CS221: Digital Design

Sequential Logic Design (Latch & FF)

A. Sahu

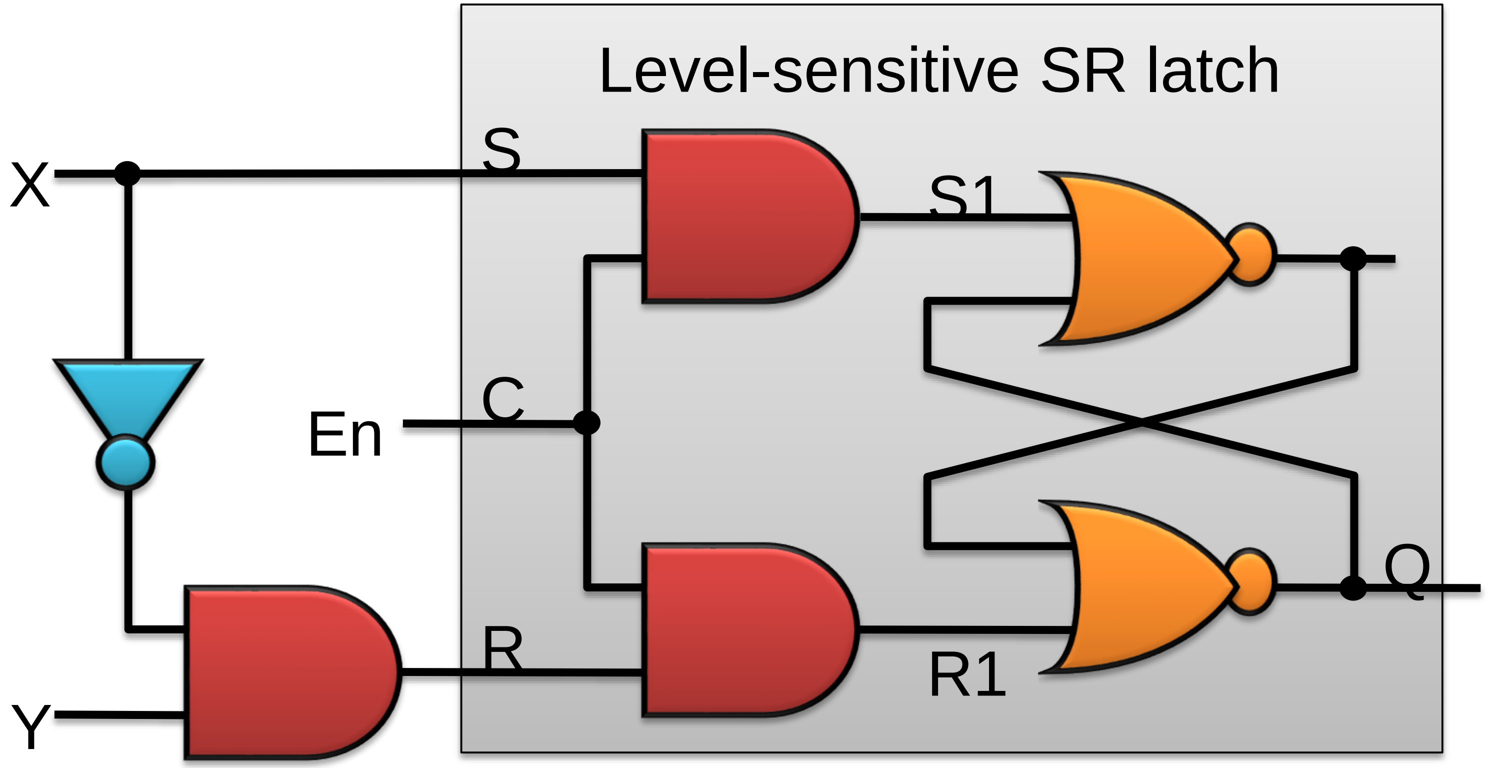
Dept of Comp. Sc. & Engg.

Indian Institute of Technology Guwahati

Outline

- Combinational Vs Sequential Logic Design
- Design a **flip-flop**, that stores one bit
  - RS latch
- Stabilizing RS latch : Level Sensitive
- Clocked Latch : Flip Flop- Edge Sensitive
- D, JK, T flip flops
- Characterization Table and Equation
  - RS, D, JK and T Flip flop

Solution: Ensure, Stabilize, Store



Ensure Stage

Never Happens SR=11

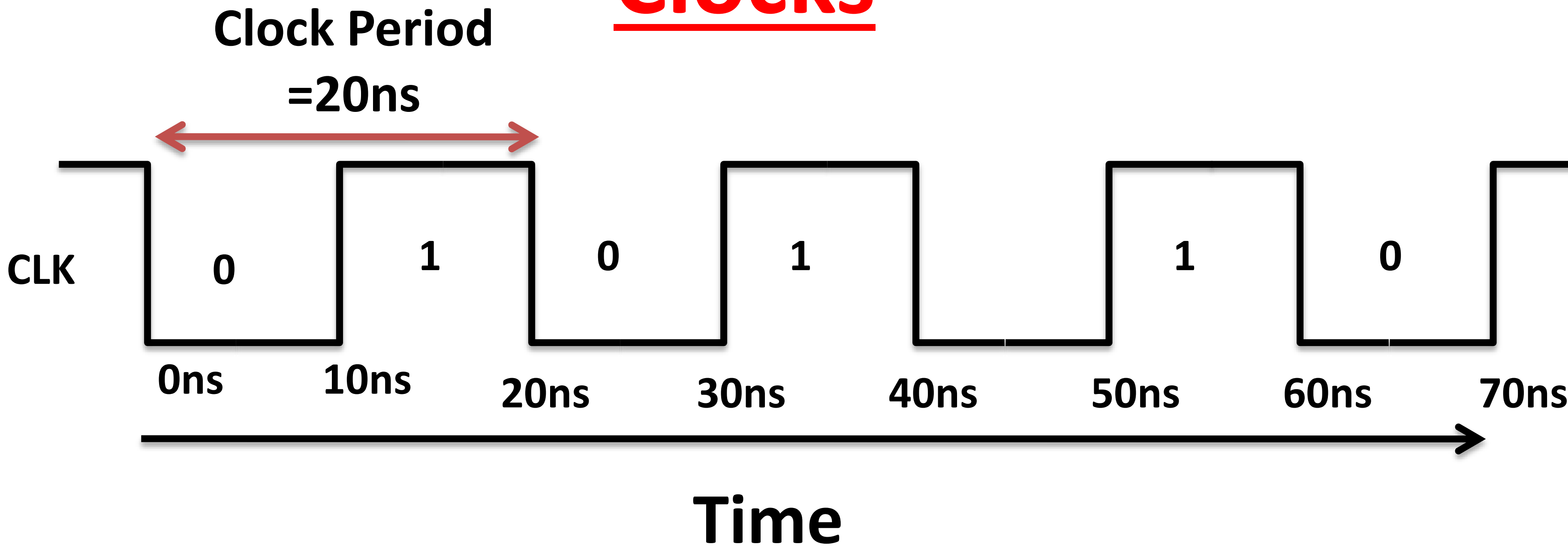
Stabilize Stage

When C=0 Stabilize SR and Use when C=1

Store Stage

Store bit

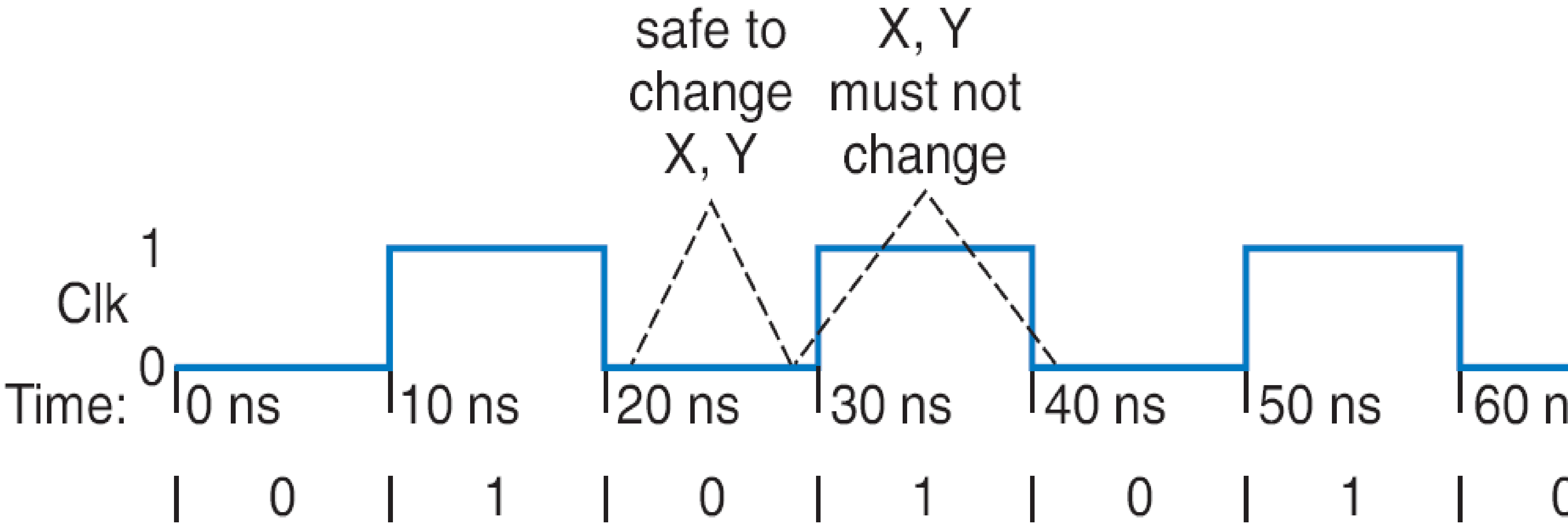
Clocks

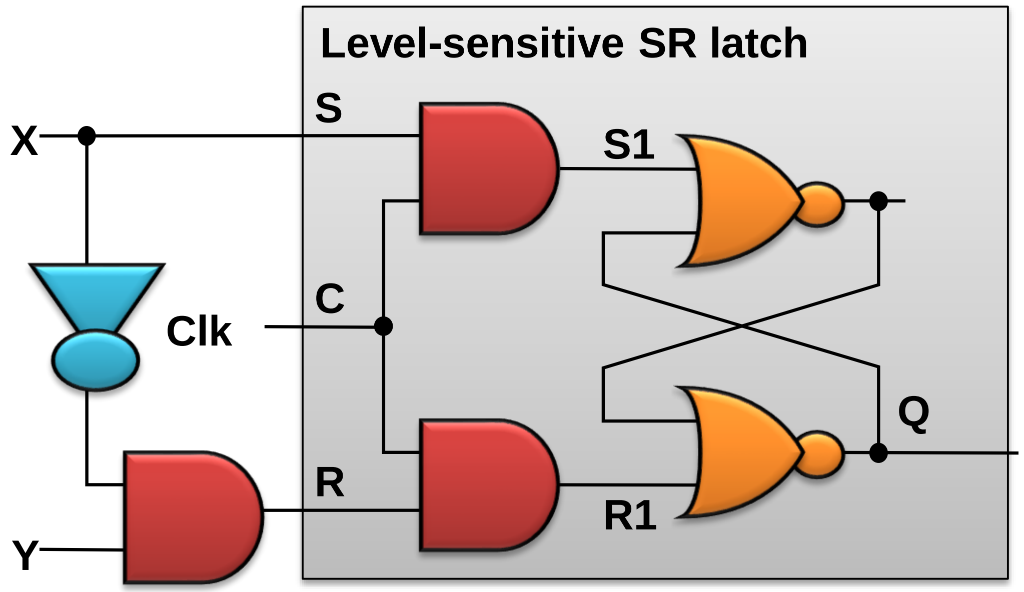


- ***Clock period***: time interval between pulses
- ***Clock cycle***: one such time interval
- ***Clock frequency***: 1/period
  - frequency = 1 / 20 ns = 50 MHz
    - 1 Hz = 1/s

