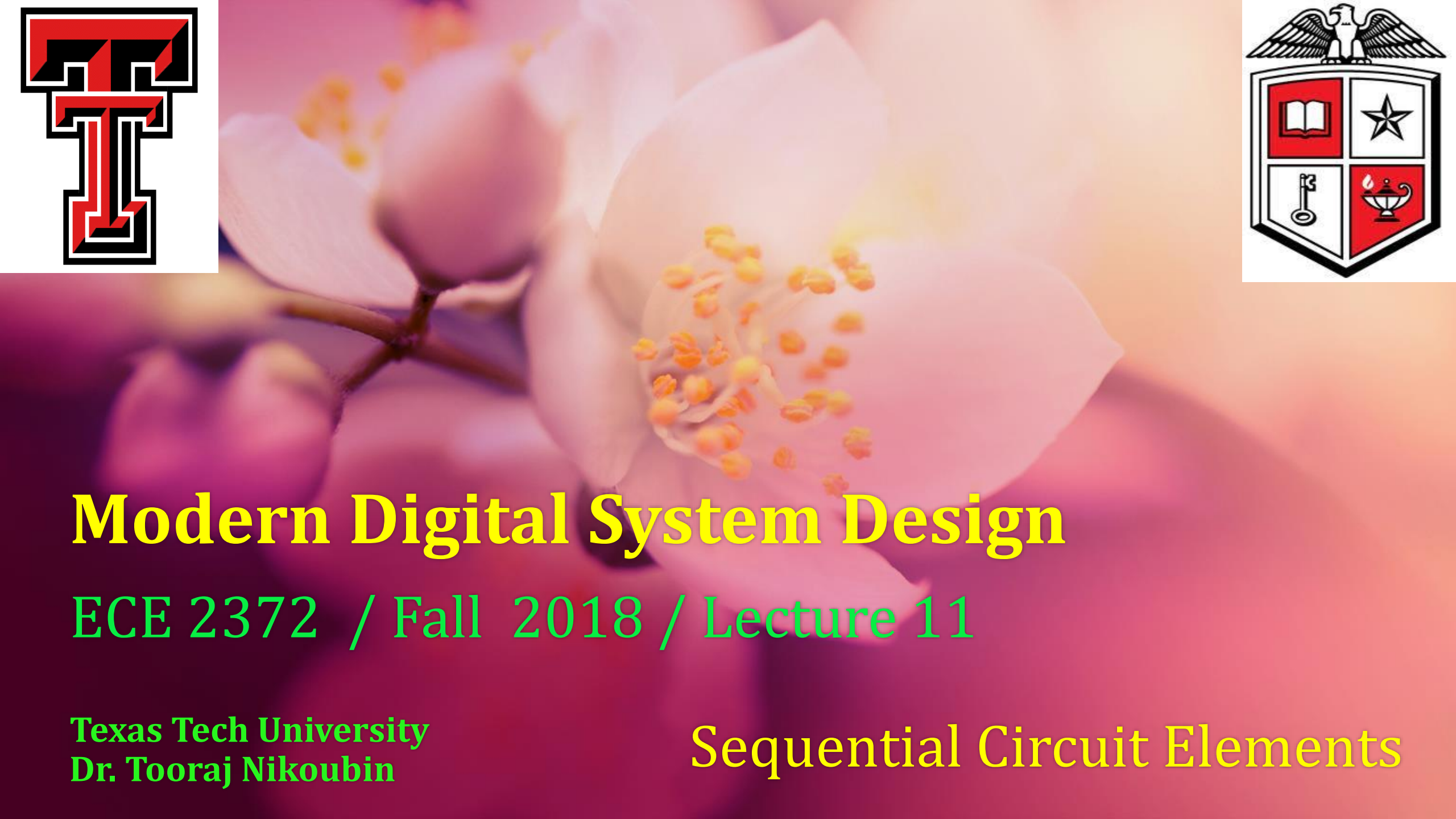
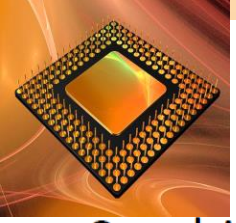


Modern Digital System Design

ECE 2372 / Fall 2018 / Lecture 11

Texas Tech University
Dr. Tooraj Nikoubin

Sequential Circuit Elements



Outline

- Combinational Vs Sequential Logic Design
- Design a new building block, a **flip-flop**, that stores one bit
- Combine that block to build multi-bit storage – a **register**
- Describe the sequential behavior using a **finite state machine (FSM)**
- Convert a finite state machine to a **controller** – a sequential circuit having a register and combinational logic

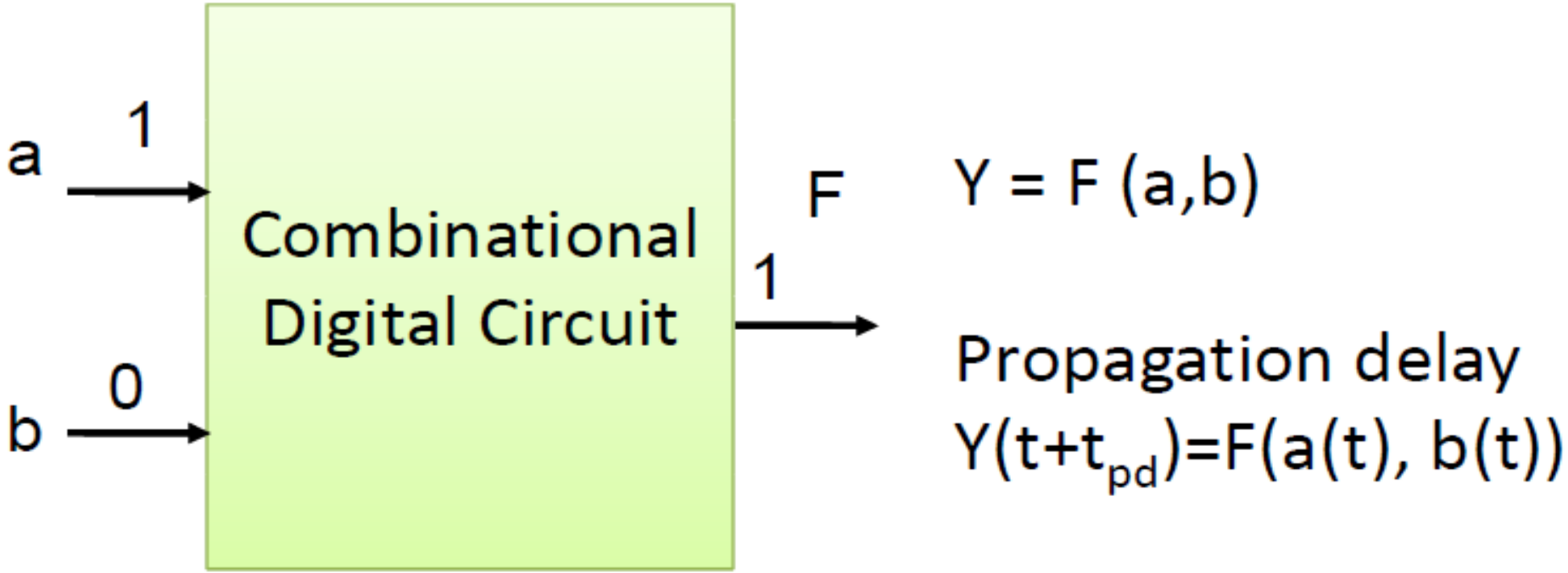
- Design a new building block, a **flip-flop**, that stores one bit

