Symbol

Putting It Together

- The Clock controls when data flows through the latch
- When CLK = 1, latch is *transparent*, i.e., allows new data
- When CLK = 0, latch is *opaque,* i.e., blocks new data and retains old data
- A D latch is also called a *level-sensitive* latch

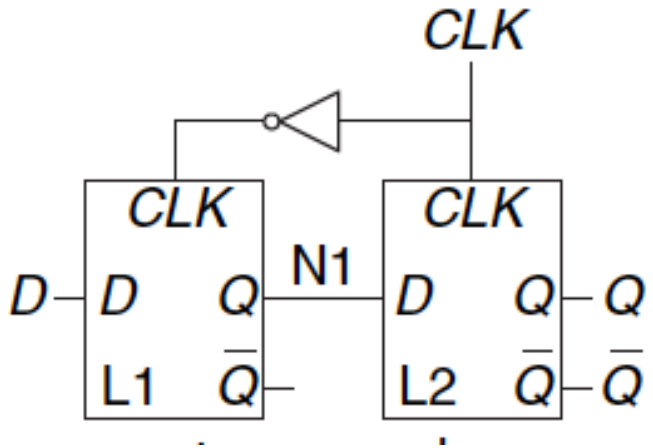
CLK	D	$\overline{D}$	S	R	Q	$\overline{Q}$
0	X	$\overline{X}$	0	0	Q_{prev}	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

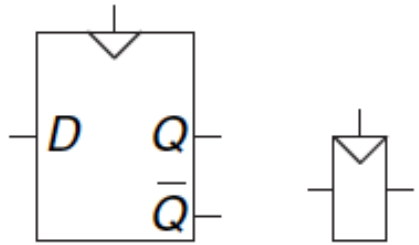
Limitation of D Latch

- Notice when CLK = 1, Q changes state
- If CLK stays at 1, then Q will keep changing, making the circuit unstable
- The **D flip-flop** addresses this by changing state at a particular *instant* in time

D Flip-Flop

- A D flip-flop can be built from two back-to-back D latches, controlled by complementary clocks
- When CLK = 0, latch L1 is transparent and L2 is opaque
  - The value at D propagates to Q via N1
- When CLK = 1, latch L1 is opaque and L2 is transparent
  - The value at D is blocked from reaching Q
- When clock rises from 0 to 1, value at D is copied to Q
- At all other times, Q retains its old value
- A D flip-flop is also called an *edge-triggered* or a *positive edge-triggered* flip-flop

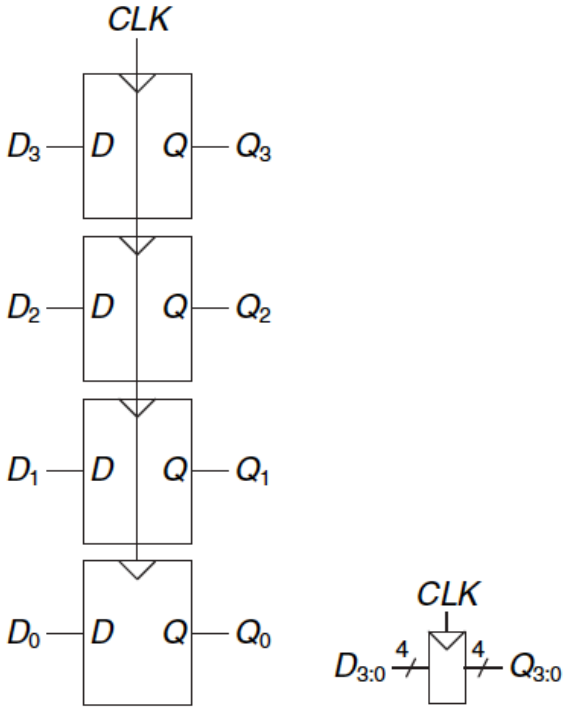




D flip-flop symbols

Register

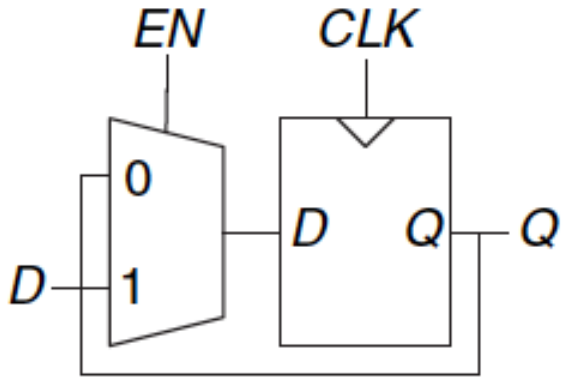
- An N-bit register is a bank of N flip-flops with a common CLK to update all N bits at the same time



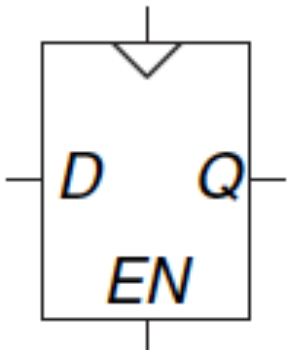
4-bit register schematic and symbol

Enable Flip-Flop

- Recall the D flip-flop changes state when clock edge rises from 0 to 1
- But we may not want to update state at every rising clock edge
- An *Enabled flip-flop* allows us to update state at every alternate rising edge of the clock
- To this end, it add a 2:1 MUX to the D flip-flop
  - with **EN** as select signal
  - **D** and $\mathbf{Q_{prev}}$ as inputs signals
- When **EN = 0**, the clock is ignored old value is retained
- When **EN = 1**, behaves like D flip-flop