Freq	Period
1 GHz	1 ns
100 MHz	10 ns
10 MHz	100 ns

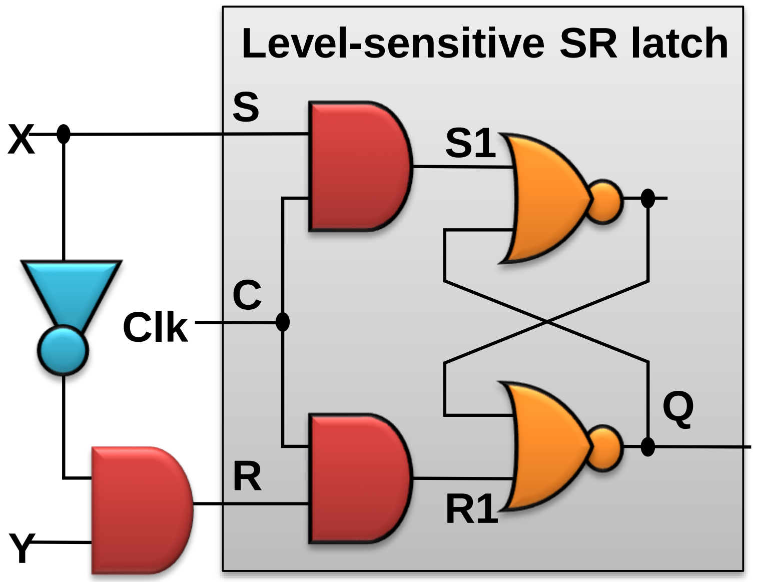
Clock Signal for RS latch





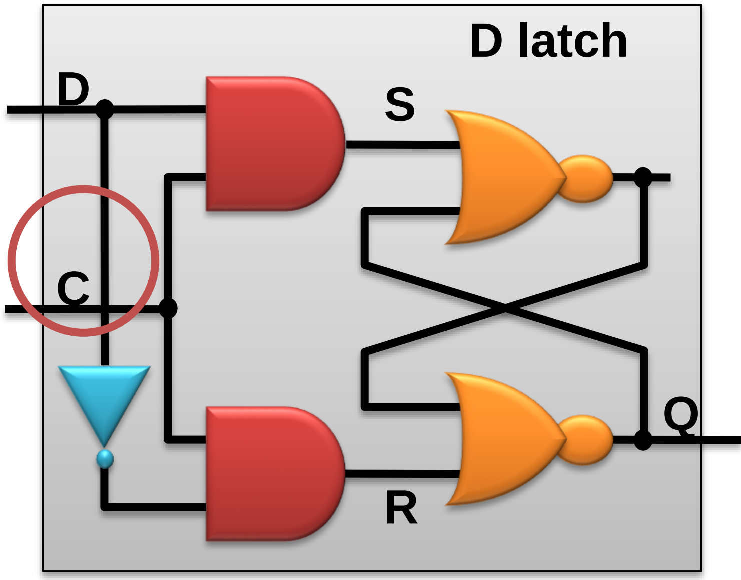
Clock Signals for a RS Latch

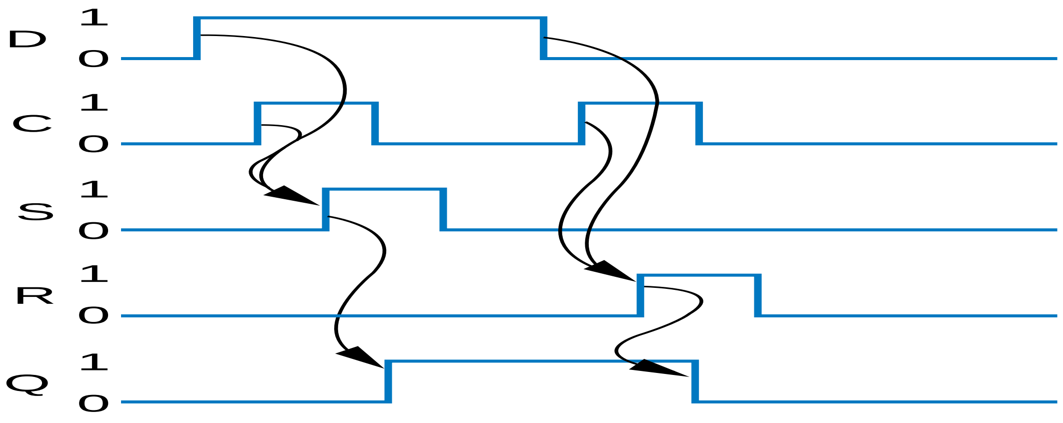
- How do we know when it's safe to set C=1?
  - Most common solution –make C pulse up/down
  - C=0: Safe to change X, Y   C=1: Must *not* change X, Y
  - ***Clock*** signal -- Pulsing signal used to enable latches
    - Because it ticks like a clock
  - Sequential circuit whose storage components all use clock signals: ***synchronous*** circuit

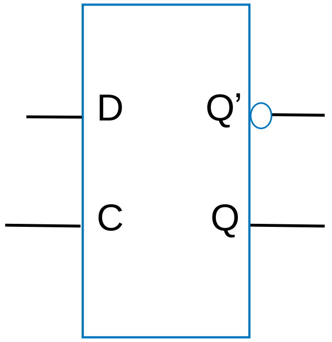


Level-Sensitive D Latch

- SR latch requires careful design to ensure SR=11 never occurs
- D latch relieves designer of that burden
  - Inserted inverter ensures R always opposite of S



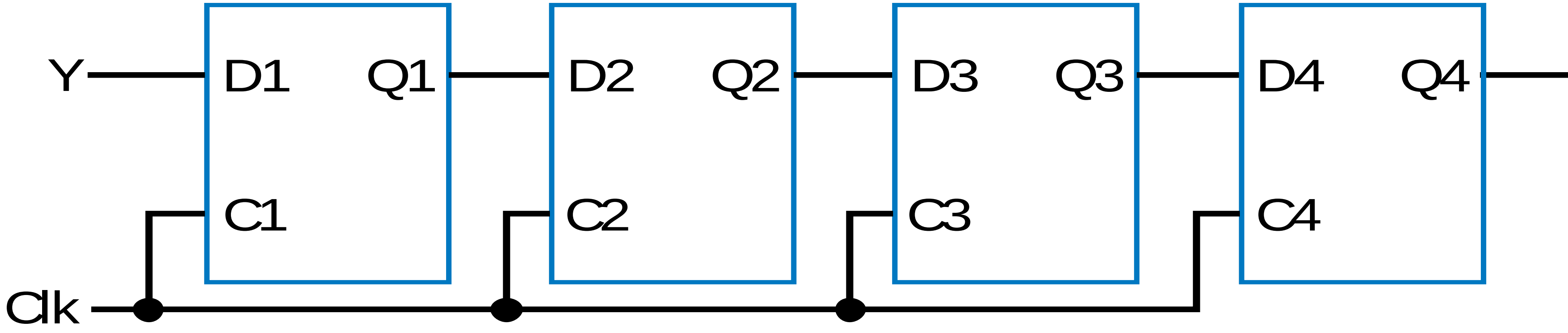




D latch symbol

Level-Sensitive D Latches

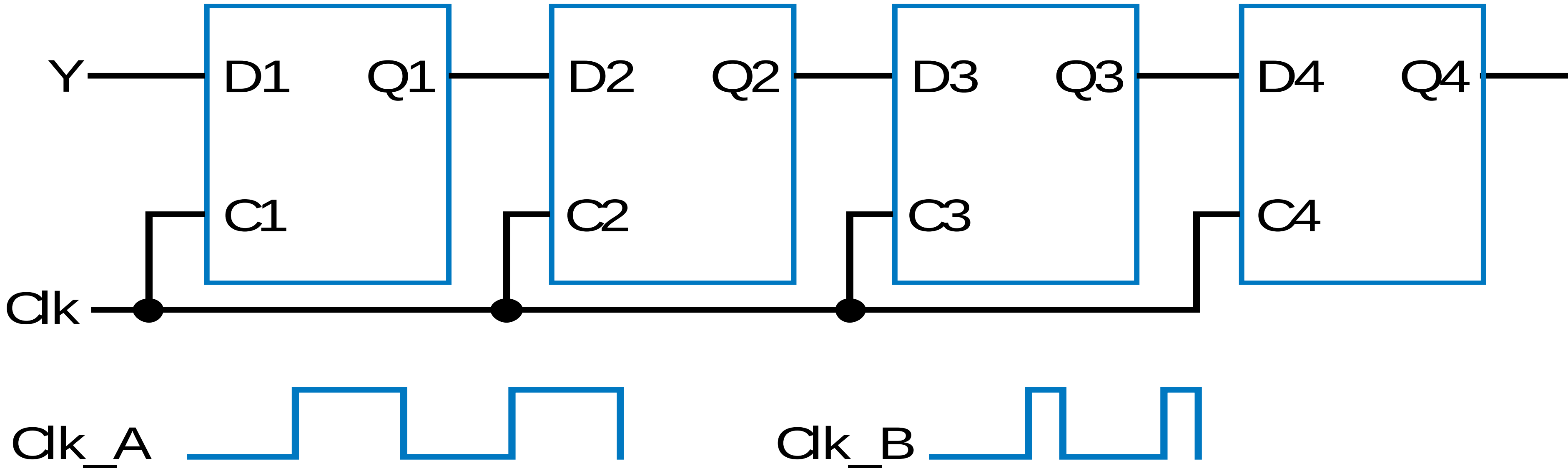
- Suppose D FFs are arrange in linear fashion connected using a single clock signal Clk
- Every clock we want to shift one bit to right
  - Right shift one bit per cycle



Does this circuit (with level sensitive D latch) Shift one bit per cycle ?

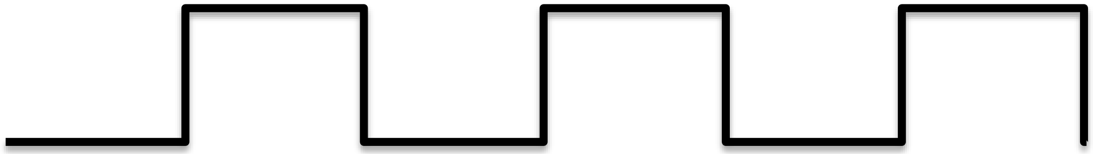
Problem with Level-Sensitive D Latch

- D latch still has problem (as does SR latch)
  - When C=1, through how many latches will a signal travel?
  - Depends on for how long C=1
    - Clk_A -- signal may travel through multiple latches
    - Clk_B -- signal may travel through fewer latches
  - **Hard to pick C that is just the right length**



Problem with Level-Sensitive D Latch

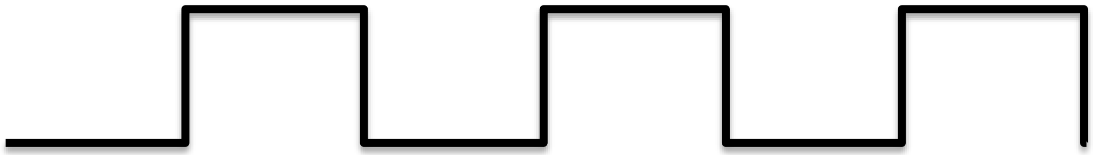
- We want do the work: one per clock cycle
  - Independent of length of clock (1 time)



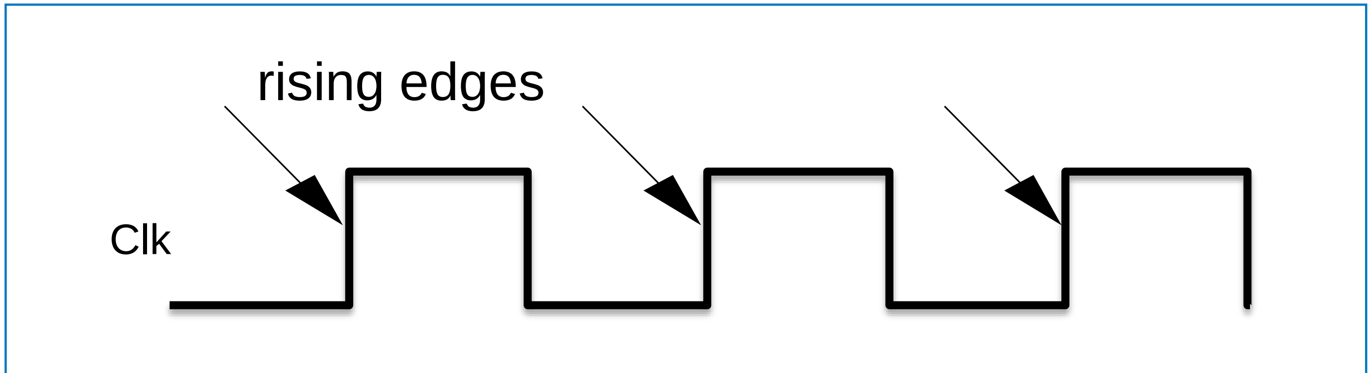
Is there any solution to this ?