- Latch Vs Flip Flop
- Master Slave Flip-Flop
- D Flip-Flop, J-K Flip-Flop and T Flip-Flop

- Combine that block to build multi-bit storage – a **register**

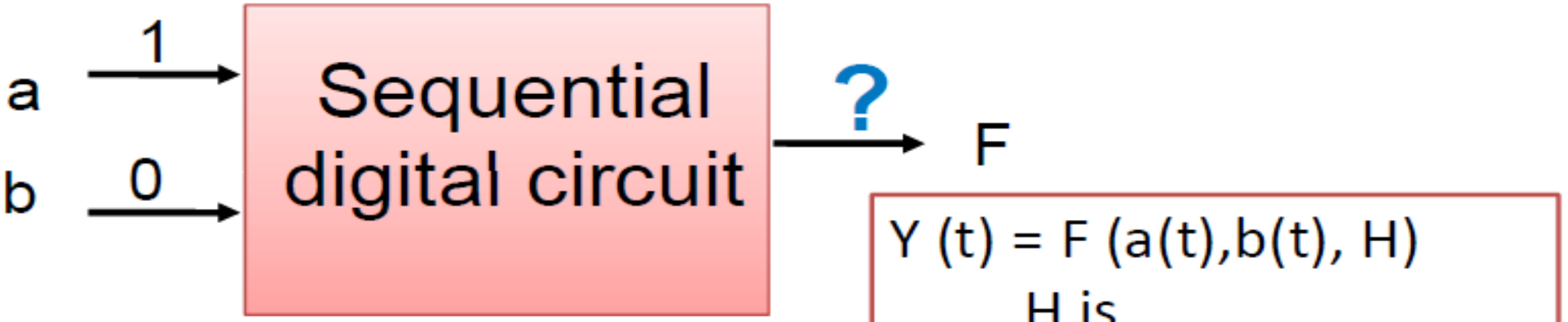
Combinational Vs Sequential Logic

- Combinational circuit
  - **Output depends on present input**
  - Examples: F (A,B,C), FA, HA, Multiplier, Decoder, Multiplexor, Adder, Priority Encoder



Combinational Vs Sequential Logic

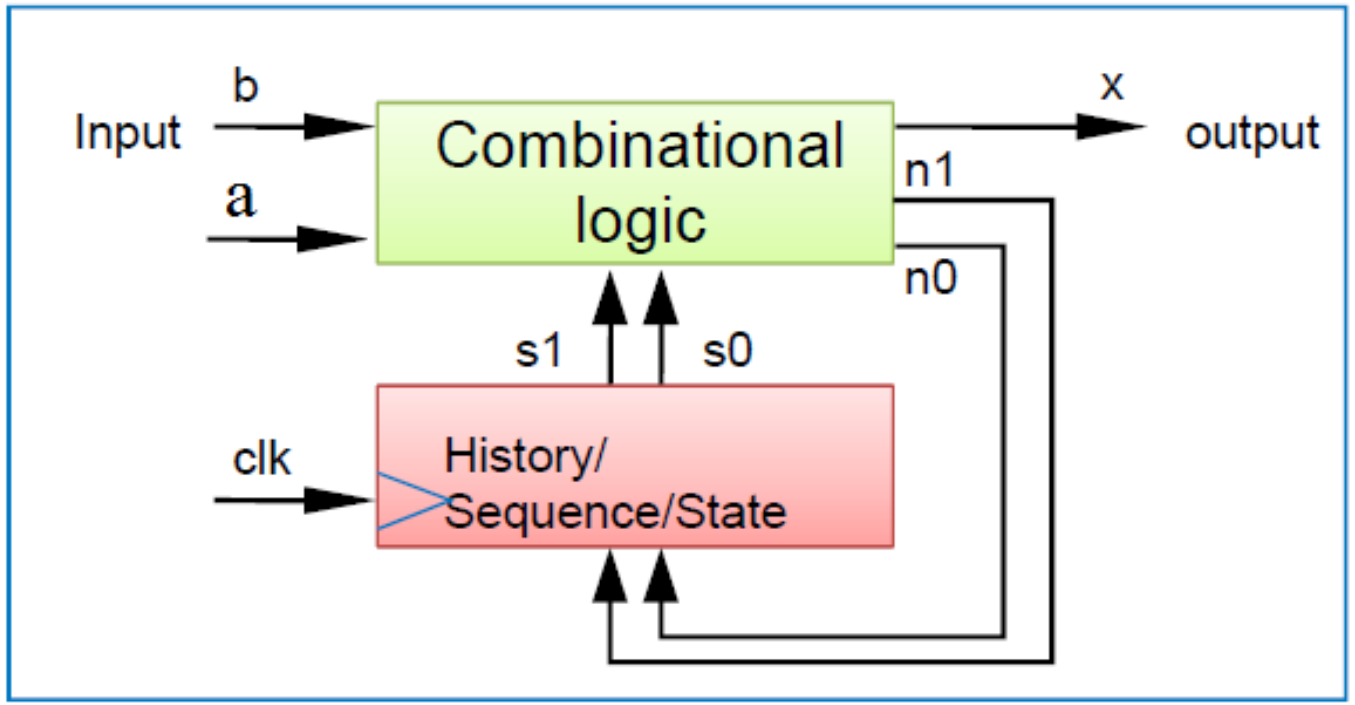
- Sequential circuit
  - Output depends not just on present inputs
  - But also on past sequence of inputs (State)
    - Stores bits, also known as having "state"
- Simple example: a circuit that counts up in binary