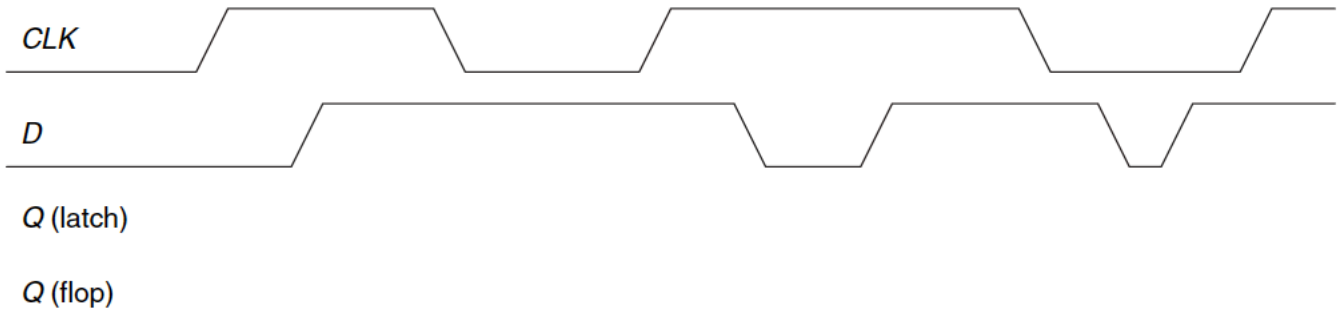
Schematic



Symbol

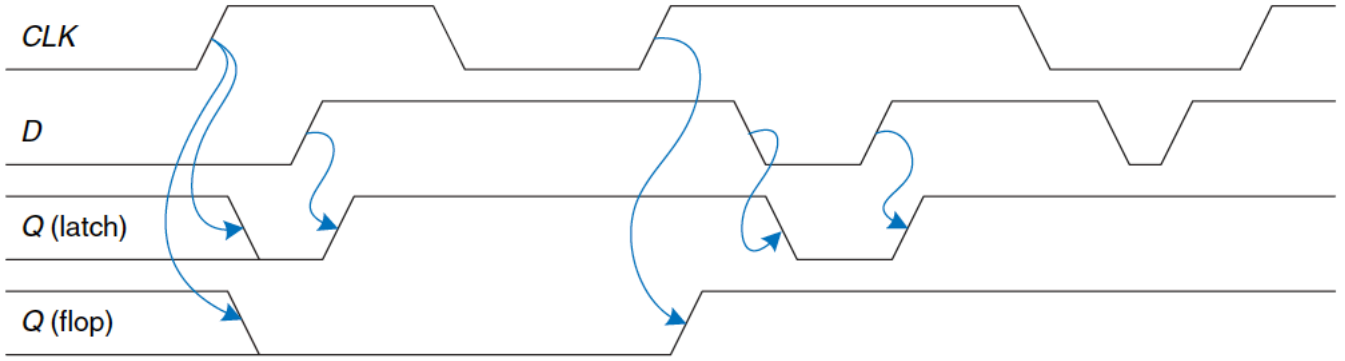
Timing of Sequential Circuits

- Recall that a digital clock is represented as a timing waveform with rising edges indicating the transition from 0 to 1
- To appreciate the differences between latches and flip-flops, consider the timing waveform below
- Using the waveform, we will determine the output, Q, for each device
  - We assume a slight delay for Q to respond to input changes



Timing of Sequential Circuits

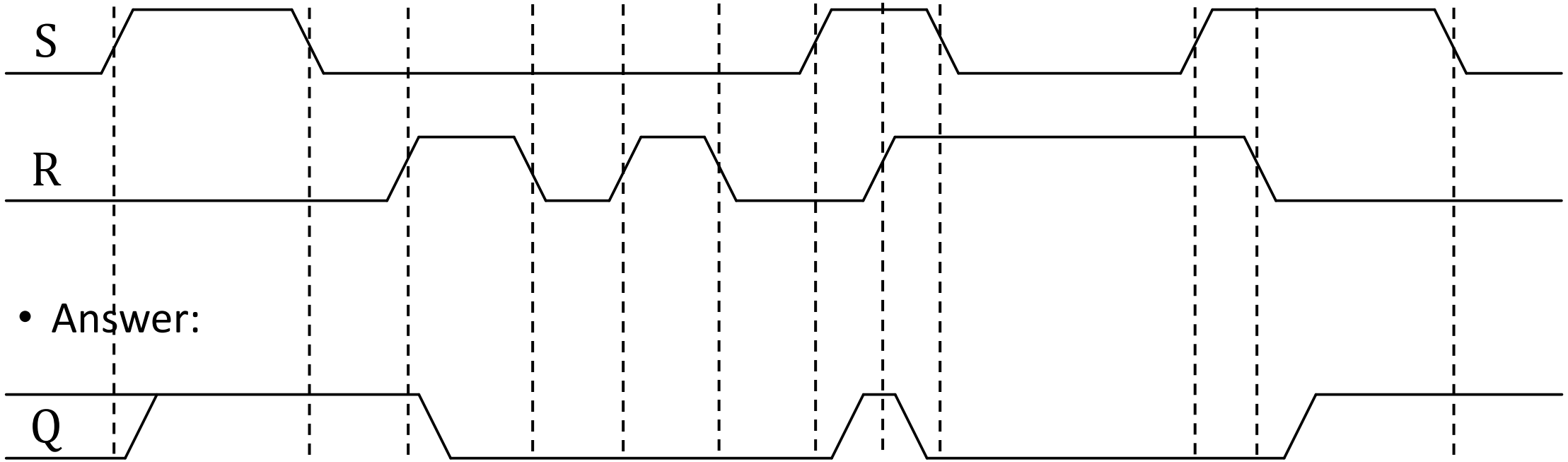
- Initial value of Q is 0 or 1 (indicated by the horizontal line pair)
- For the latch
  - on the rising edge of CLK, D = 0, so Q = 0
  - Subsequently, Q follows D whenever CLK = 1, otherwise Q is retained
- For the flip-flop,
  - On the rising edge of CLK, D is copied to Q
  - At all other times, Q is retained



Timing Waveforms: Example #1

S	R	Q	$\bar{Q}$
0	0	Q_{prev}	$\bar{Q}_{prev}$
0	1	0	1
1	0	1	0
1	1	0	0

- Consider the input waveform below of an SR latch. Sketch the output (Q) of the latch.

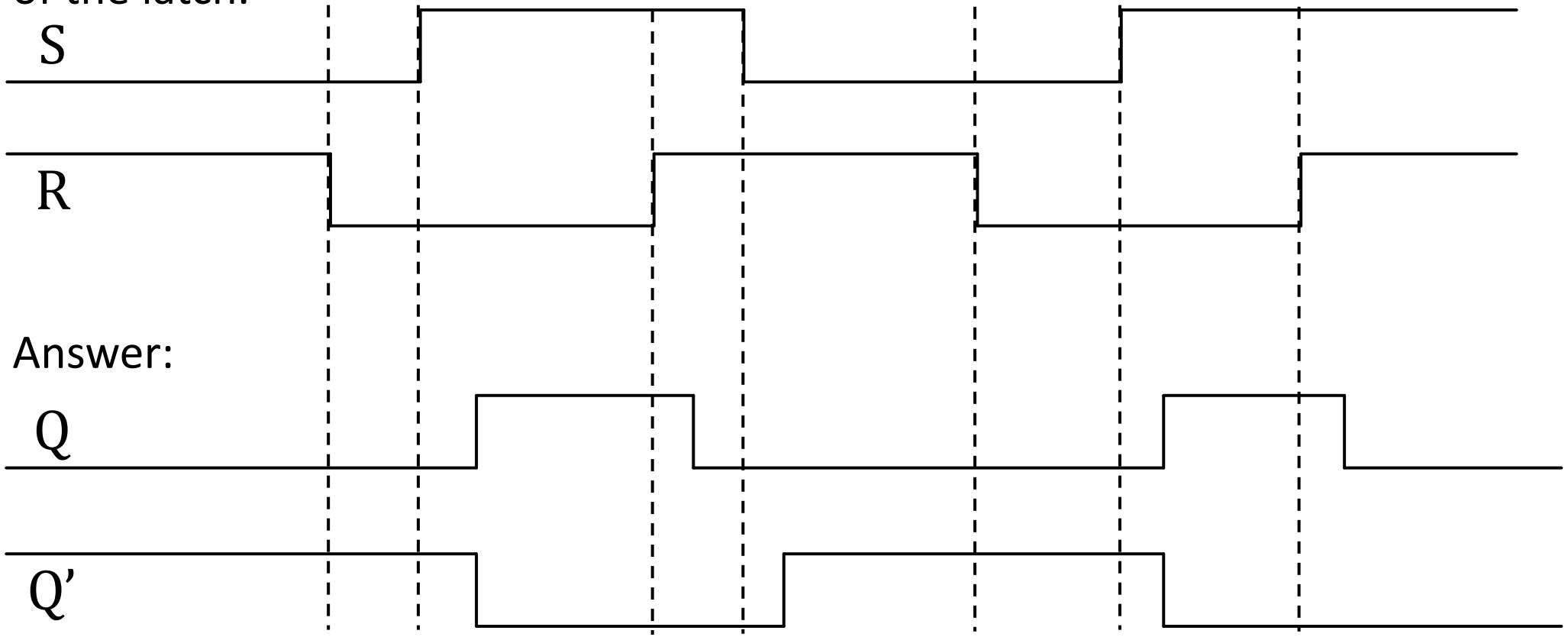


- Answer:

Timing Waveforms: Example #2

S	R	Q	$\bar{Q}$
0	0	Q_{prev}	$\bar{Q}_{prev}$
0	1	0	1
1	0	1	0
1	1	0	0

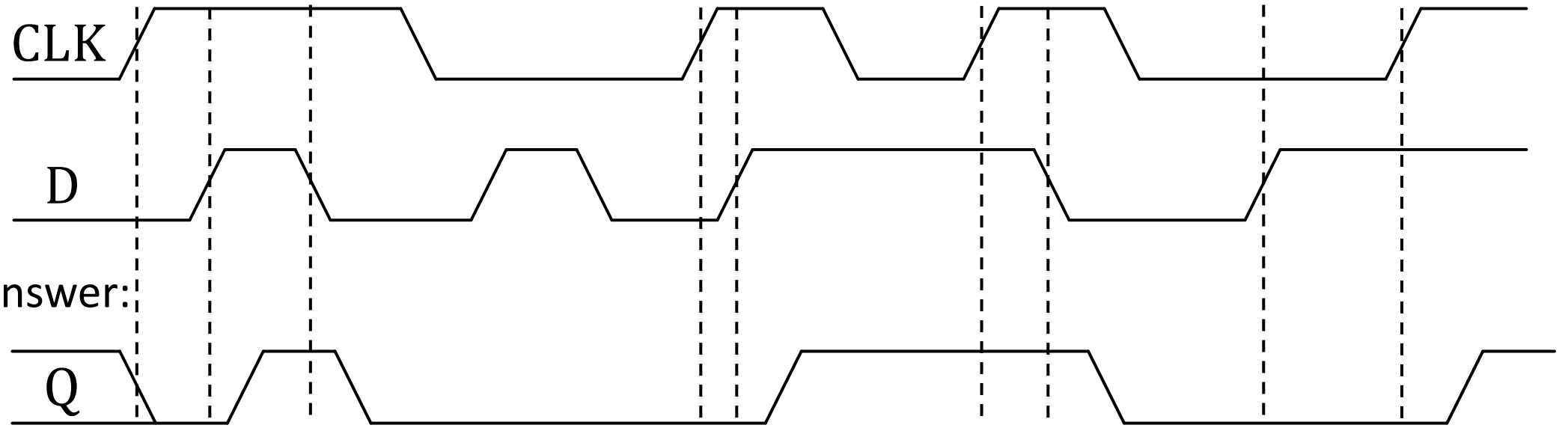
- Consider the input waveform below of an SR latch. Sketch the outputs (Q and Q') of the latch.



- Answer:

Timing Waveforms: Example #3

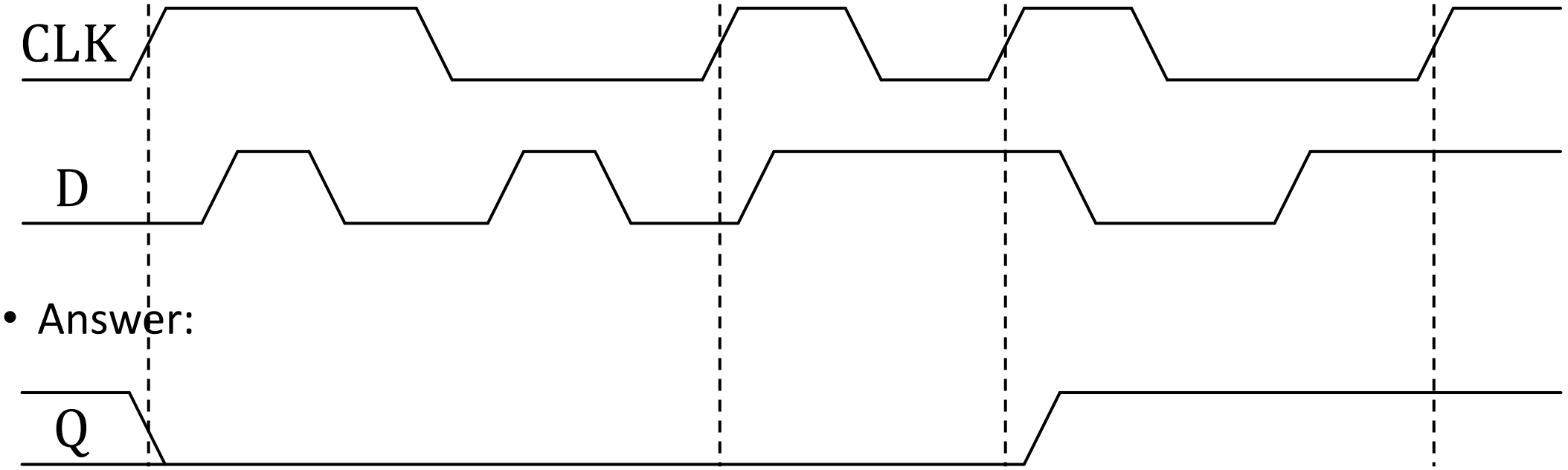
- Consider the input waveform below of a D latch. Sketch the output (Q) of the latch.



- Answer:

Timing Waveforms: Example #4

- Consider the input waveform below of a D flip-flop. Sketch the output (Q) of the flip-flop.



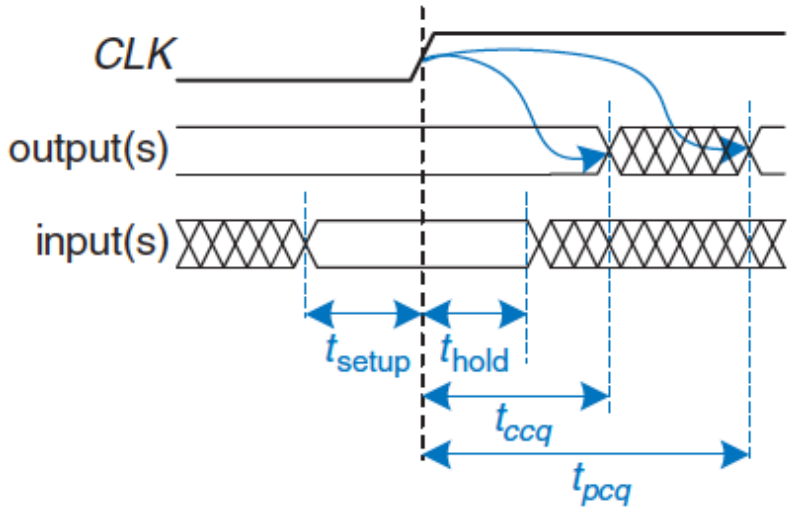
- Answer:

Timing Of Sequential Logic

- Recall that a flip-flop copies input D to output Q on the rising edge of the clock.
- What happens if D is changing the same time the clock rises?
  - We cannot take its input
  - The clock period ($\mathbf{T_c}$) has to be long enough for D to be stable before we can take its input (also called sampling)
- This limitation restricts the system speed

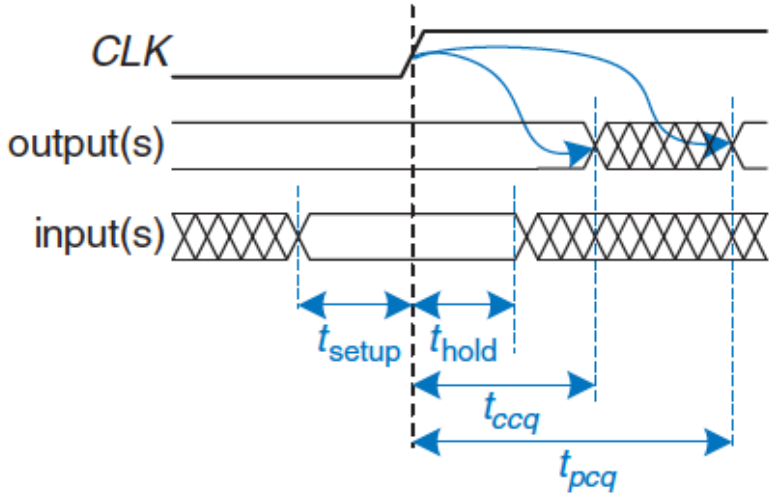
Timing Of Sequential Logic

- Sequential circuits have timing specifications to reason about the delay caused by changing signals
  - **setup time**: denoted t_{setup}, this is the length of time *before* the clock edge that input data must be stable for a circuit to *sample* the input correctly
  - **hold time**: denoted t_{hold}, this is the length of time *after* the clock edge that input data must be stable for the circuit to *sample* the input correctly
  - **aperture time**: $t_{setup} + t_{hold}$



Timing Of Sequential Logic

- Additional Specifications
  - **clock-to-*Q* contamination delay**: denoted t_{ccq}, this is the minimum length of time after the clock edge that we will expect to see changes in the output *Q*
  - **clock-to-*Q* propagation delay**: denoted t_{pcq}, this is the maximum length of time after the clock edge that the output *Q* is guaranteed to be stable (i.e., to stop changing)
- The inputs of a sequential circuit must be stable during the *setup* and *hold* time to guarantee that that flip-flop samples signals when they are stable
- t_{setup}, t_{hold}, t_{ccq}, and t_{pcq} are set by the manufacturer

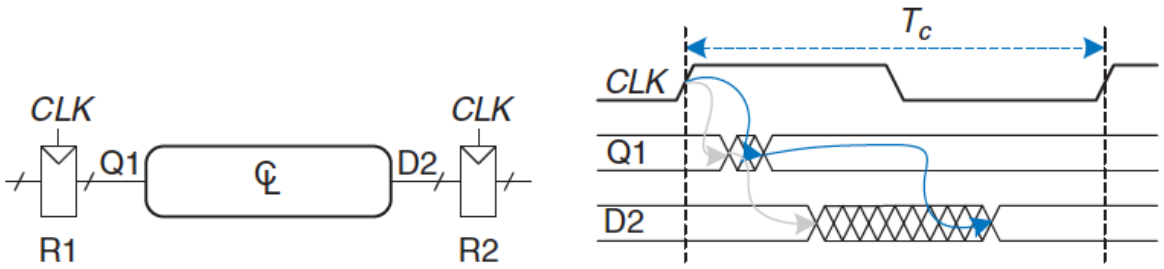


System Timing

- The clock period or cycle time, $\mathbf{T_c}$, is the time between rising edges of a repetitive clock cycle.
  - Measures of time:
    - 1 millisecond (ms) = 1 thousandth second = 10^{-3} second
    - 1 microsecond (μs) = 1 millionth second = 10^{-6} second
    - 1 nanosecond (ns) = 1 billionth second = 10^{-9} second
    - 1 picosecond (ps) = 1 trillionth second = 10^{-12} second
- Its reciprocal, $\mathbf{1/\ T_c}$, is the *clock frequency.*
  - Measures of speed:
    - 1 kilohertz (KHz) = 1 thousand Hz = 10^3 Hz
    - 1 megahertz (MHz) = 1 million Hz = 10^6 Hz
    - 1 gigahertz (GHz) = 1 billion Hz = 10^9 Hz
    - 1 terahertz (THz) = 1 trillion Hz = 10^{12} Hz
- All else being same, increasing the clock frequency increases the work a digital system can accomplish per unit time.

Analyzing Timing Constraints

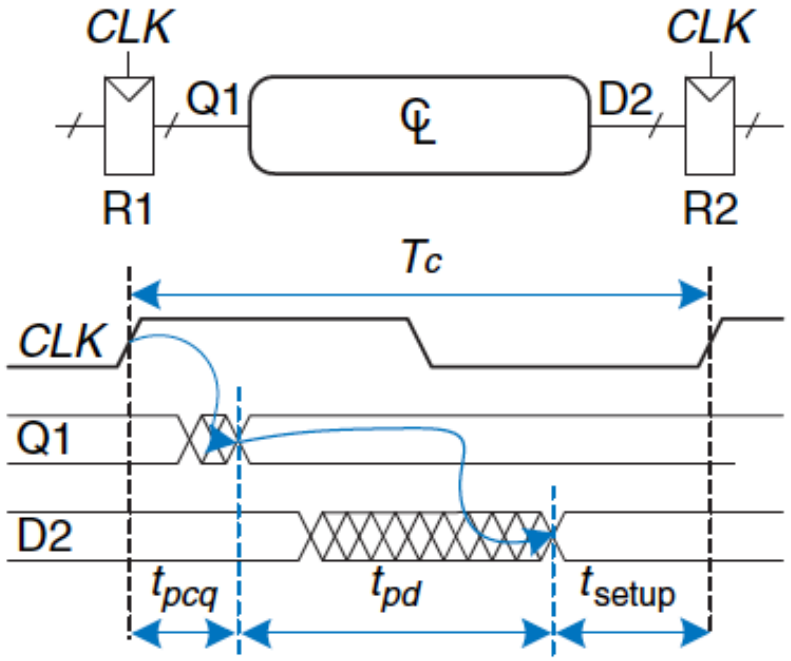
- Consider the sequential circuit below whose clock period we wish to determine



- On the rising edge of the clock, register R1 produces Q1
- Q1 enters a combinational logic block and produces D2, the input to register R2
- Gray arrows indicate contamination delay thru R1 and combinational logic
- Blue arrows indicate propagation delay thru R1 and combinational logic
- We will use this to analyze the setup time and hold time of R2