Problem with Level-Sensitive D Latch

- We want do the work: one per clock cycle
  - Independent of length of clock (1 time)



- Can we design bit storage that only stores a value on the rising edge of a clock signal?
  - There is exactly one rising edge per clock cycle
  - There is exactly one falling edge per clock cycle



Make Edge Sensitive Bit Storage

- Latch : Level sensitive storage
- Flip-Flop : Edge sensitive storage
  - Value get changed only at edges of clock

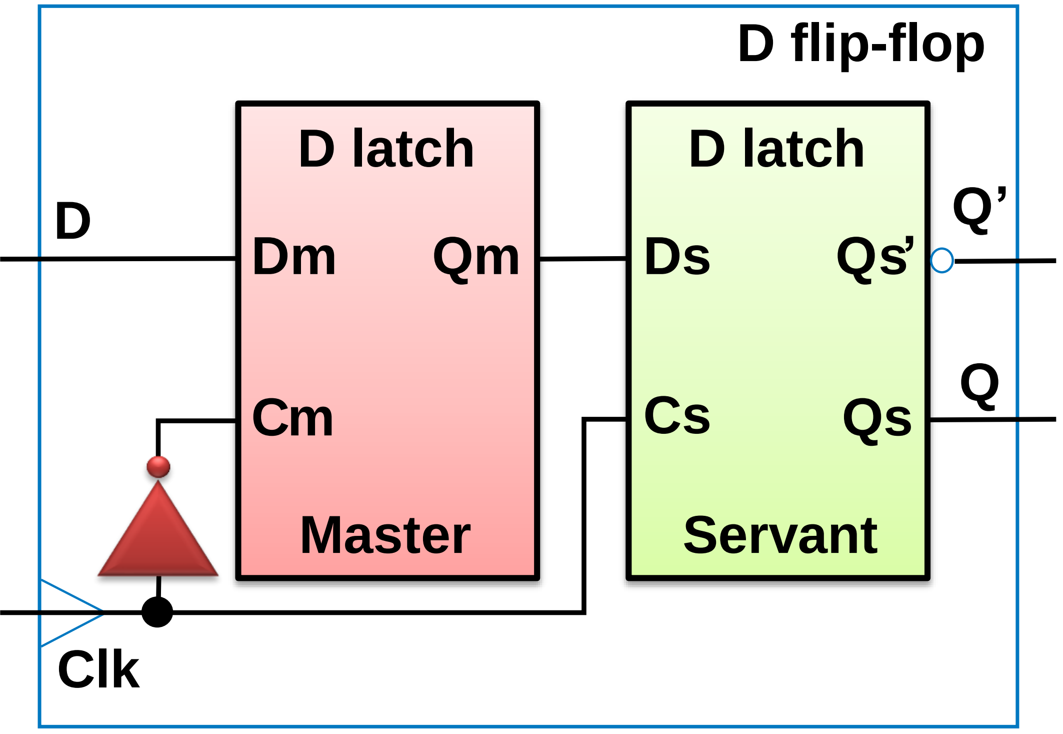
How to make a Flip Flop out of Latch?

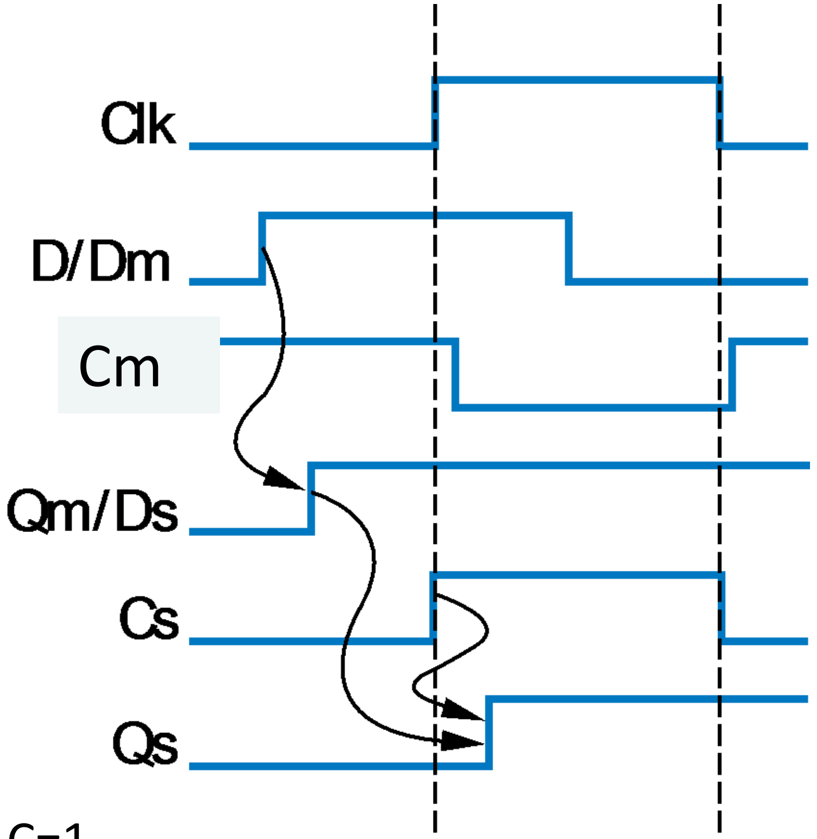
Master -Slave D Flip-Flop

- **Two latches**, output of first goes to input of second, master latch has inverted clock signal
- So master loaded when C=0, then servant when C=1
- When C changes from 0 to 1, master disabled, servant loaded with value that was at D just before C changed
  - i.e., Value at D during rising edge of C

Master -Slave D Flip-Flop

- ***Flip-flop***: stores on clock edge, not level

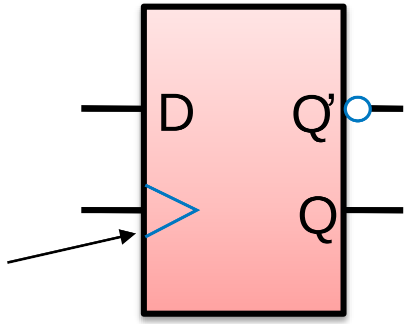




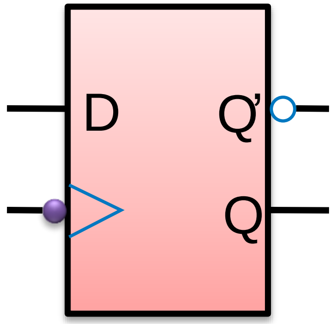
Master loaded when C=0, then servant when C=1
When C changes from 0 to 1, master disabled, servant loaded with value that was at D just before C changed -- i.e., value at D during rising edge of C

D Flip-Flop (Rising & Falling Edges)

The triangle means clock input, edge triggered



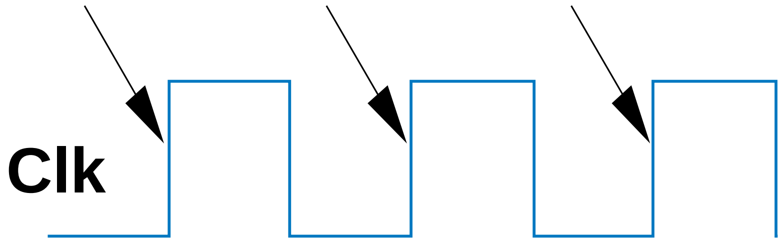
Symbol for rising-edge triggered D flip-flop



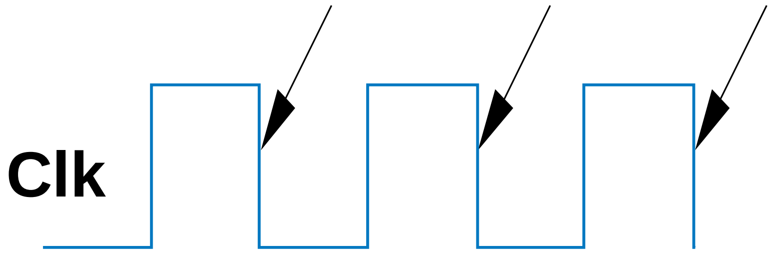
Internal design: Just invert servant clock rather than master

Symbol for falling-edge triggered D flip-flop

Rising edges

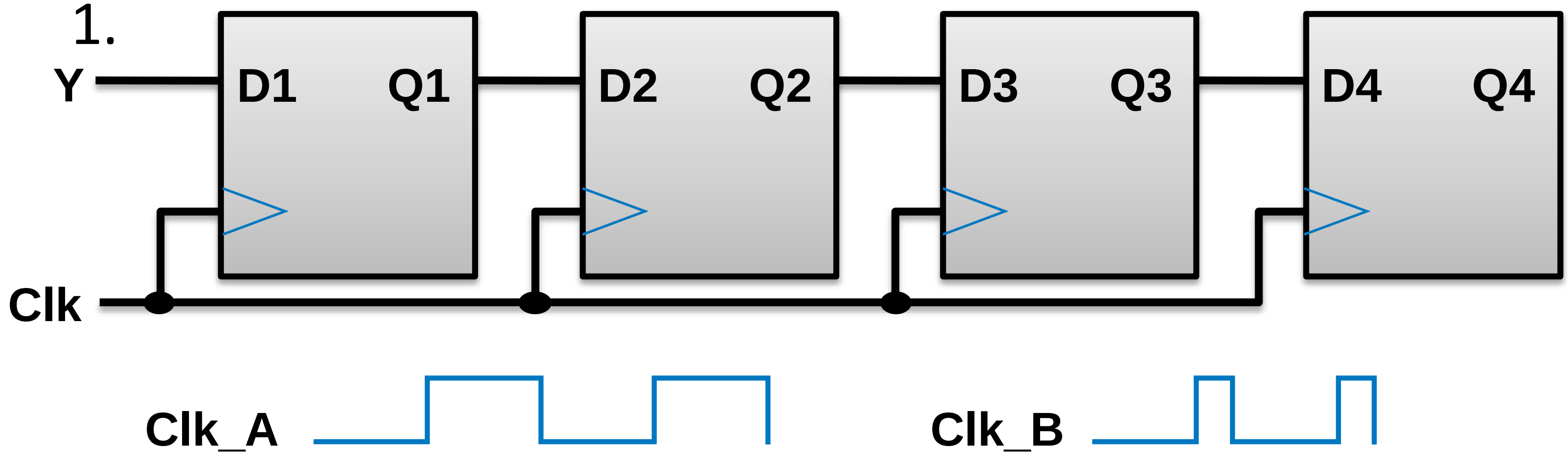


Falling edges



D Flip-Flops

- Solves problem of not knowing through how many latches a signal travels when C=1
  - Signal travels through exactly one FF, for Clk_A or Clk_B. **Why?**
  - Because on rising edge of Clk, all four flip-flops are loaded simultaneously -- then all four no longer pay attention to their input, until the next rising edge. Doesn't matter how long Clk is 1.