Must know sequence of past inputs to know output

Y (t) = F (a(t),b(t), H)
H is History/Sequence/State

Sequential Circuit



Where to Store this History

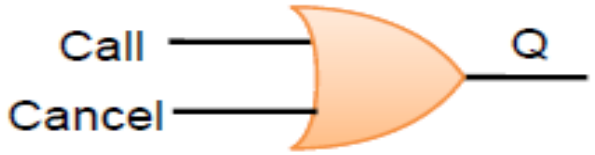
(Memory Element)

Y (t) = F (a(t),b(t), H)

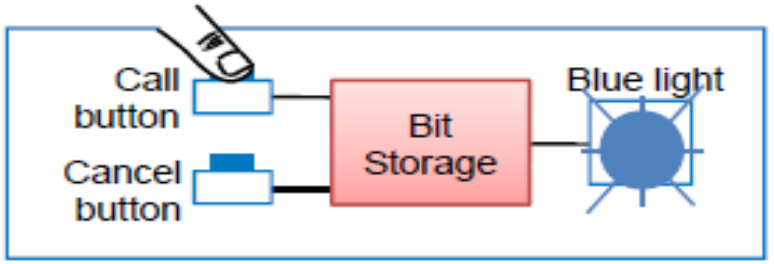
H is History/Sequence/State

Example Needing Bit Storage

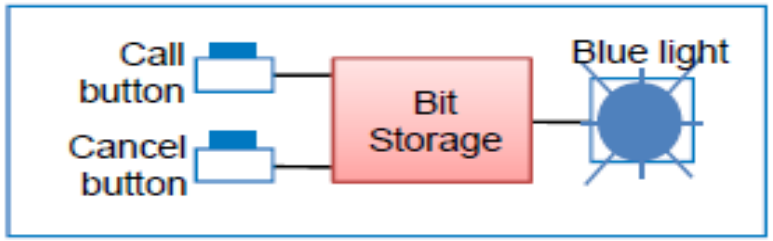
- Flight attendant call button
  - Press call: light turns on
    - ***Stays on*** after button released
  - Press cancel: light turns off
  - Logic gate circuit to implement this?



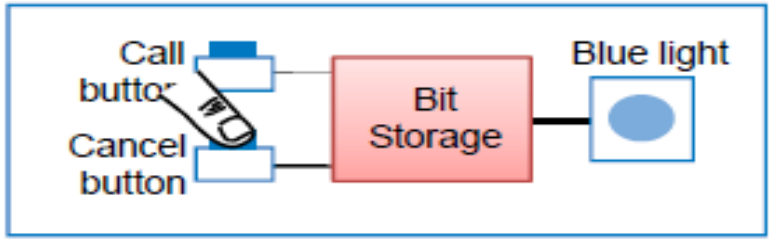
Doesn't work. Q=1 when Call=1, but doesn't stay 1 when Call returns to 0



1. Call button pressed – light turns on



*2. Call button released – light **stays on***

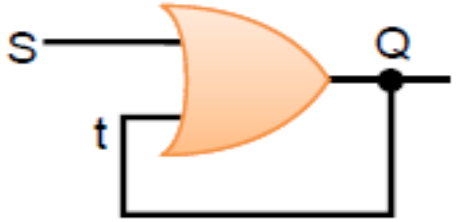


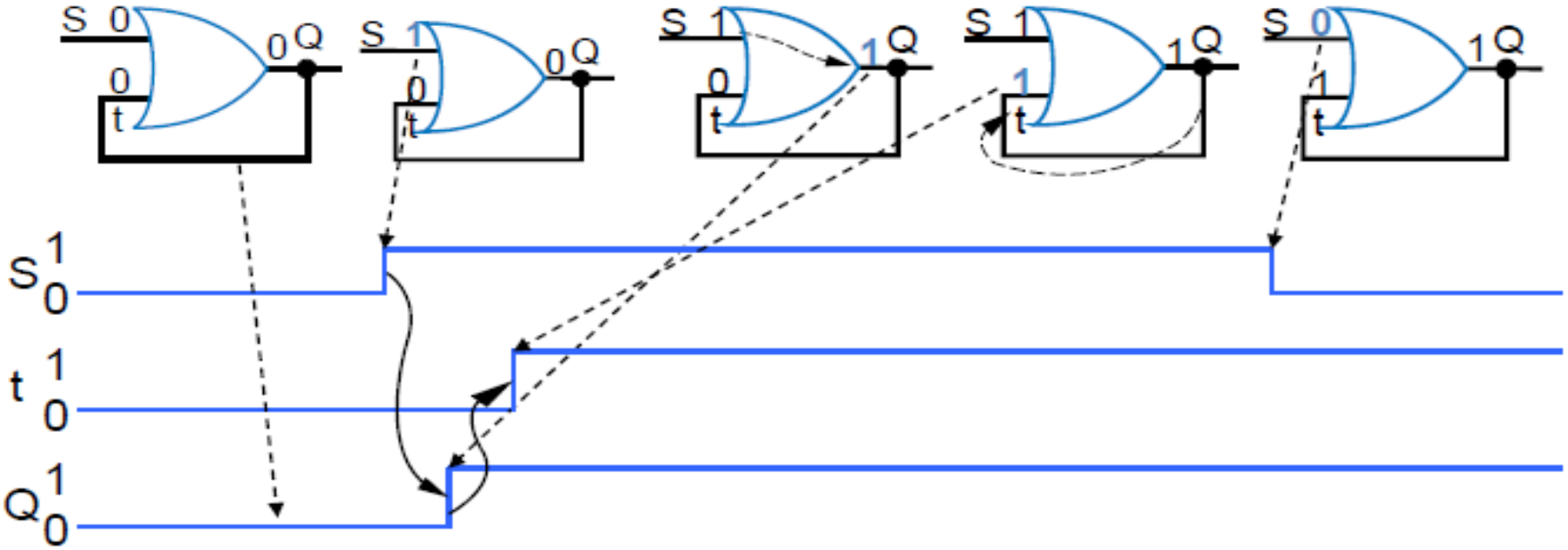
3. Cancel button pressed – light turns off

Need some form of "feedback" in the circuit

First attempt at Bit Storage

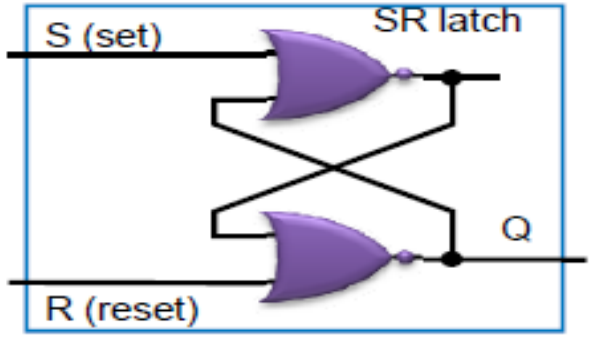
- We need some sort of feedback
  - Does circuit on the right do what we want?
    - No: Once Q becomes 1 (when S=1), Q stays 1 forever – no value of S can bring Q back to 0