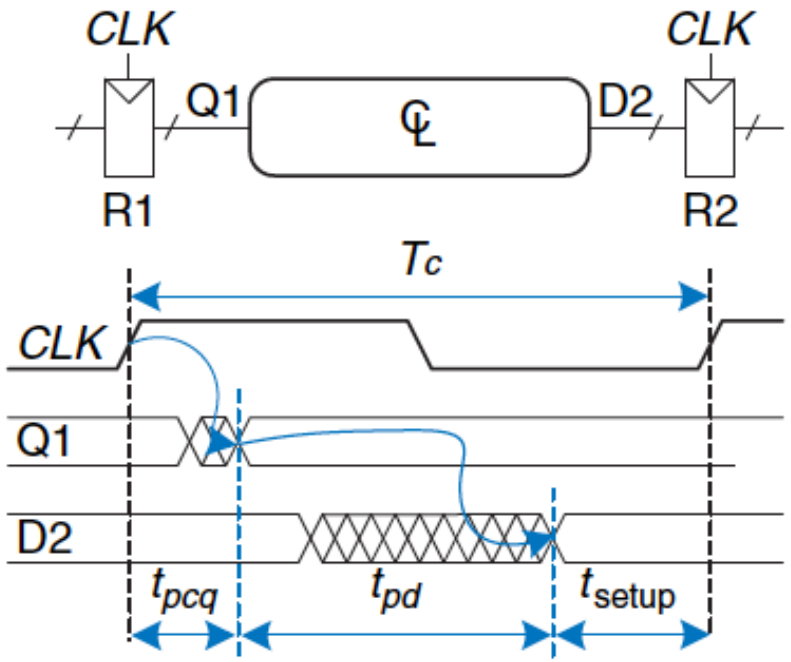
Setup Time Constraint

- D2 must settle no later than the *setup time* before the next clock edge
- Hence, the clock period cannot be shorter than the propagation delay of R1 and the combinational logic and setup time of R2
  - $\boldsymbol{T_c \geq t_{pcq} + t_{pd} + t_{setup}}$
- Clock period is often set by marketing in commercial designs
- Propagation delays and setup times are set by the manufacturer of the flip-flops
- Hence, we rearrange to obtain an upper bound on $\boldsymbol{t_{pd}}$
  - $\boldsymbol{t_{pd} \leq T_c - (t_{pcq} + t_{setup})}$
- This is inequality is the *setup time constraint* or *max-delay constraint*



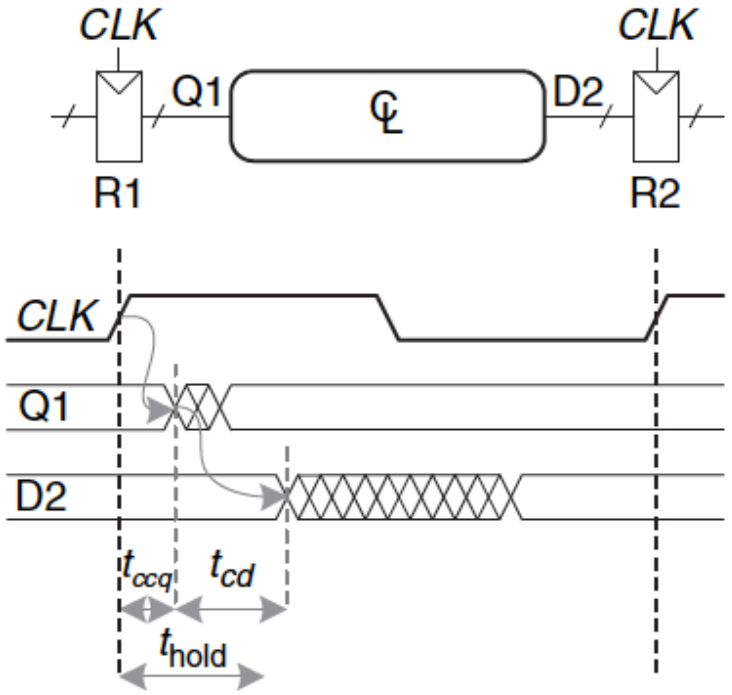
Setup Time Constraint

- The term $(t_{pcq} + t_{setup})$ is called sequencing overhead
- Ideally, the entire clock period is available to combinational logic, but sequencing the flip-flop cuts into the time
- What happens if t_{pd} is too large?
  - D2 may not have settled when R2 samples it
  - The circuit will malfunction
- To solve this problem, either
  - Increase clock period, or
  - Shorten combinational logic propagation delay



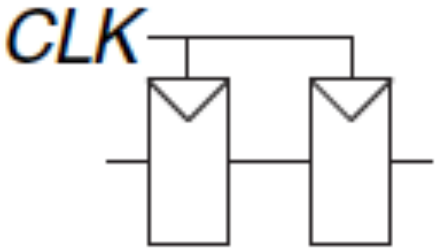
Hold Time Constraint

- Recall that the input data needs to be stable for t_{hold} time after the clock's rising edge for the circuit to sample correctly
- The register R2 must also have a *hold time constraint*
- The figure shows that D2 will change as early as $t_{ccq} + t_{cd}$ after the rising edge of the clock. Why?
  - Q1 changes as early as t_{ccq}
- Hence, it must be that $\boldsymbol{t_{hold} \leq t_{ccq} + t_{cd}}$
- But t_{hold} and t_{ccq} are fixed by manufacturers
- Therefore, we rearrange the inequality to get a lower bound on t_{cd}
  - $\boldsymbol{t_{cd} \geq t_{hold} - t_{ccq}}$ *(hold time constraint* or *min-delay constraint)*



Hold Time Constraint

- What will happen to *hold time constraint* if two flip-flops are cascaded back-to-back, with no combinational logic between them (see figure)
  - t_{cd} = 0. No contamination delay due combinational logic
  - Hence, hold time constraint is $\boldsymbol{t_{hold} \leq t_{ccq}}$
- *Thus, a reliable flip-flop must have a hold time shorter than its contamination delay*
- Often flip-flops are designed to have t_{hold} = 0, hence the constraint will always hold
- Even then, hold time constraint is critical
  - If violated the only solution is to increase contamination delay or add buffers to increase delay
  - This involves redesigning the circuit, which is expensive
- In further discussions, we will assume that $\boldsymbol{t_{hold}}$ = 0 unless stated otherwise