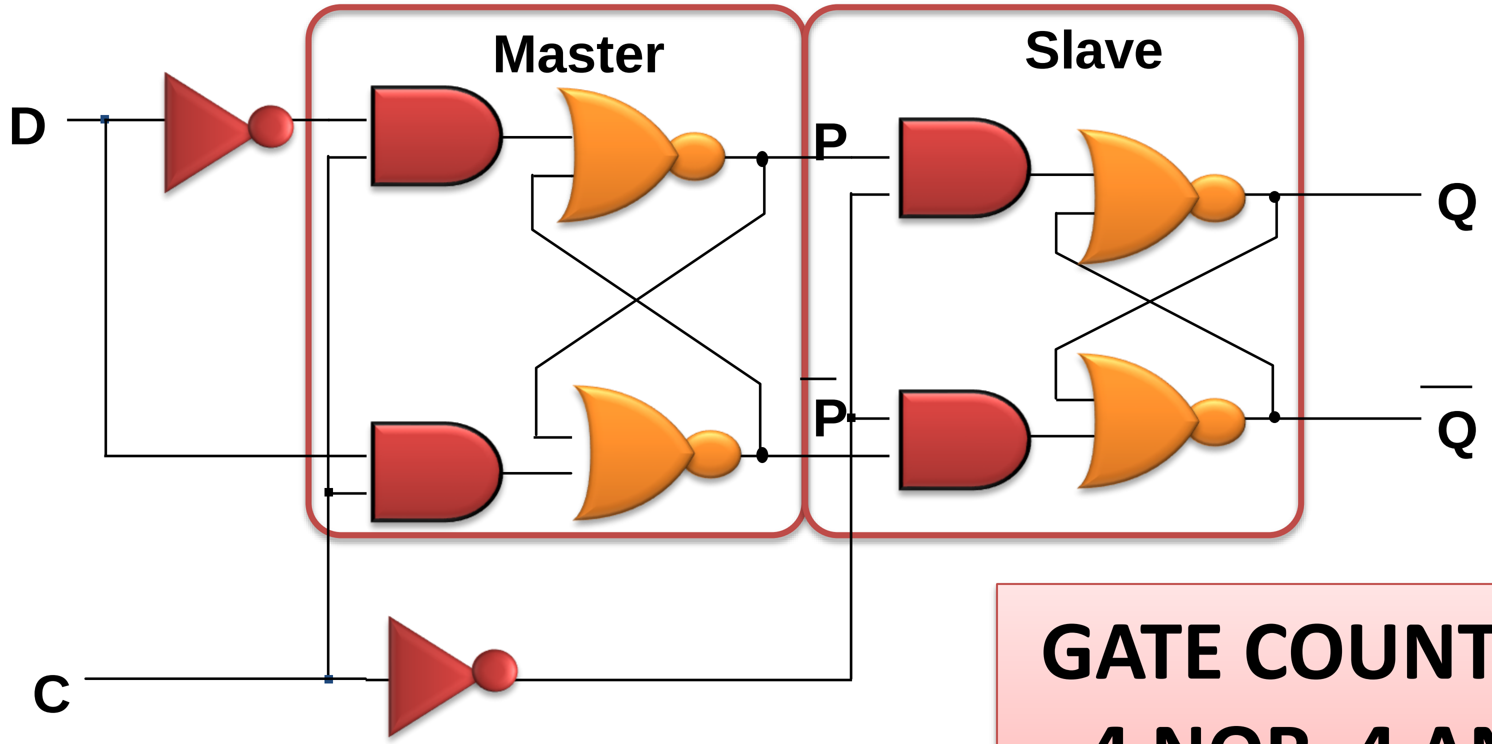


Two latches inside each flip-flop

D Latch vs. D Flip-Flop

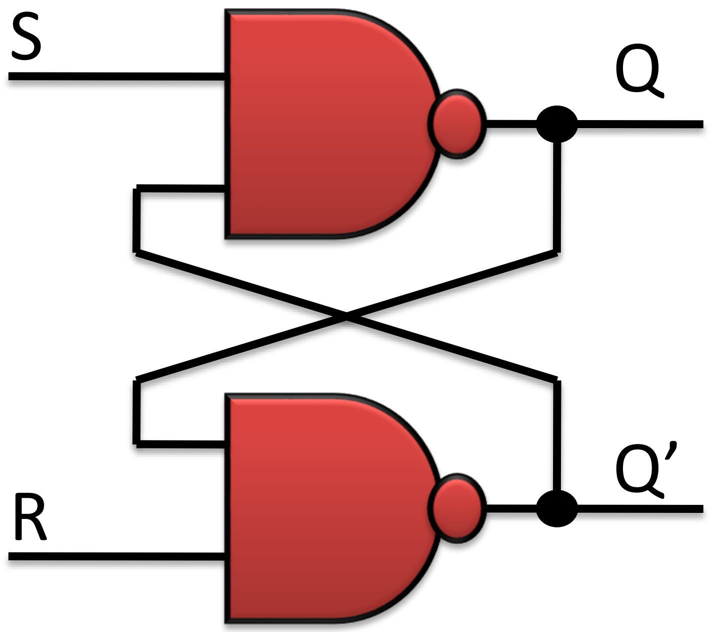
- Latch is level-sensitive: Stores D when C=1
- Flip-flop is edge triggered: Stores D when C changes from 0 to 1
  - Saying “level-sensitive latch,” or “edge-triggered flip-flop,” is redundant
  - Two types of flip-flops -- rising or falling edge triggered.