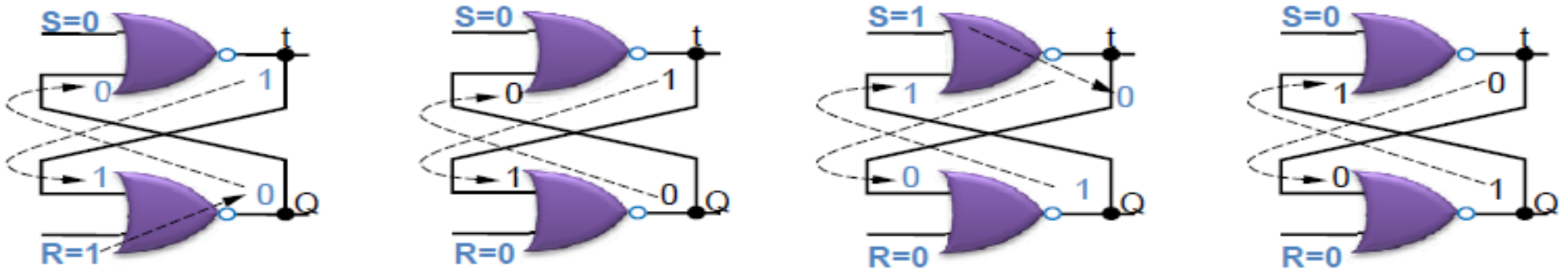


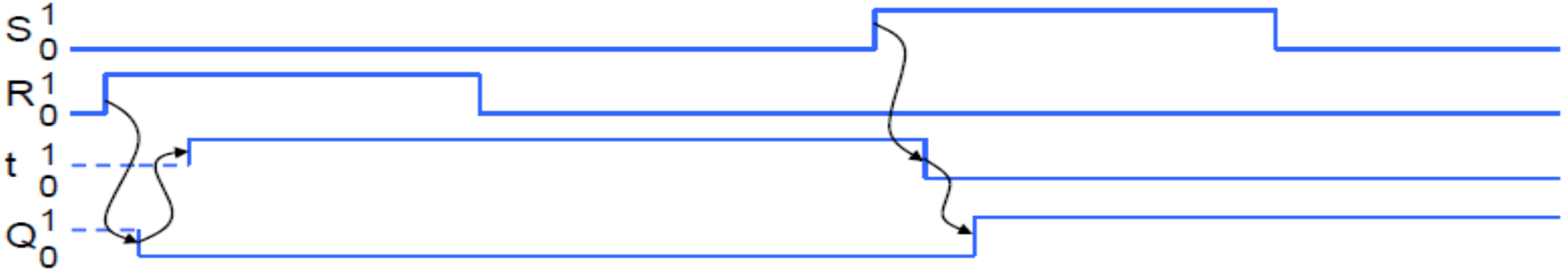


Bit Storage Using an SR Latch

- Does the circuit to the right, with cross-coupled NOR gates, do what we want?
  - Yes! How did someone come up with that circuit? Maybe just trial and error, a bit of insight...



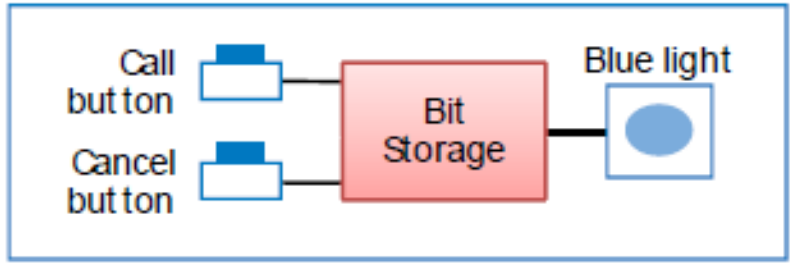


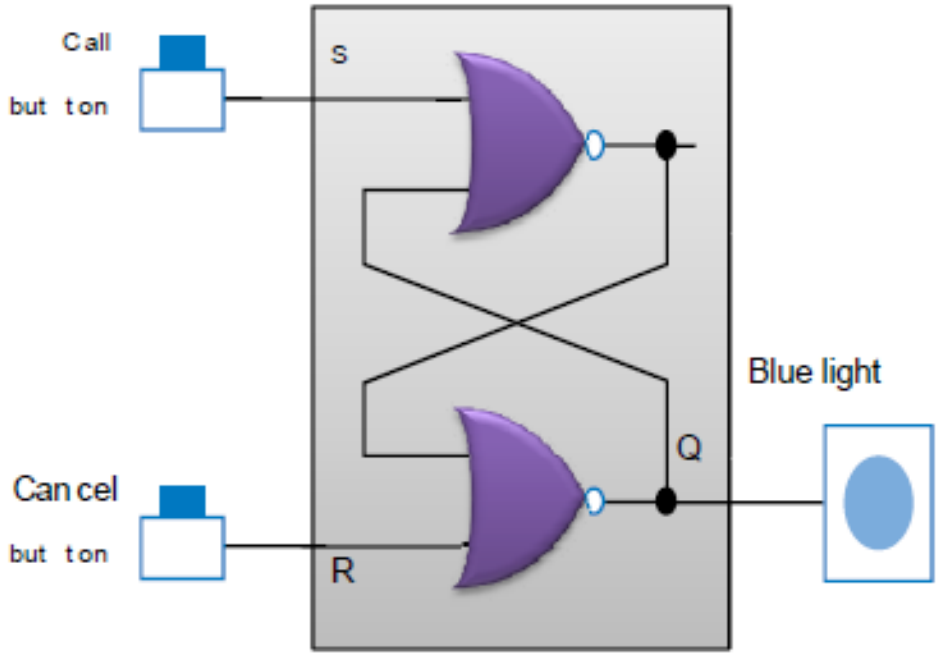


Example Using SR Latch for Bit Storage

- SR latch can serve as bit storage in previous example of flight-attendant call button
  - Call=1 : sets Q to 1
    - Q stays 1 even after Call=0
  - Cancel=1 : resets Q to 0