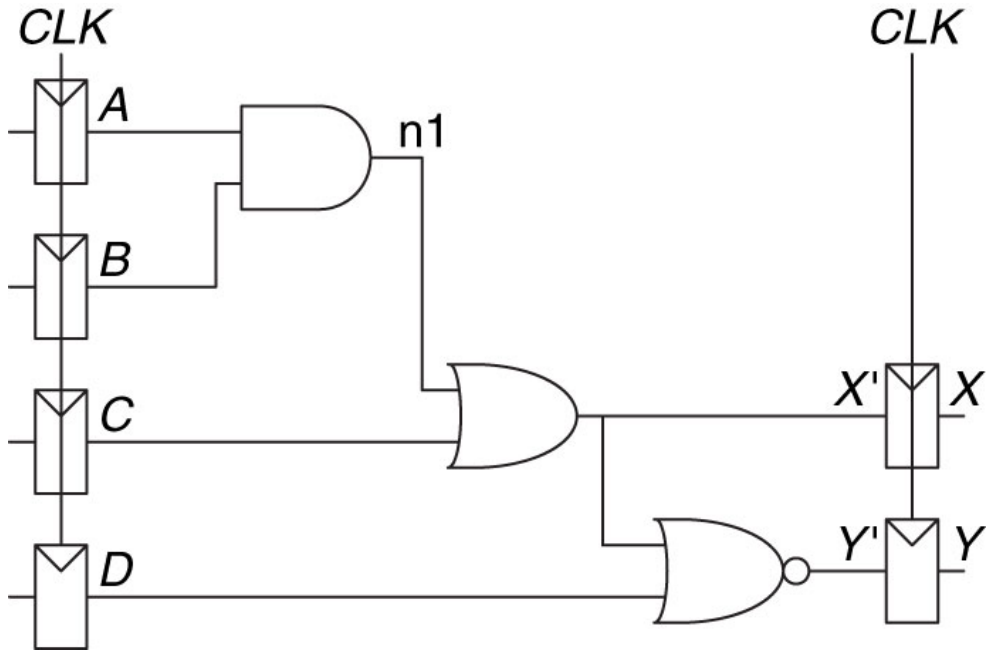


Putting It Together

- Sequential circuits have setup and hold time constraints that dictate the max and min delays of combination circuits between flip-flops
- Modern flip-flops are designed assuming that min delay thru combinational logic is 0, i.e, flip-flops can be cascaded back-to-back
- The max delay constraint limits the no. of consecutive gates on the critical path of a circuit, because a high frequency means a shorter clock period

Timing Analysis: Example #1

- Consider the circuit below and the given timing specifications
- Suppose we wanted to compute the minimum clock period for this circuit to function properly.



Flip-flops

$t_{ccq} = 30$ ps

$t_{pcq} = 50$ ps

$t_{setup} = 60$ ps

$t_{hold} = 70$ ps

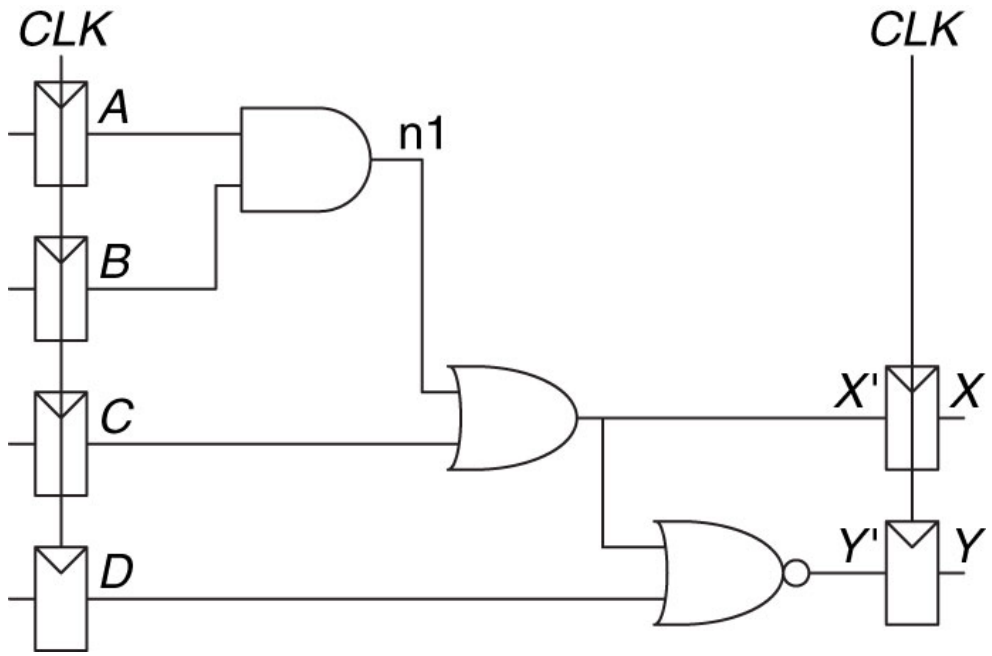
Gates

$t_{cd} = 25$ ps

$t_{pd} = 35$ ps

Timing Analysis: Example #1

- We know that $T_c \geq t_{pcq} + t_{pd} + t_{setup}$
- t_{pcq} and t_{setup} are fixed. Hence, we determine t_{pd} along critical path



Flip-flops

$t_{ccq} = 30$ ps

$t_{pcq} = 50$ ps

$t_{setup} = 60$ ps

$t_{hold} = 70$ ps

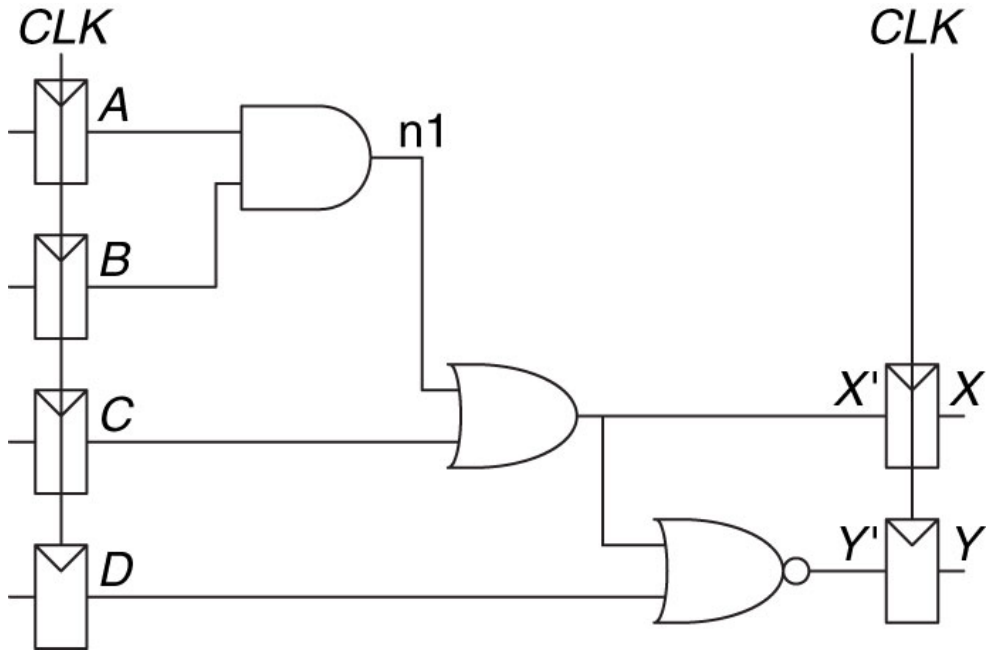
Gates

$t_{cd} = 25$ ps

$t_{pd} = 35$ ps

Timing Analysis: Example #1

- The critical path occurs when A rises from 0 to 1, B = 1, C = 0, and D = 0
- 3 gates activated along critical path. Hence, t_{pd} = 3 X 35 = 105 ps