Positive Edge Triggered D-Flip Flop: Optimization



GATE COUNT: 10
4 NOR, 4 AND
and 2 NOT

Remember: SR Latch with NAND Gates

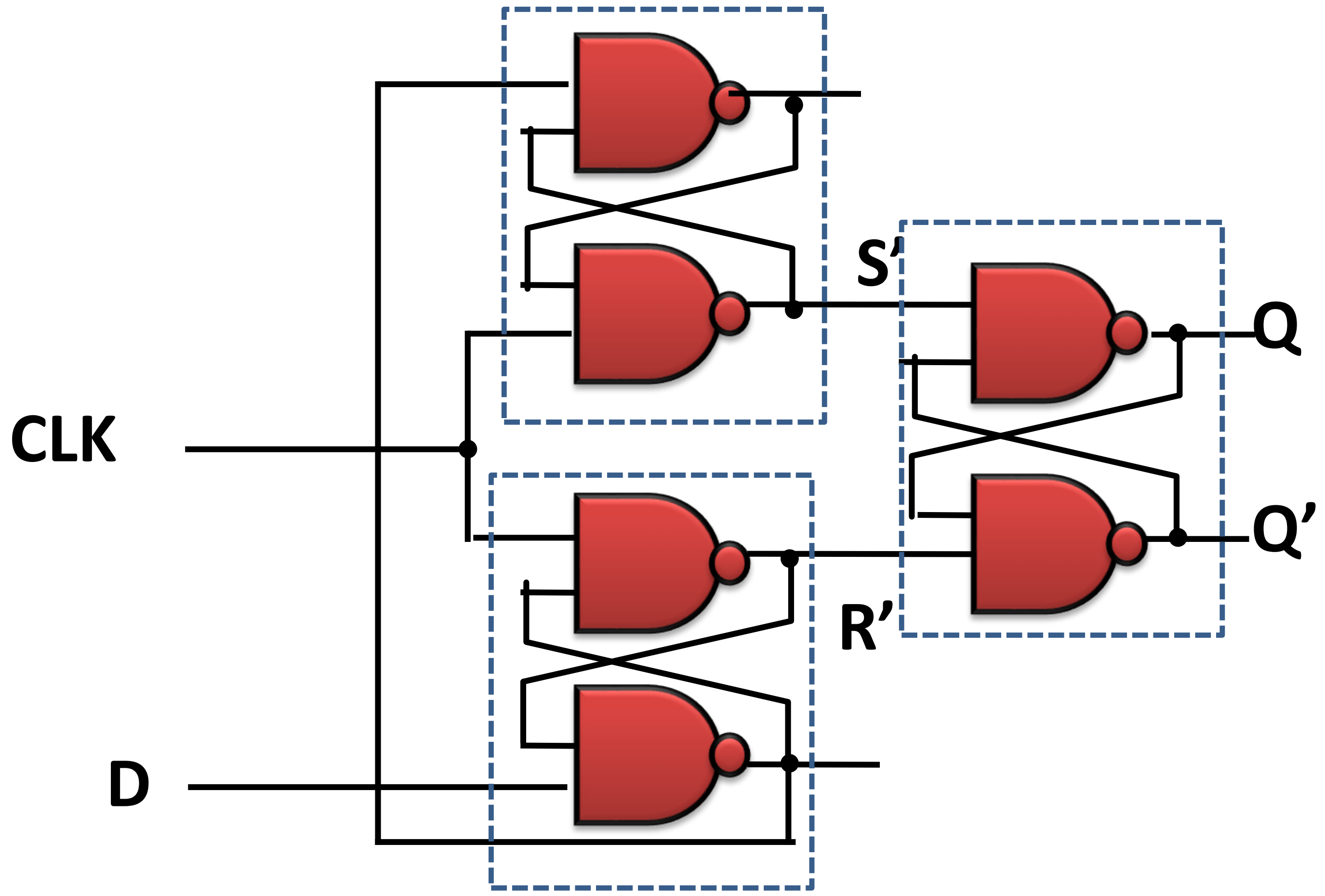


S	R	Q+
0	0	1*0* (Unpredictable)
0	1	1
1	0	0
1	1	Q

Opposite to SR Latch with NOR Gates

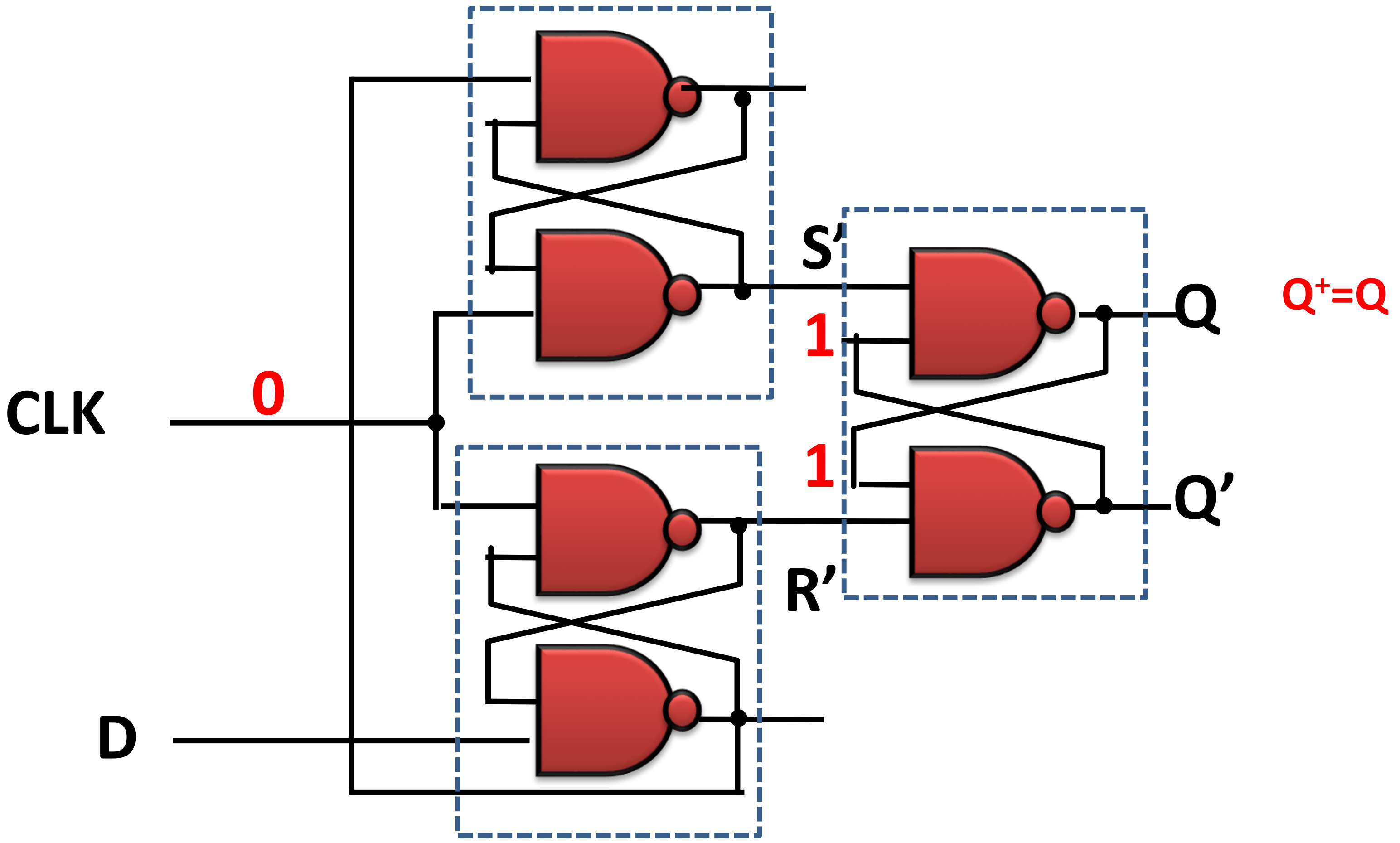
Set will do Q=0 and Reset will Q=1

Positive-Egde Trigeered D-FF: Economical



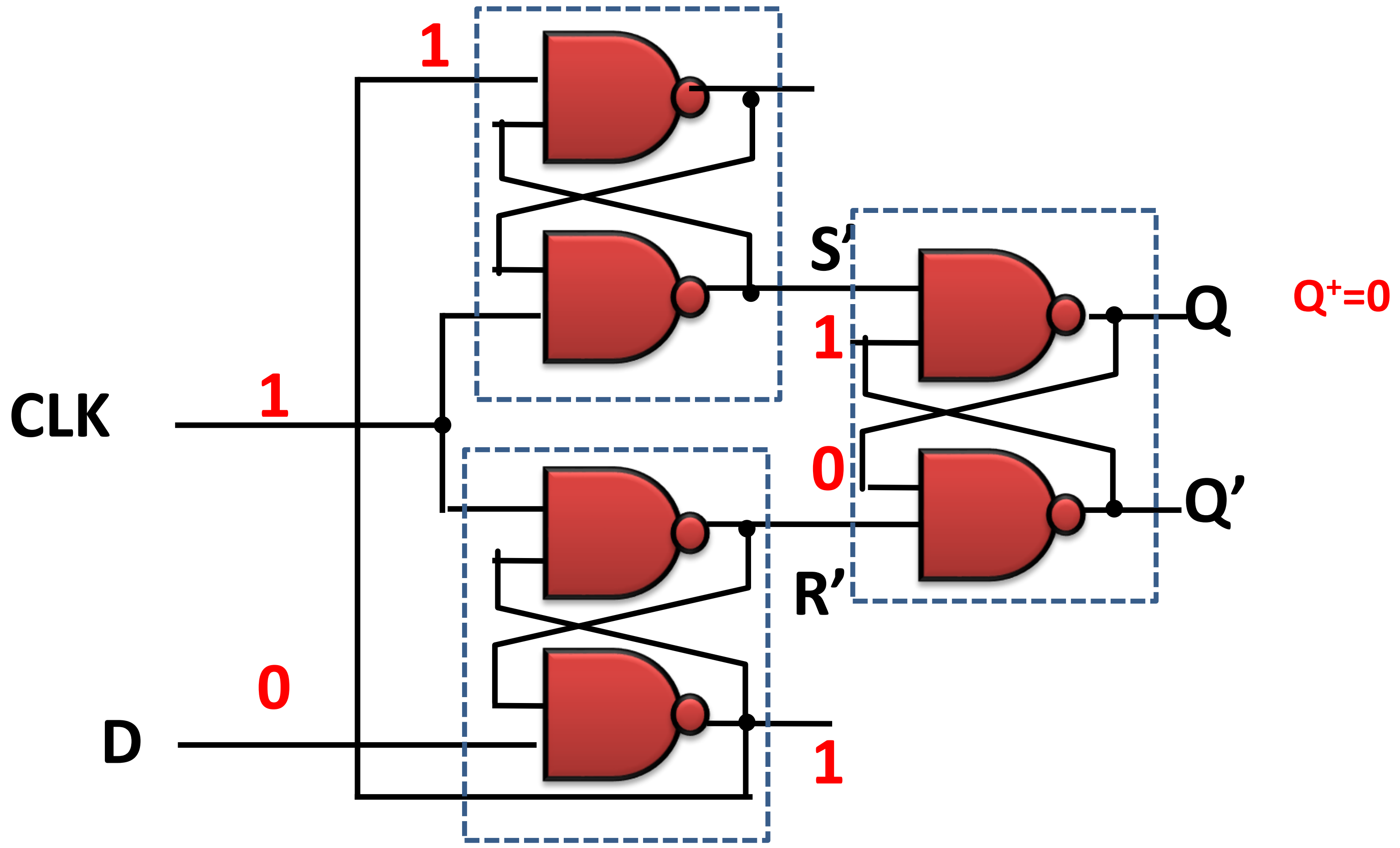
Positive-Edge Triggered D-FF: Economical

- When CLK=0, S'R'=11, $Q^+=Q$ (Independent of D)



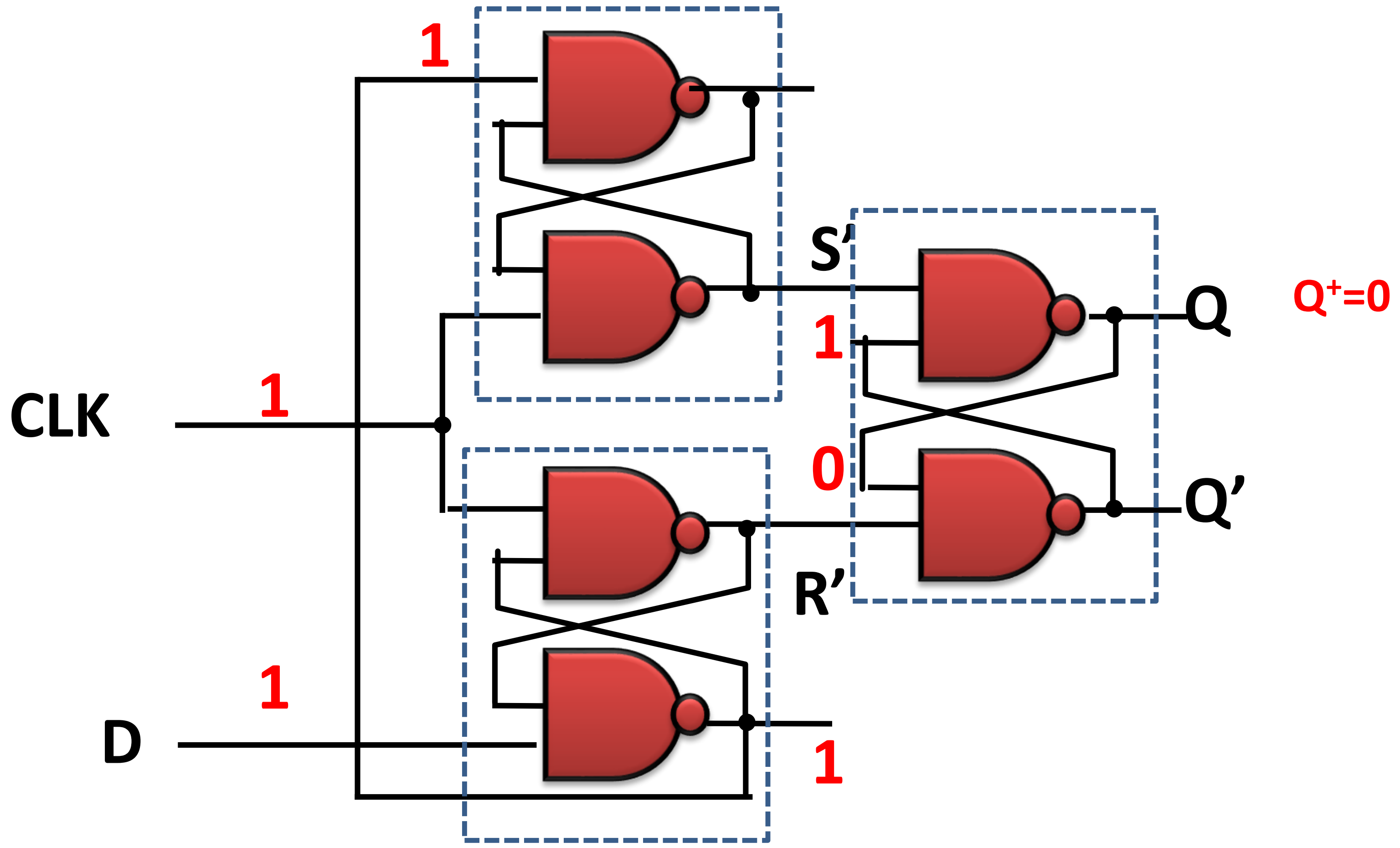
Positive-Edge Triggered D-FF: Economical