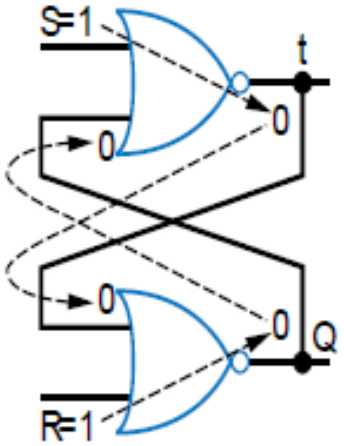
- But, there's a problem…

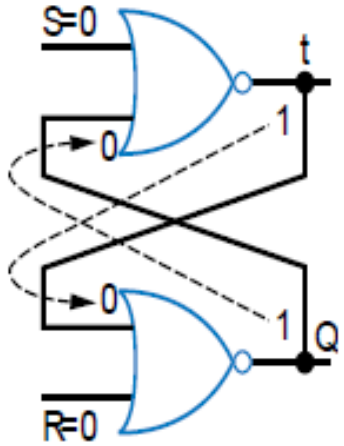


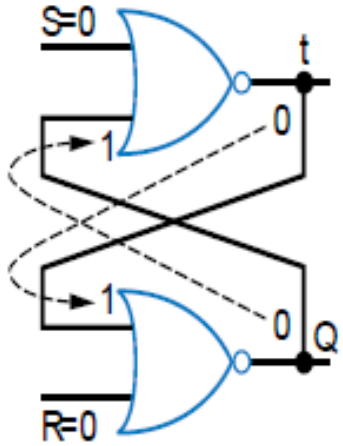


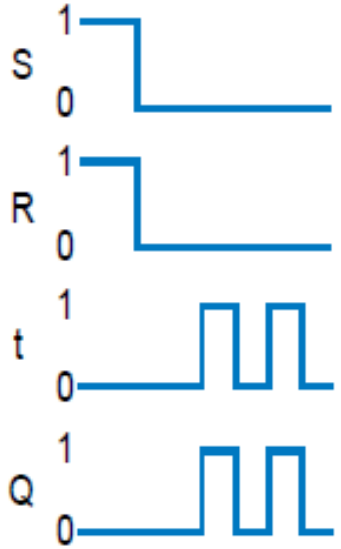
Problem with SR Latch

- Problem
  - If S=1 and R=1 simultaneously, we don't know what value Q will take

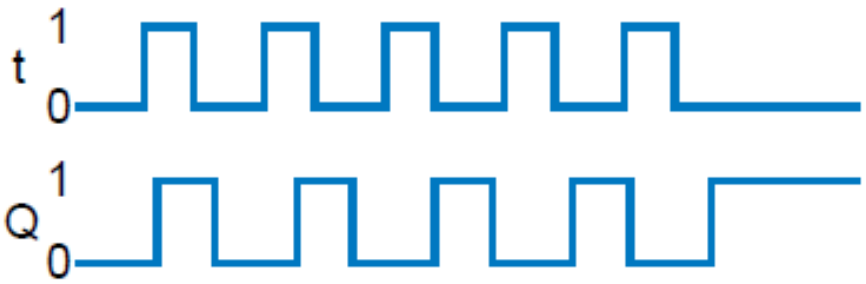






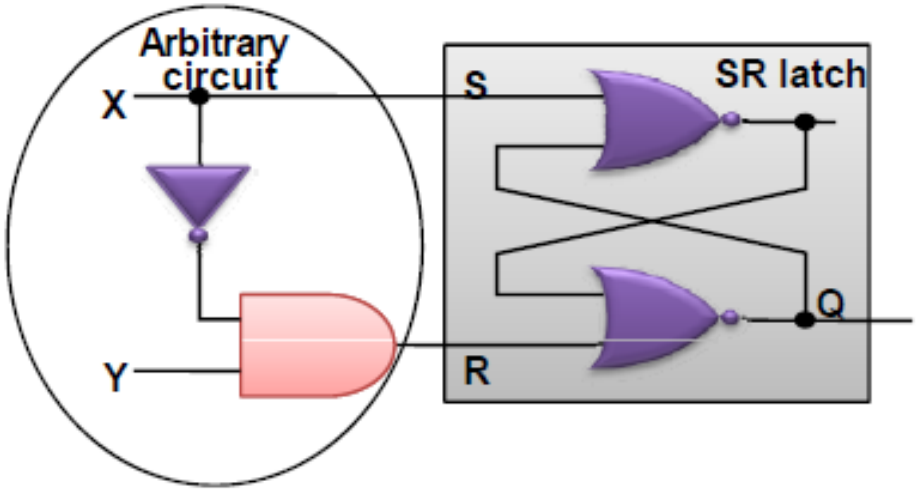


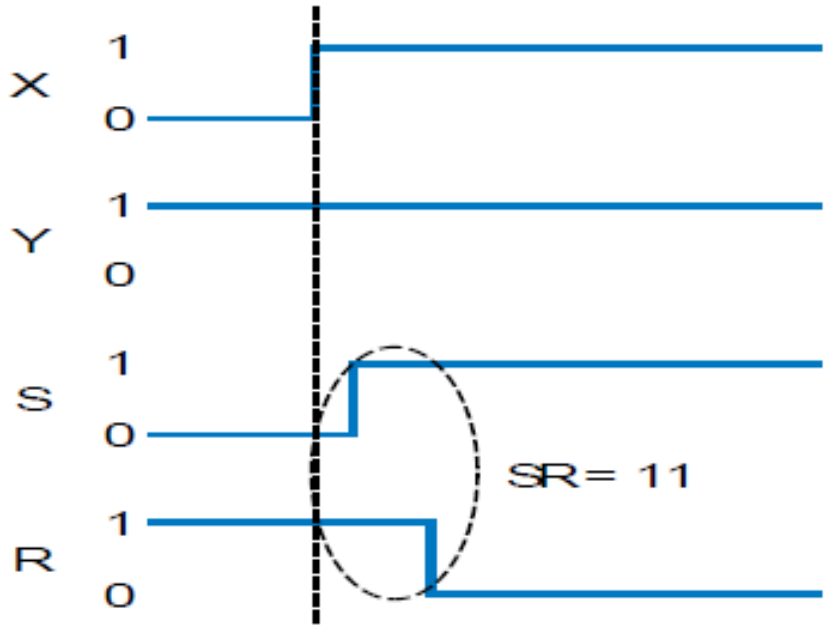
Q may oscillate. Then, because one path will be slightly longer than the other, Q will eventually settle to 1 or 0 – but we don't know which.