Flip-flops

$t_{ccq} = 30$ ps

$t_{pcq} = 50$ ps

$t_{setup} = 60$ ps

$t_{hold} = 70$ ps

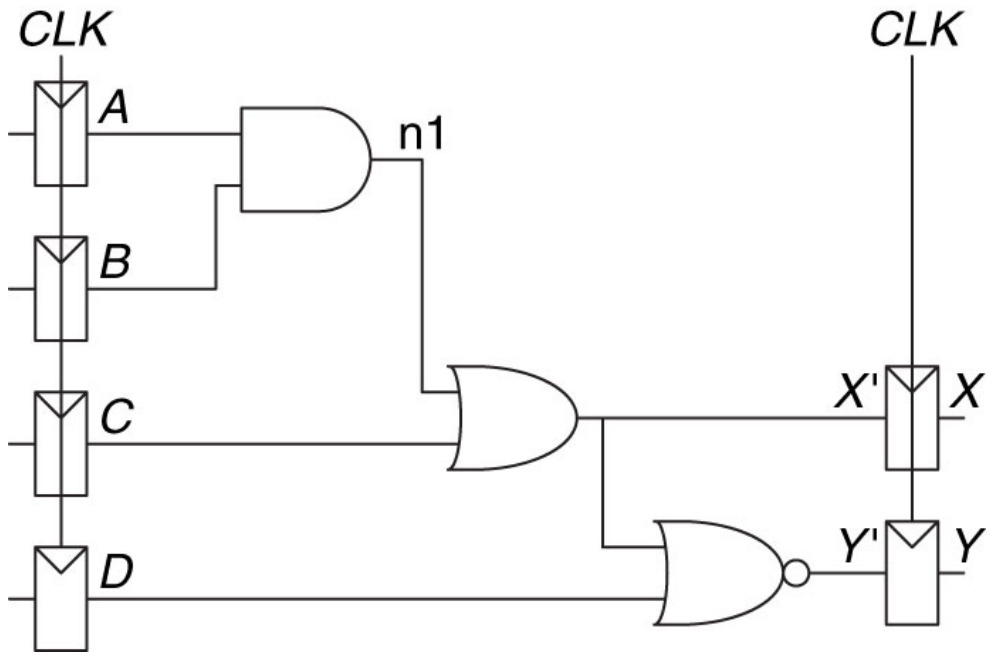
Gates

$t_{cd} = 25$ ps

$t_{pd} = 35$ ps

Timing Analysis: Example #1

- Hence, $T_c \geq t_{pcq} + t_{pd} + t_{setup} \Rightarrow T_c \geq 50 + 105 + 60 = 215\ ps$
- Max. clock frequency, $1/T_c$, is approx. 4.5 GHz



Flip-flops

$t_{ccq} = 30$ ps

$t_{pcq} = 50$ ps

$t_{setup} = 60$ ps

$t_{hold} = 70$ ps

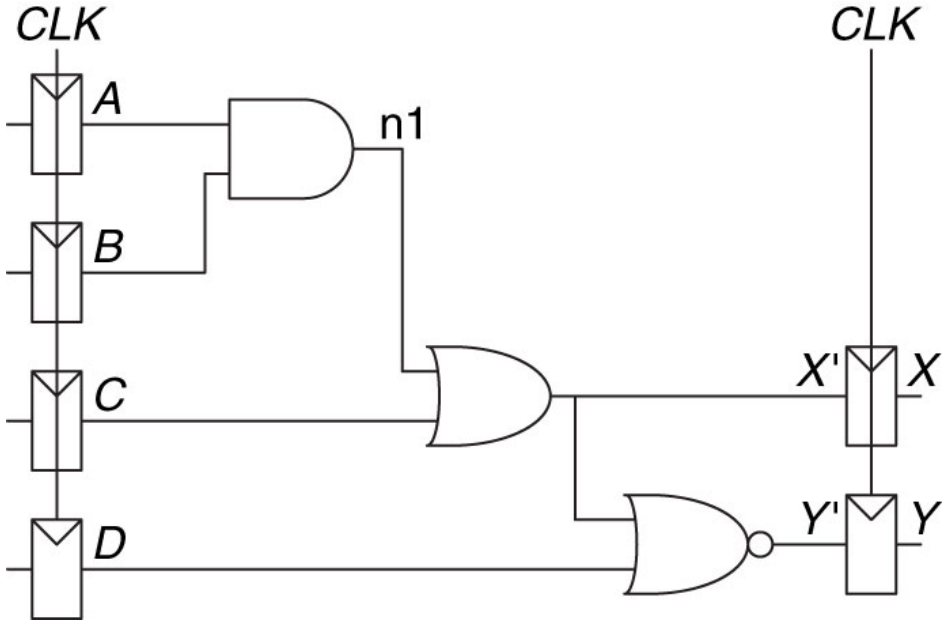
Gates

$t_{cd} = 25$ ps

$t_{pd} = 35$ ps

Timing Analysis: Example #1

- Let us check the hold time constraint
- We know that $t_{hold} \leq t_{ccq} + t_{cd}$
- We need to compute t_{cd} since the others are fixed



Flip-flops

$t_{ccq} = 30$ ps

$t_{pcq} = 50$ ps

$t_{setup} = 60$ ps

$t_{hold} = 70$ ps

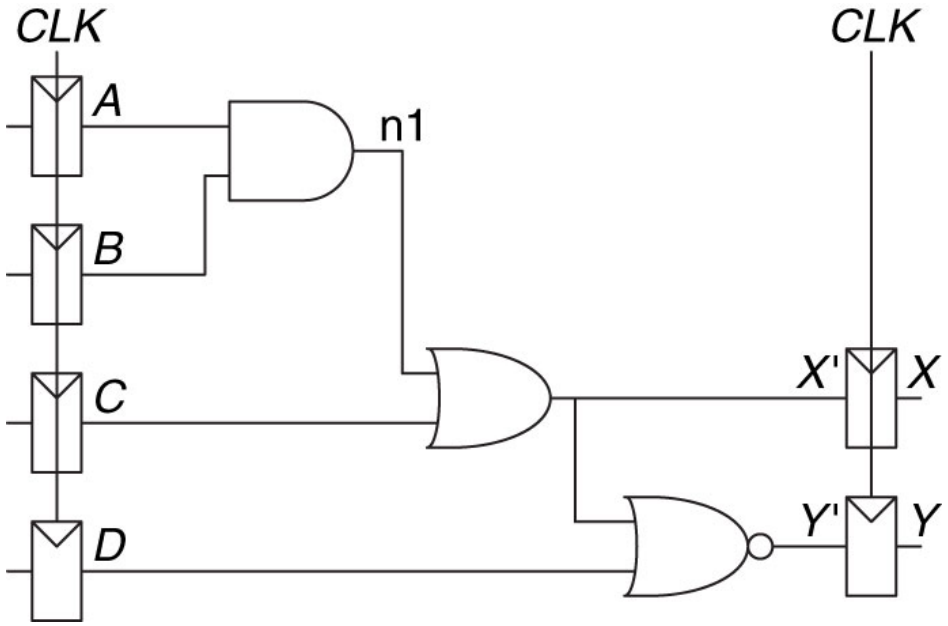
Gates

$t_{cd} = 25$ ps

$t_{pd} = 35$ ps

Timing Analysis: Example #1

- t_{cd} dictates the earliest time when the input will begin to change after the rising edge of the clock cycle
- Hence, t_{cd} is based on the shortest path that occurs when A = 0 and C rises