- When CLK=1, D=0, Q^+=0

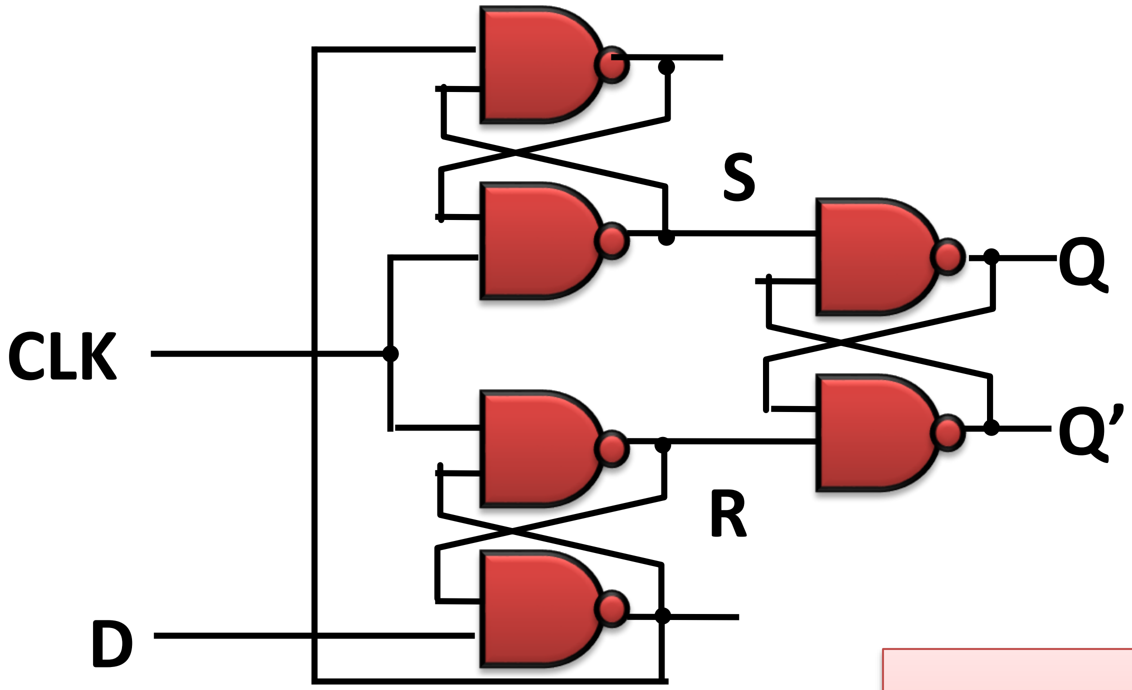


Positive-Edge Triggered D-FF: Economical

- After that When CLK=1, D=1: No changes to R'S': $Q^+=0$ **It locked**



Positive-Egde Trigeered D-FF: Economical



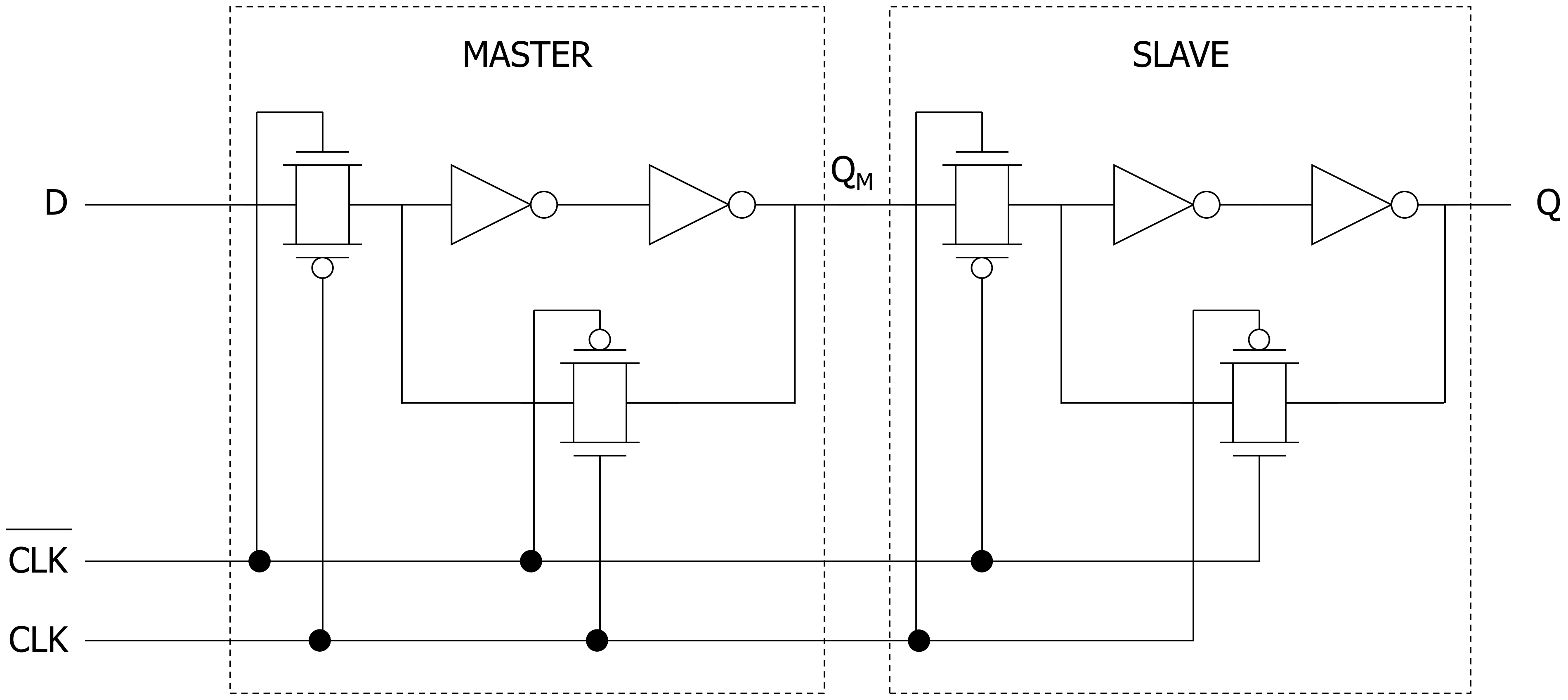
GATE COUNT: 6
6 NAND, Same types

Transistor level optimization
is
out of syllabus

But showing two slides ☺ ☺ ☺

Master-Slave Edge-Triggered Flip-Flop

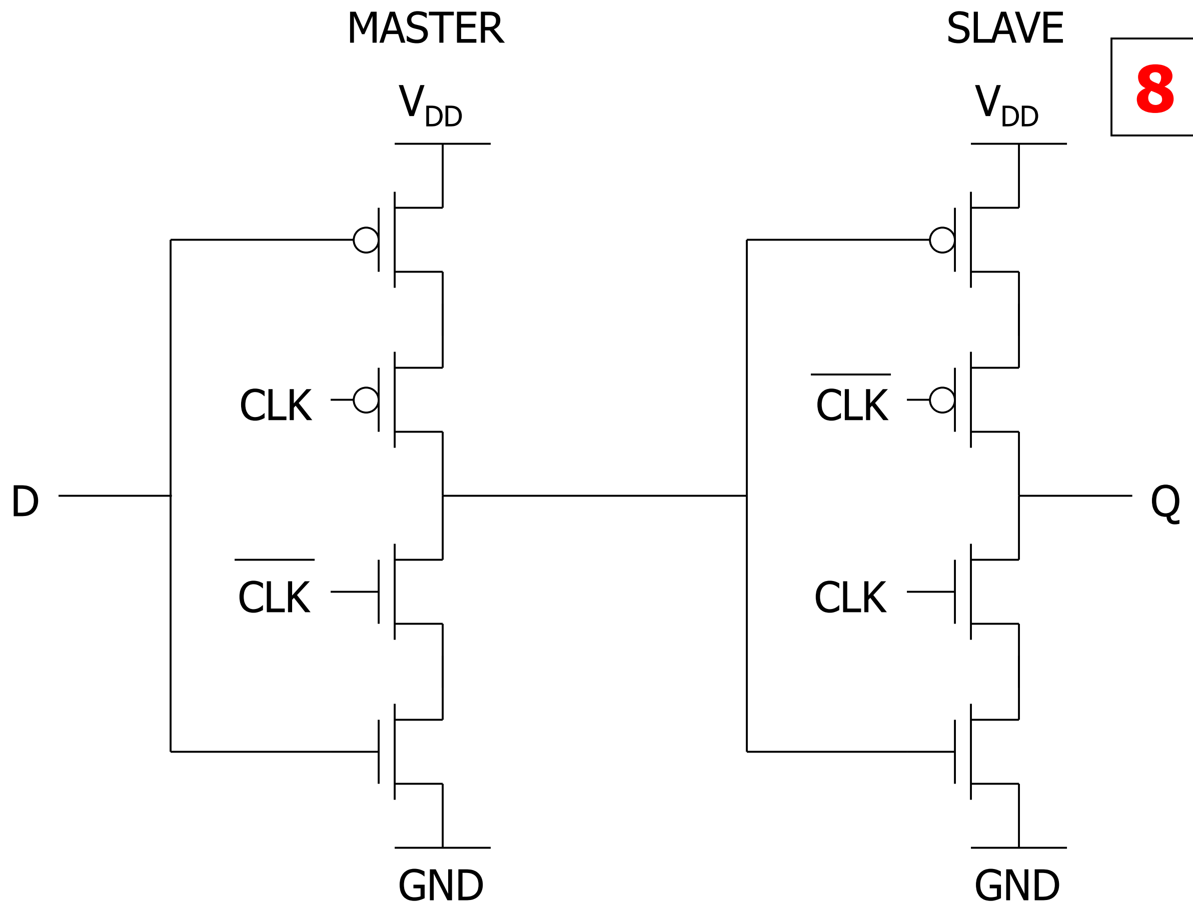
2 x 8 = 16 Transistors



More Efficient Master-Slave Edge-Triggered Flip-Flop

- Called a C^2MOS (Clocked CMOS) design

8 Transistors

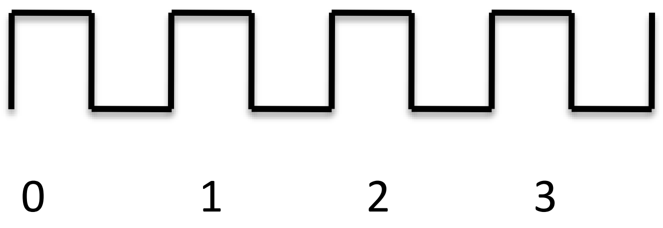


Problem handled in Designing FF

- OR Gate : worked just like a ringing bell
- OR gate with Feed back : (Q=1 can never be changed)
- Two NOR gates with cross coupled out put and input : Solved to store a bit but Race condition
- Ensure RS=11 will not happed by adding Not and AND gate
- Enable Signal to put remove : delay of added Ckt
- Master Slave Latches to make a FF
- Optimized D-FF (using only NAND Gates)

Conventions

- The circuit is *set* means output = 1
- The circuit is *reset* means output = 0
- Flip-flops have two output Q and Q’
- Due to time related characteristic of the flip-flop:
  - Q_t or Q: present state
  - Q_{t+1} or Q^+: next state

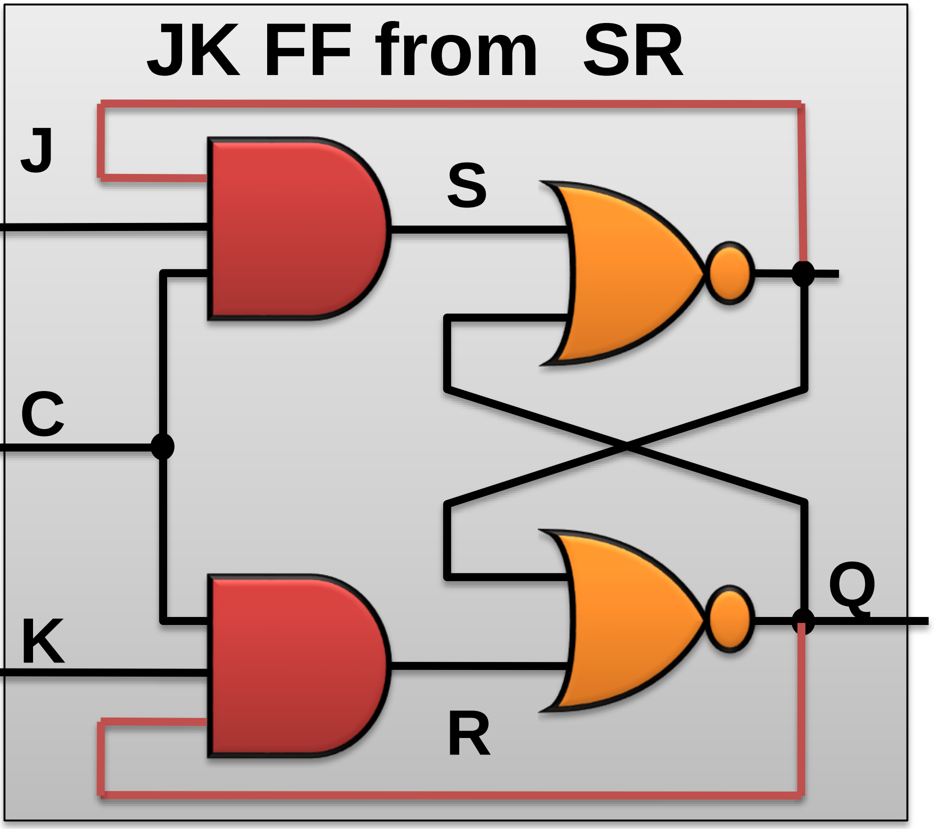


4 Type of Flip Flop

- **SR Flip Flop** : Set/Reset Flip Flop
- **D Flip Flop** : Data Flip Flop to store Bit
- **J-K Flip Flop**: Unavoidable SR=11 state to Toggle (All input values are useful)
  - The JK Flip Flop was named to honour "**Jack Kilby**" of Texas Instrument engineer who invented the concept of IC.
- **T Flip Flop**: Toggle Flip Flop