Problem with SR Latch

- Problem not just one of a user pressing two buttons at same time
- Can also occur even if SR inputs come from a circuit that supposedly never sets S=1 and R=1 at same time
  - But does, due to different delays of different paths

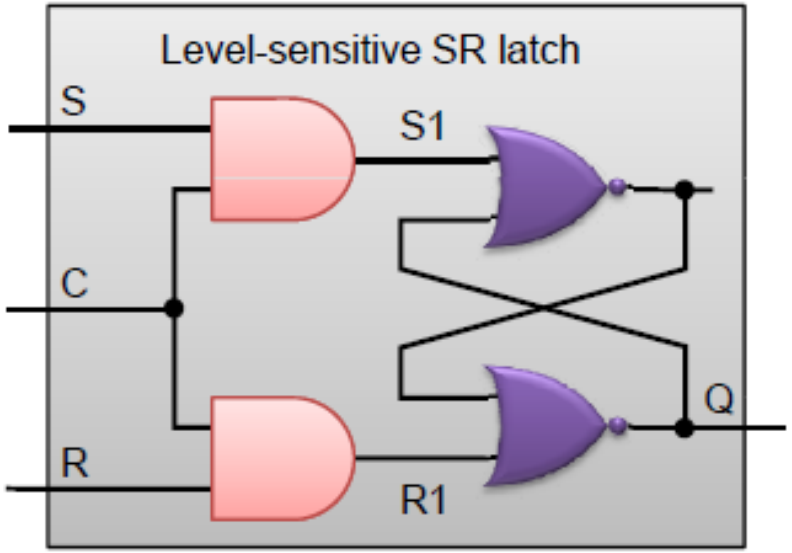


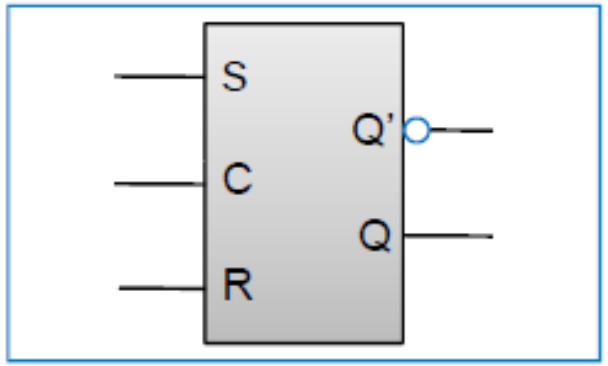


The longer path from X to R than to S causes SR=11 for short time – could be long enough to cause oscillation

Solution: Level-Sensitive SR Latch

- Add enable input "C" as shown
  - Only let S and R change when C=0
    - Enure circuit in front of SR never sets SR=11, except briefly due to path delays
  - Change C to 1 only after sufficient time for S and R to be stable
  - When C becomes 1, the stable S and R value passes through the two AND gates to the SR latch's S1 R1 inputs.