Flip-flops

$t_{ccq} = 30$ ps

$t_{pcq} = 50$ ps

$t_{setup} = 60$ ps

$t_{hold} = 70$ ps

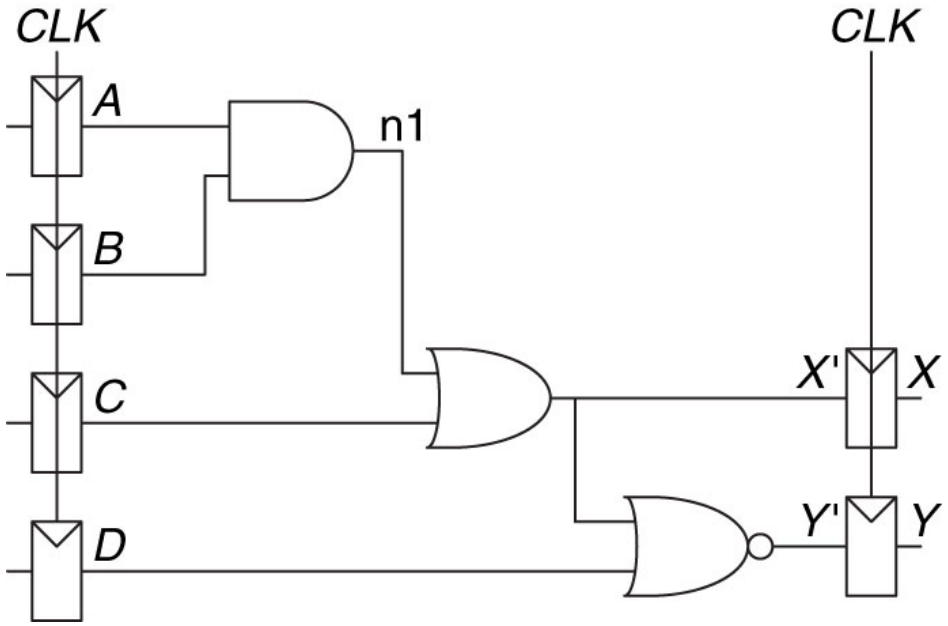
Gates

$t_{cd} = 25$ ps

$t_{pd} = 35$ ps

Timing Analysis: Example #1

- Thus, $t_{cd} = 25\ ps$ (involves only 1 gate)
- $t_{hold} \leq t_{ccq} + t_{cd}$ => $70 \leq 55$ (constraint not met!)



Flip-flops

$t_{ccq} = 30$ ps

$t_{pcq} = 50$ ps

$t_{setup} = 60$ ps

$t_{hold} = 70$ ps

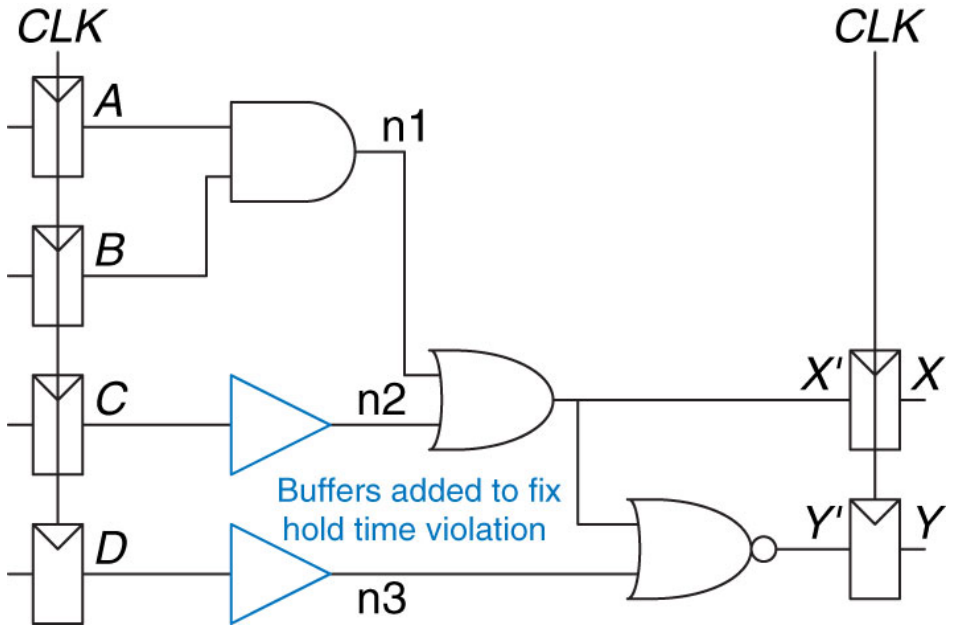
Gates

$t_{cd} = 25$ ps

$t_{pd} = 35$ ps

Timing Analysis: Example #1

- Hold time constraint violation can be fixed by increasing delay
- To this end, we add buffers after C and D. Why both C and D?
  - Because if D = 1, the Y' will change before X' causing the hold time constraint violation again!



Flip-flops

$t_{ccq} = 30$ ps

$t_{pcq} = 50$ ps

$t_{setup} = 60$ ps

$t_{hold} = 70$ ps

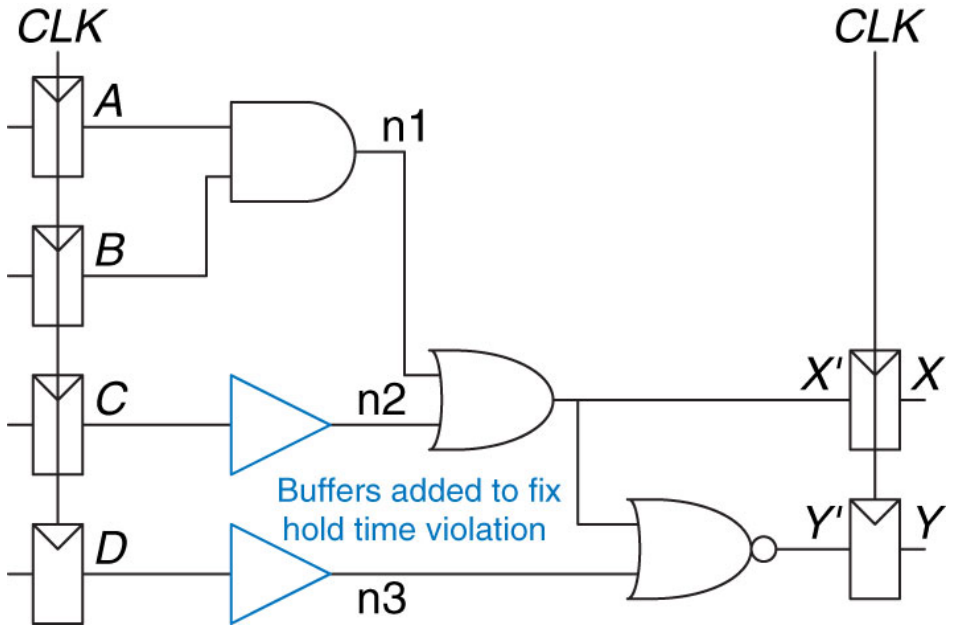
Gates

$t_{cd} = 25$ ps

$t_{pd} = 35$ ps

Timing Analysis: Example #1

- Thus, $t_{cd} = 50\ ps$ (involves 2 gates)
- $t_{hold} \leq t_{ccq} + t_{cd}$ => $t_{hold} \leq$ 80 ps (constraint met!)
- However, this will slow down the circuit



Flip-flops

$t_{ccq} = 30$ ps

$t_{pcq} = 50$ ps

$t_{setup} = 60$ ps

$t_{hold} = 70$ ps

Gates

$t_{cd} = 25$ ps

$t_{pd} = 35$ ps