J-K Latches

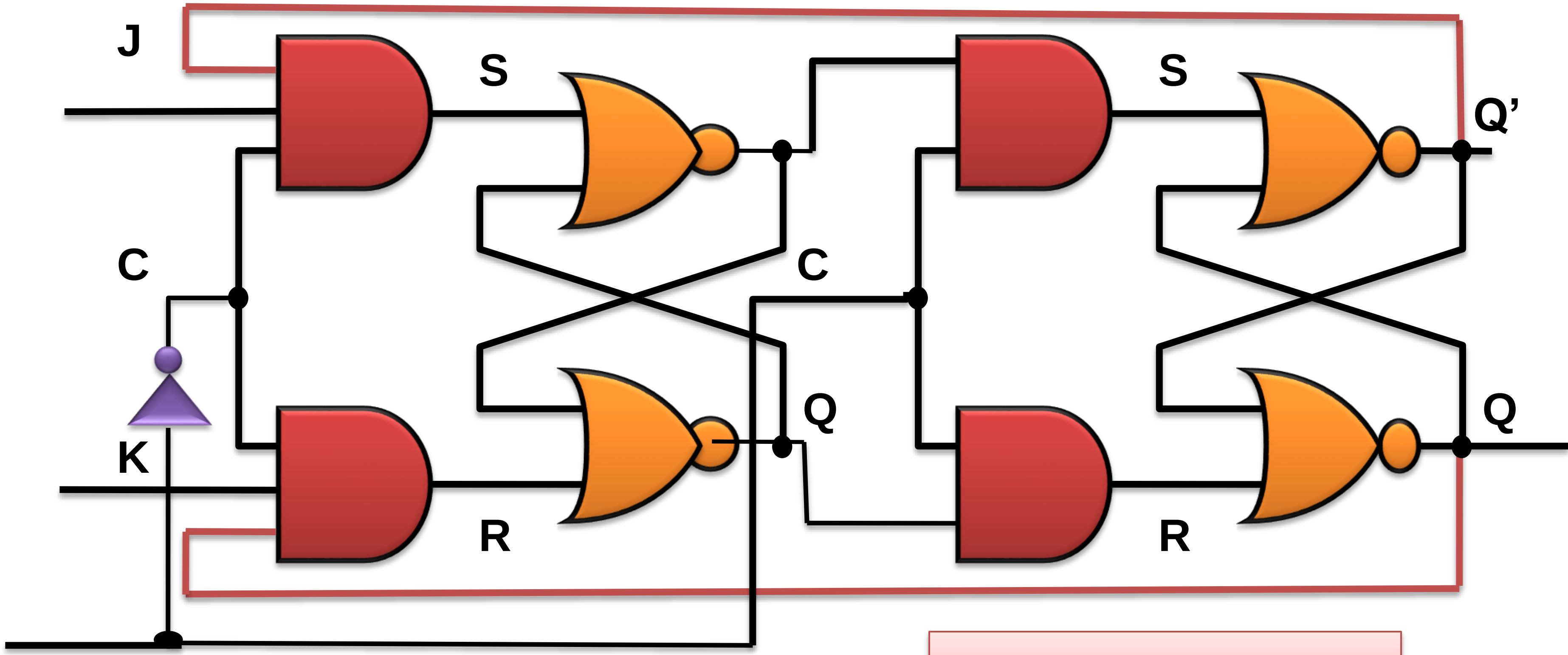
- The JK flip-flop augments the behavior of the SR flip-flop (J=Set, K=Reset) by interpreting the S = R = 1 condition as a "flip" or toggle command.



J	K	Q+
0	**0**	**Qt**
0	1	0
1	0	1
1	**1**	**Qt’**

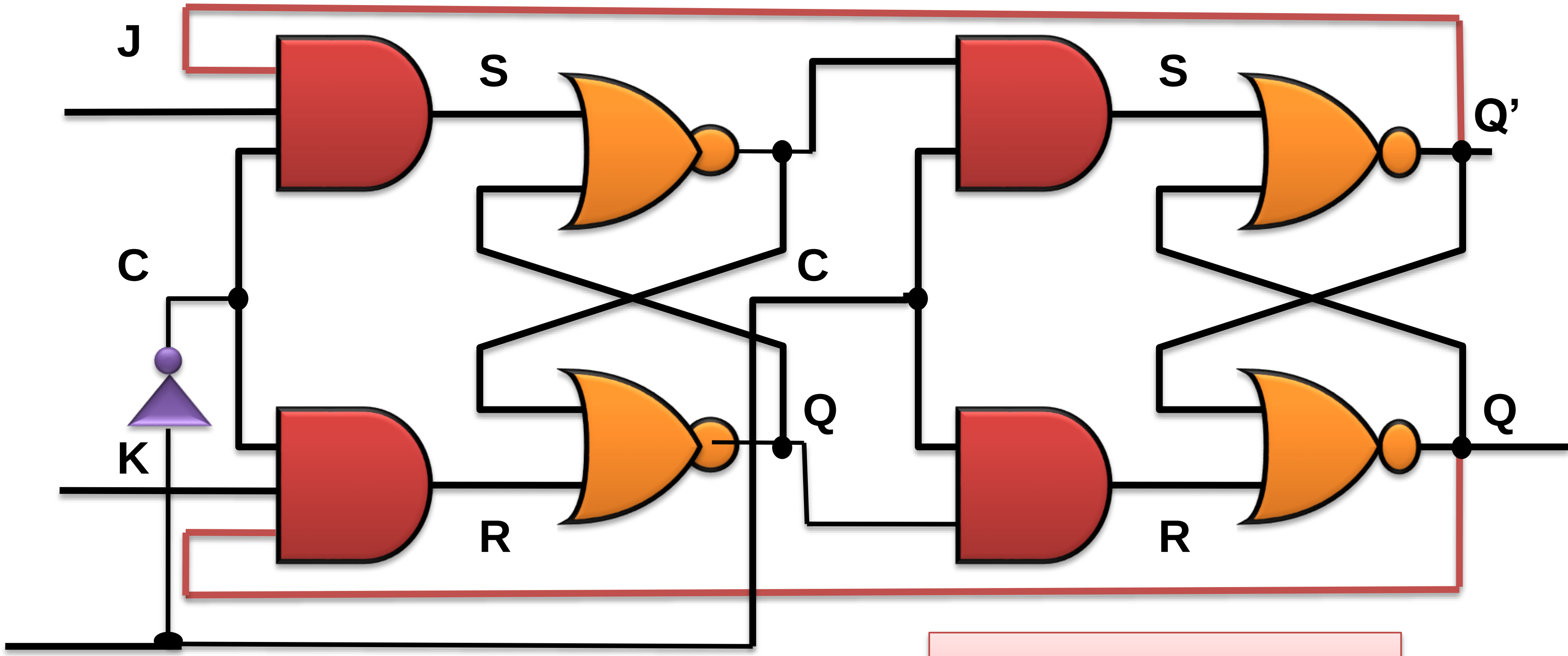
$$Q^{+} = K'Q + JQ'$$

Master Slave J-K Flip Flop



$Q^{+} = K'Q + JQ'$

Master Slave J-K Flip Flop



$Q^{+} = K'Q + JQ'$

J-K Flip Flop

- To synthesize a D flip-flop, simply set K equal to the complement of J.
- The JK flip-flop is a universal flip-flop
  - Because it can be configured to work as any FF
  - T flip-flop or D flip-flop or SR flip-flop.

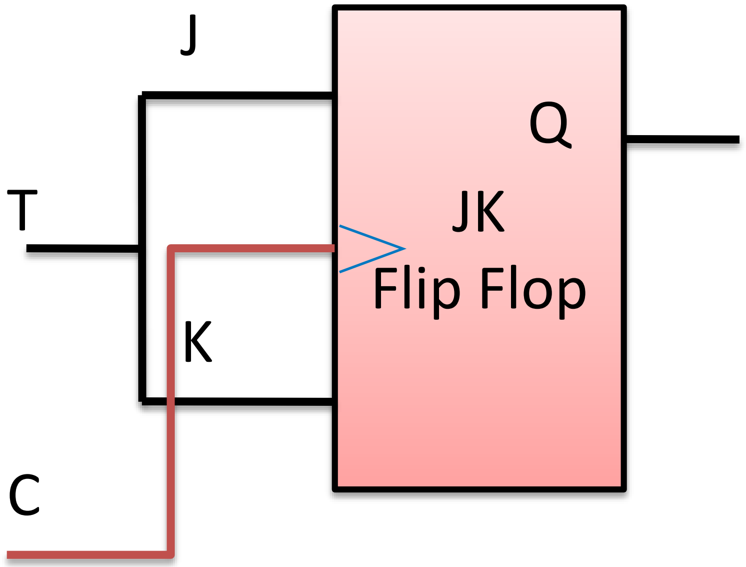
J=T	K=T	Q+
0	**0**	**Qt**
0	1	0
1	0	1
1	**1**	**Qt’**

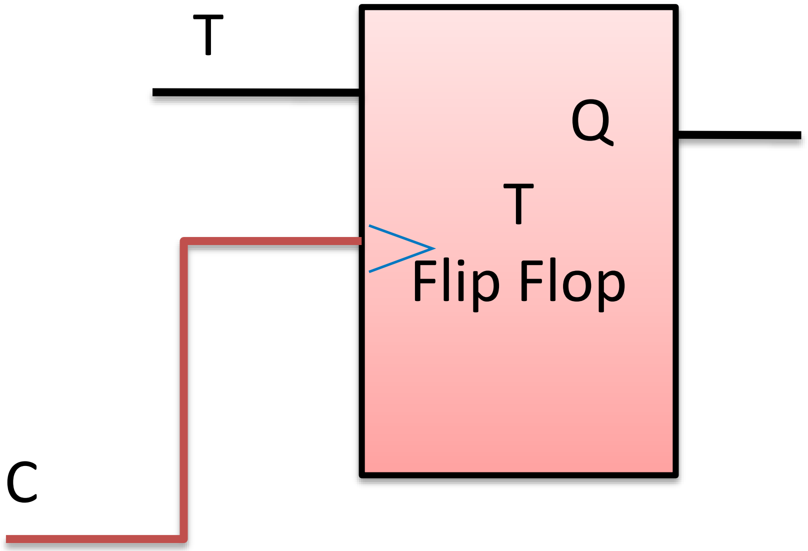
J=D	K=D’	Q+
0	0	Qt
0	**1**	**0**
1	**0**	**1**
1	1	Qt’

J=S	K=R	Q+
0	**0**	**Qt**
0	**1**	**0**
1	**0**	**1**
1	1	Qt’

Toggle Flip-Flop: T FF

- J=K=1, Q^+= Q'





4 Types of Flip-Flops

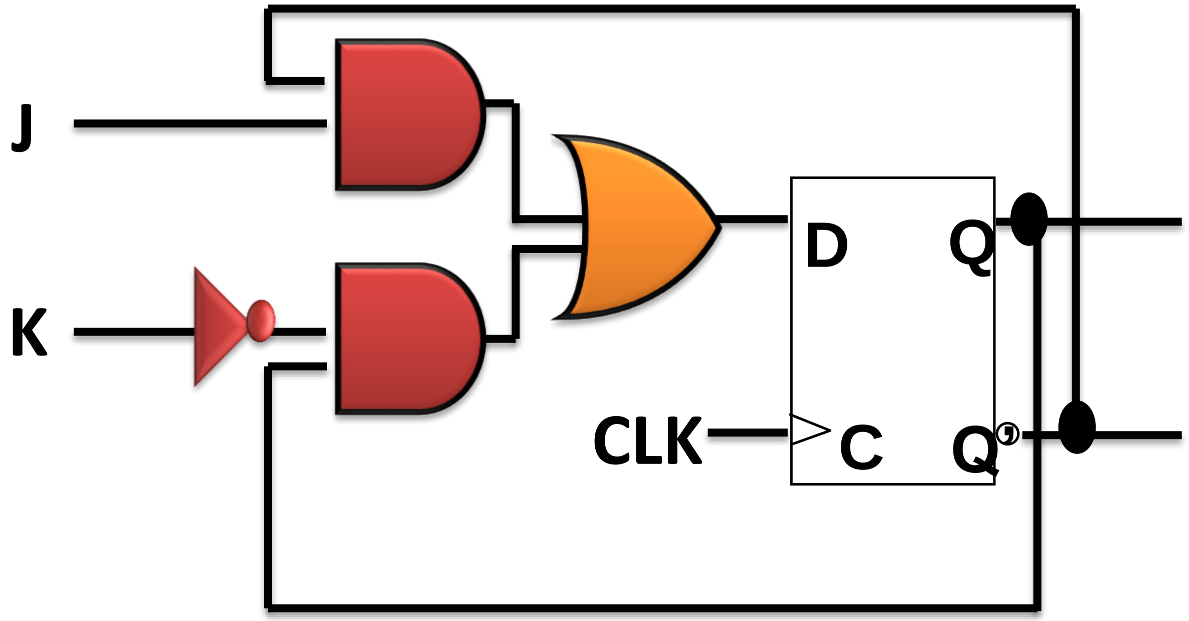
S	R	Q+
0	0	Qt
0	**1**	**0**
1	**0**	**1**
1	1	U