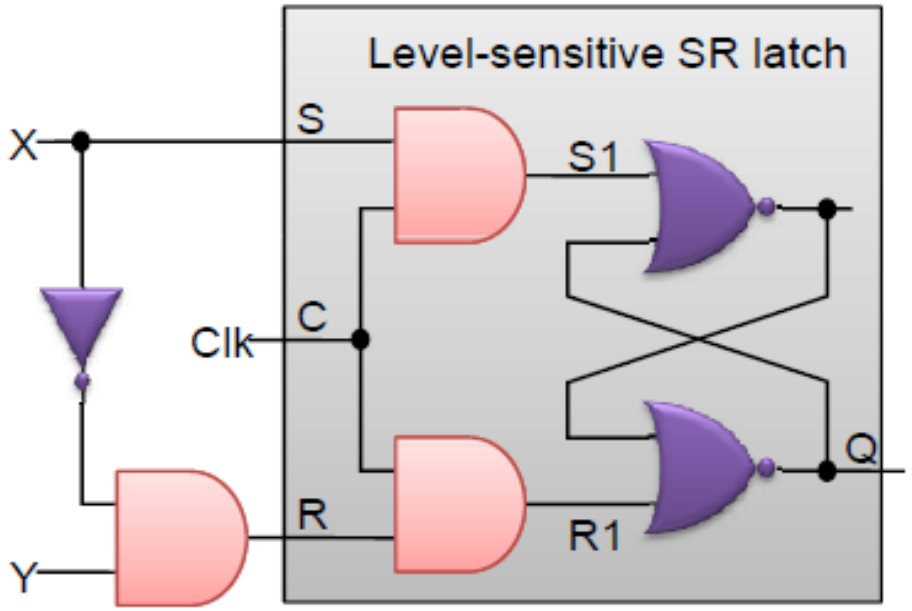


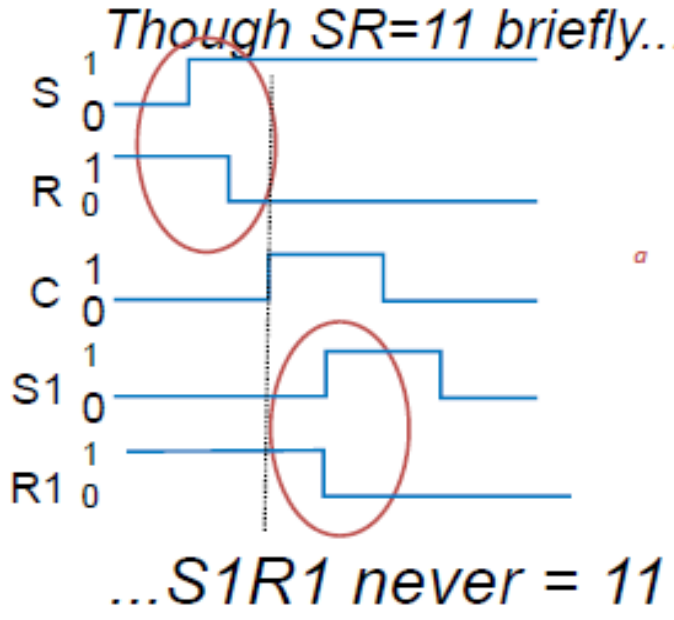


Level-sensitive
SR latch symbol

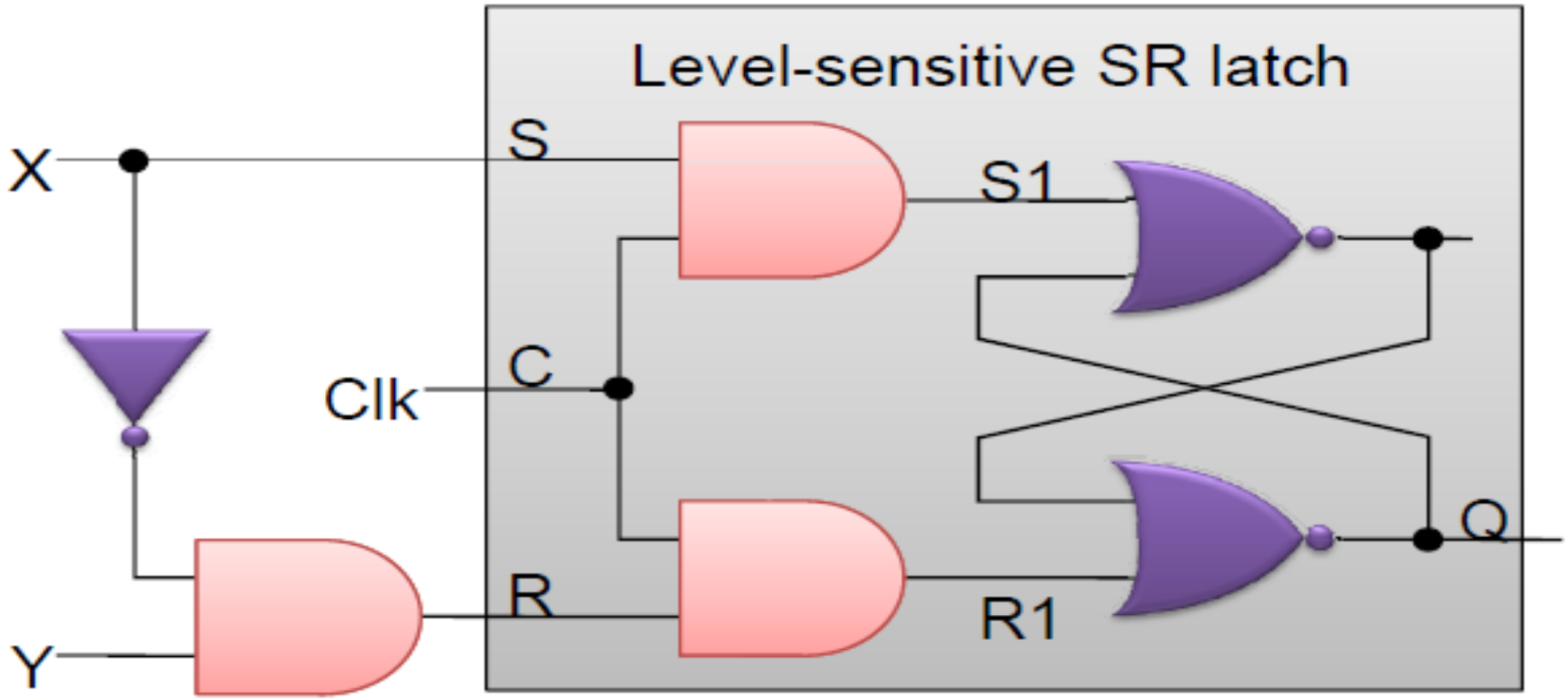
Solution: Level-Sensitive SR Latch

- Add enable input "C" as shown
  - Only let S and R change when C=0
    - Enure circuit in front of SR never sets SR=11, except briefly due to path delays
  - Change C to 1 only after sufficient time for S and R to be stable
  - When C becomes 1, the stable S and R value passes through the two AND gates to the SR latch's S1 R1 inputs.





Solution: Ensure, Stabilize, Store



Ensure Stage

Never Happens
SR=11

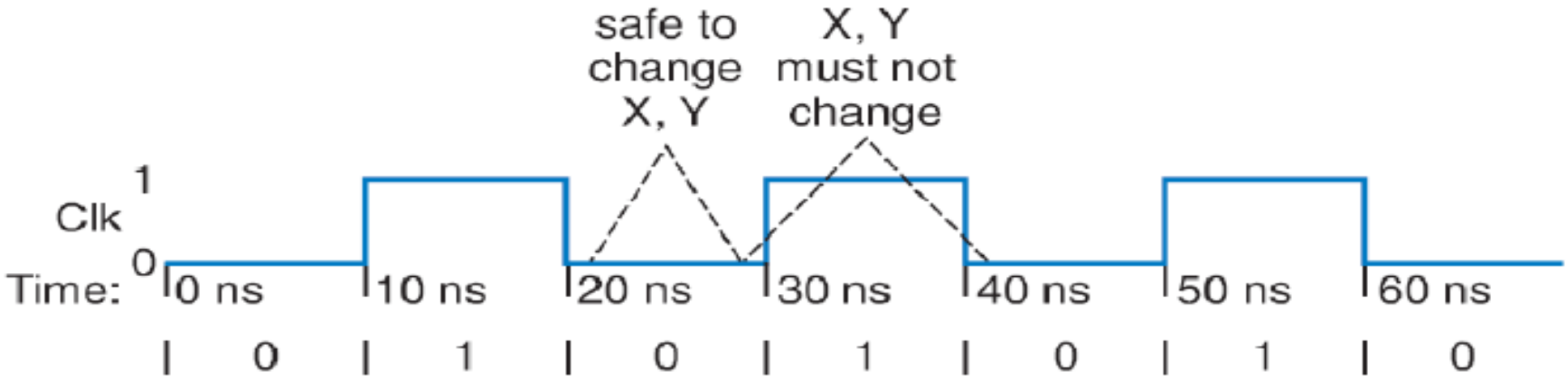
Stabilize Stage

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

Store Stage

Store bit

Clocks

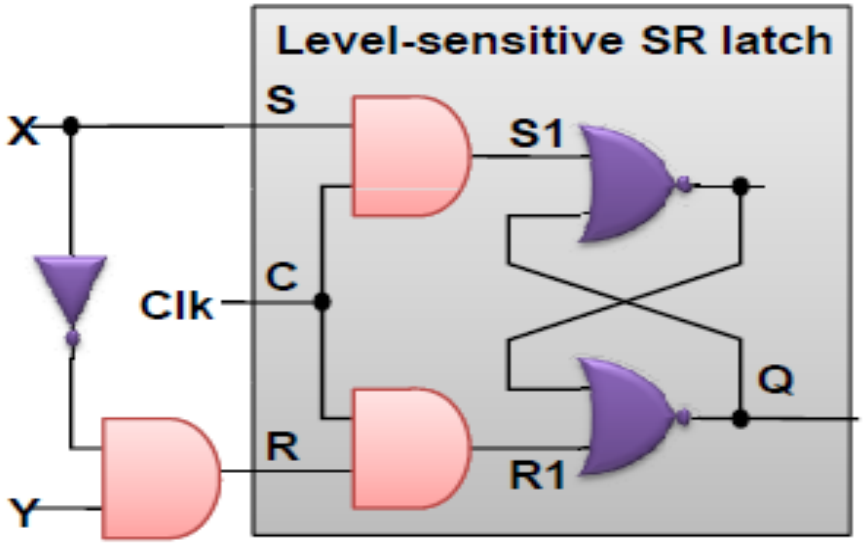


- ***Clock period***: time interval between pulses
  - Above signal: period = 20 ns
- ***Clock cycle***: one such time interval
  - Above signal shows 3.5 clock cycles
- ***Clock frequency***: 1/period
  - Above signal: frequency = 1 / 20 ns = 50 MHz
    - 1 Hz = 1/s