J	K	Q+
0	**0**	**Qt**
0	1	0
1	0	1
1	**1**	**Qt’**

D	Q+
0	0
1	1

T	Q+
0	Qt
1	Qt’

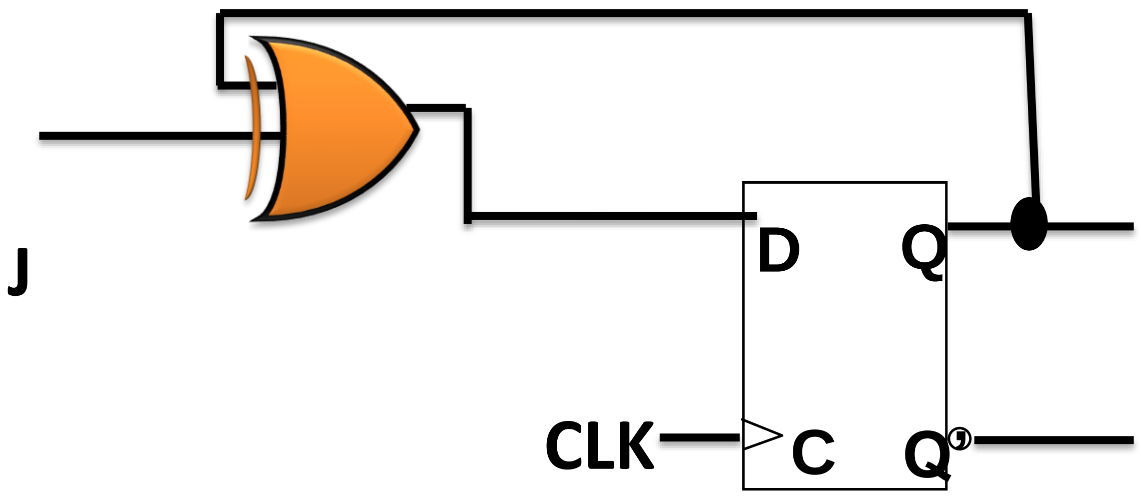
Given a D FF: Construct JK FF



J	K	D	K'Q+JQ'
0	0	Q	1Q+0Q'
0	1	0	0Q+0Q'
1	0	1	1Q+1Q'
1	1	Q'	0Q+1Q'

D= K'Q + JQ'

Given a D FF: Construct T FF



T	D	TQ'+T'Q
0	Q	0Q'+1Q
1	Q'	1Q'+0Q

D= TQ' + TQ

Characteristic Equations

- A descriptions of the next-state table of a flip-flop
- Constructing from the Karnaugh map for Q_{t+1} in terms of the present state and input

Characteristic tables

- The tables that we've made so far are called characteristic tables.
  - They show the next state Q(t+1) in terms of the current state Q(t) and the inputs.
  - For simplicity, the control input C is not usually listed.
  - Again, these tables don't indicate the positive edge-triggered behavior of the flip-flops that we'll be using.

J	K	Q+
0	**0**	**Qt**
0	1	0
1	0	1
1	**1**	**Qt'**

D	Q+
0	0
1	1

T	Q+
0	Qt
1	Qt'

Characteristic equations

- We can also write characteristic equations, where the next state Q(t+1) is defined in terms of the current state Q(t) and inputs.

J	K	Q+
0	**0**	**Qt**
0	1	0
1	0	1
1	**1**	**Qt'**

$$\begin{aligned} Q^+ &= J'K'Q + JKQt' + JK' \\ &= J'K'Q + JKQ' + JK'Q' + JK'Q \\ &= J'K'Q + JK'Q + JKQ' + JK'Q' \\ &= (J+J')K'Q + J(K+K')Q' \end{aligned}$$

$$\mathbf{Q^+ = K'Q + JQ'}$$

$$Q(t+1) = K'Q(t) + JQ'(t)$$

Characteristic equations

- We can also write characteristic equations, where the next state Q(t+1) is defined in terms of the current state Q(t) and inputs.

D	Q+
0	0
1	1

$Q^+ = D$

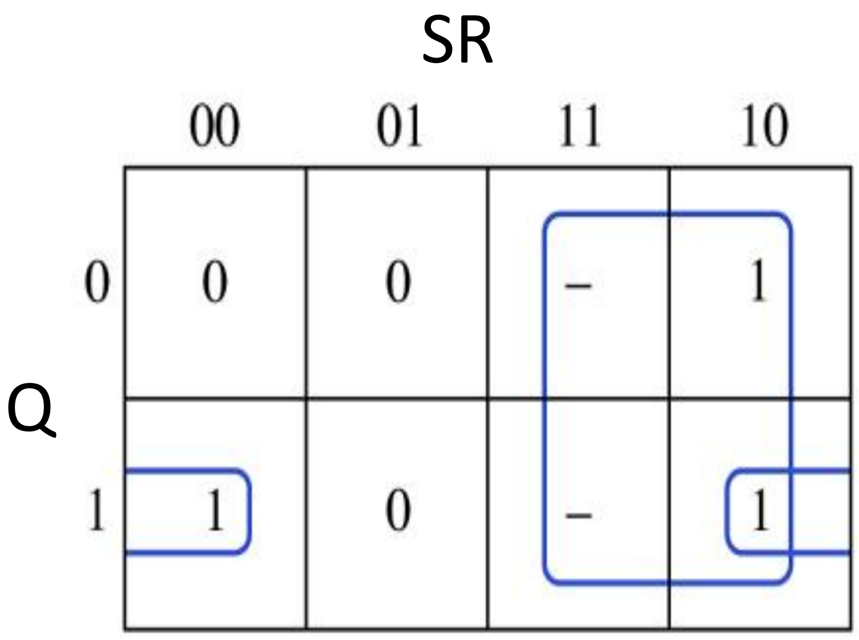
$Q(t+1) = D$

T	Q+
0	Qt
1	Qt’

$Q^+ = T'Q + TQ' = T \oplus Q$

$Q(t+1) = T'Q(t) + TQ'(t) = T \oplus Q(t)$

Characteristic equations



$$Q^{+} = S + R'Q \quad (SR=0)$$

Flip Flop Type	Characteristic Equation
SR	$Q^{+} = S + R'Q \quad (SR=0)$
JK	$Q^{+} = JQ' + K'Q$
D	$Q^{+} = D$
T	$Q^{+} = TQ' + T'Q = T \oplus Q$

FF with Asynchronous Inputs

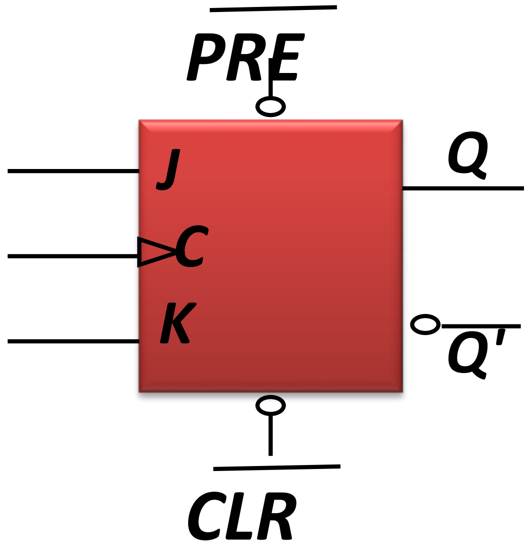
- S-R, D and J-K inputs are synchronous inputs
  - As data on these inputs are transferred to the flip-flop's output
  - Only on the triggered edge of the clock pulse.
- Asynchronous inputs affect the state of the flip-flop independent of the clock
- Example:
  - *Preset* (*PRE*) and *clear* (*CLR*)
  - or *direct set* (*SD*) and *direct reset* (*RD*)

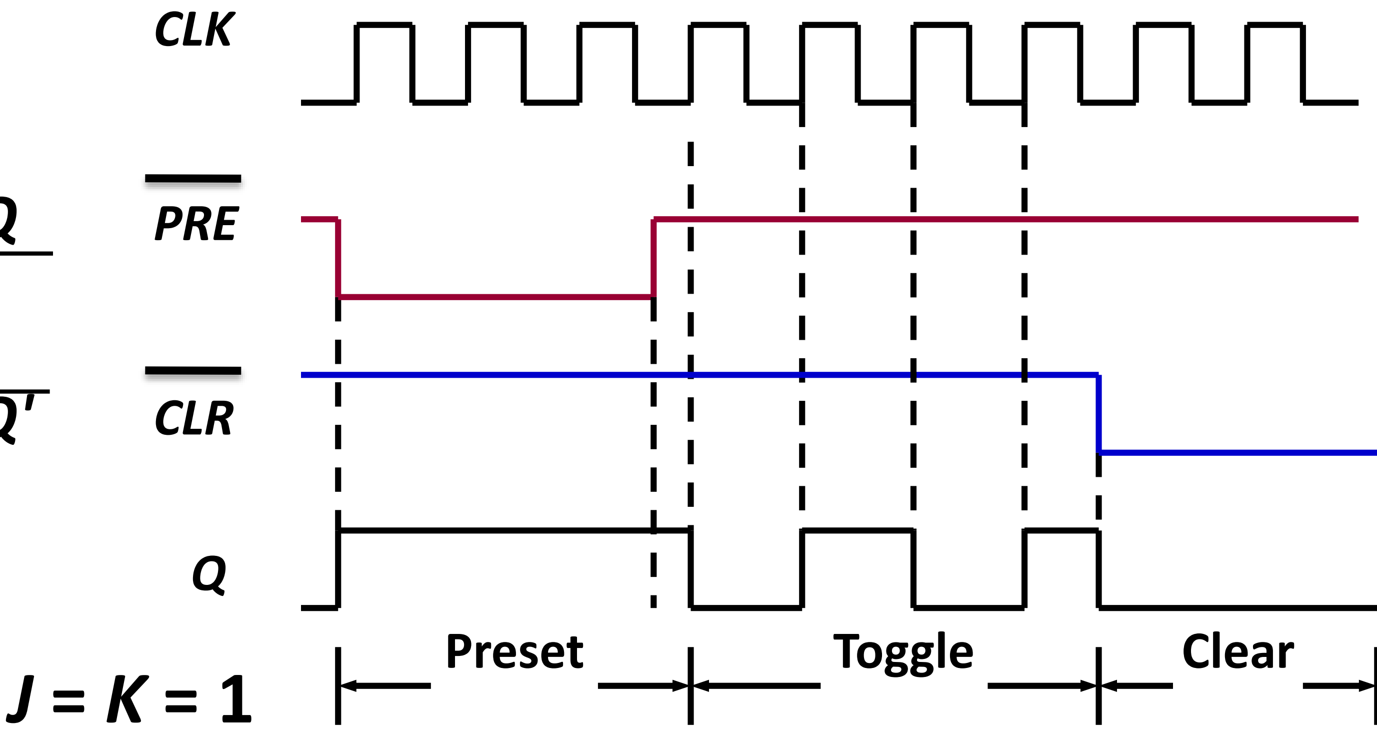
FF with Asynchronous Inputs

- When *PRE*=HIGH, *Q* is immediately set to HIGH.
- When *CLR*=HIGH, *Q* is immediately cleared to LOW.
- Flip-flop in normal operation mode when both *PRE* and *CLR* are LOW.

Asynchronous Inputs

- A J-K flip-flop with active-LOW preset and clear inputs.





Thanks