Freq	Period
100 GHz	0.01 ns
10 GHz	0.1 ns
1 GHz	1 ns
100 MHz	10 ns
10 MHz	100 ns

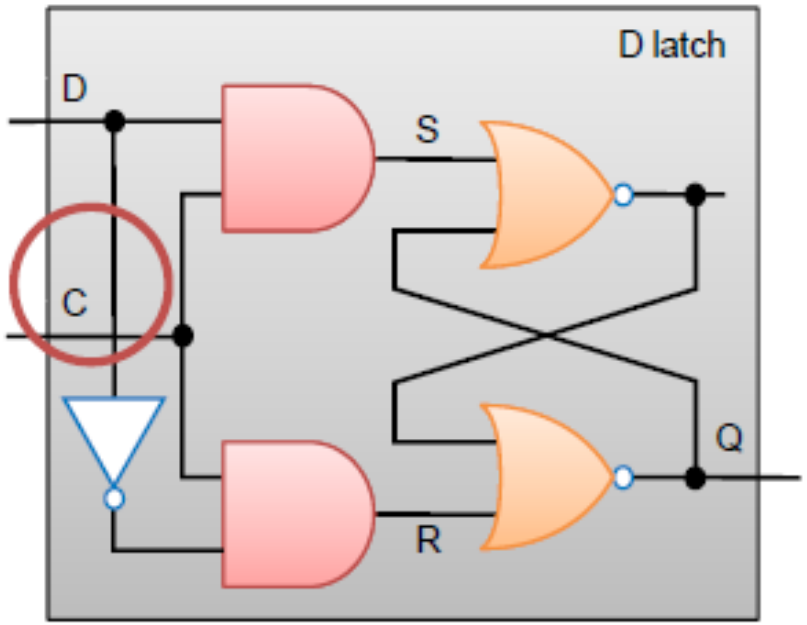
Clock Signals for a Latch

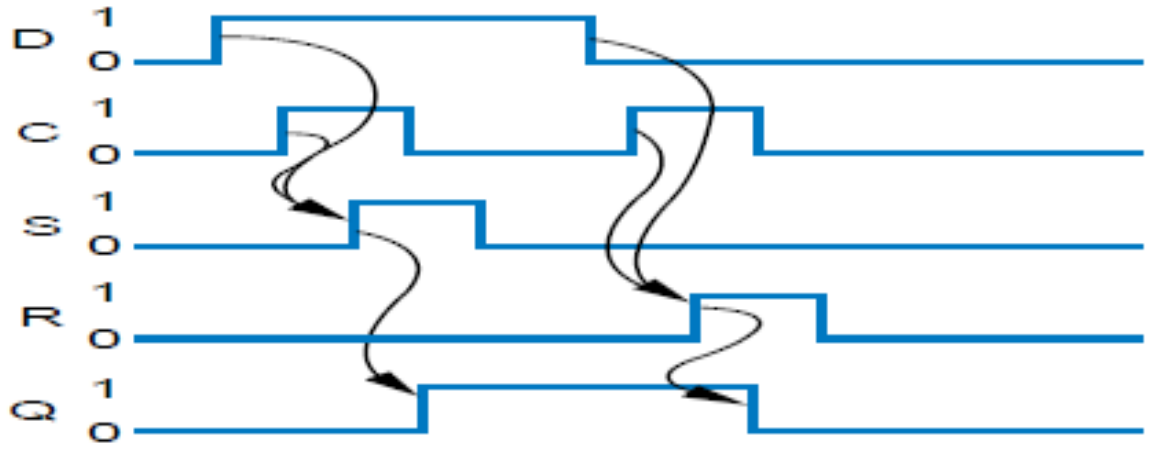
- How do we know when it's safe to set C=1?
  - Most common solution –make C pulse up/down
  - C=0: Safe to change X, Y C=1: Must *not* change X, Y
  - ***Clock*** signal -- Pulsing signal used to enable latches
    - Because it ticks like a clock
  - Sequential circuit whose storage components all use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

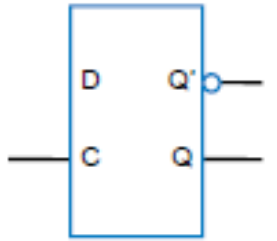


Level-Sensitive D Latch

- 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



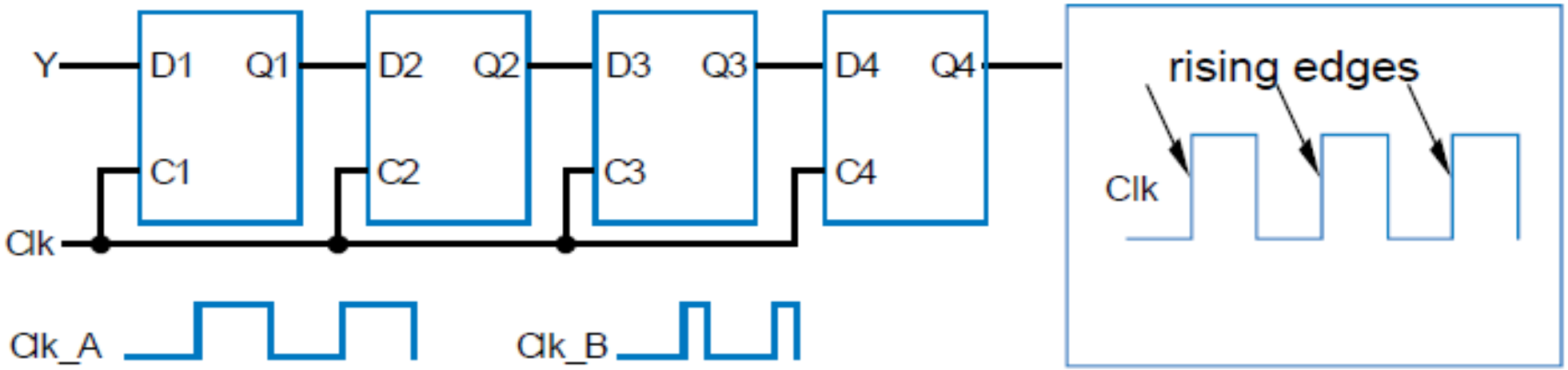




D latch symbol

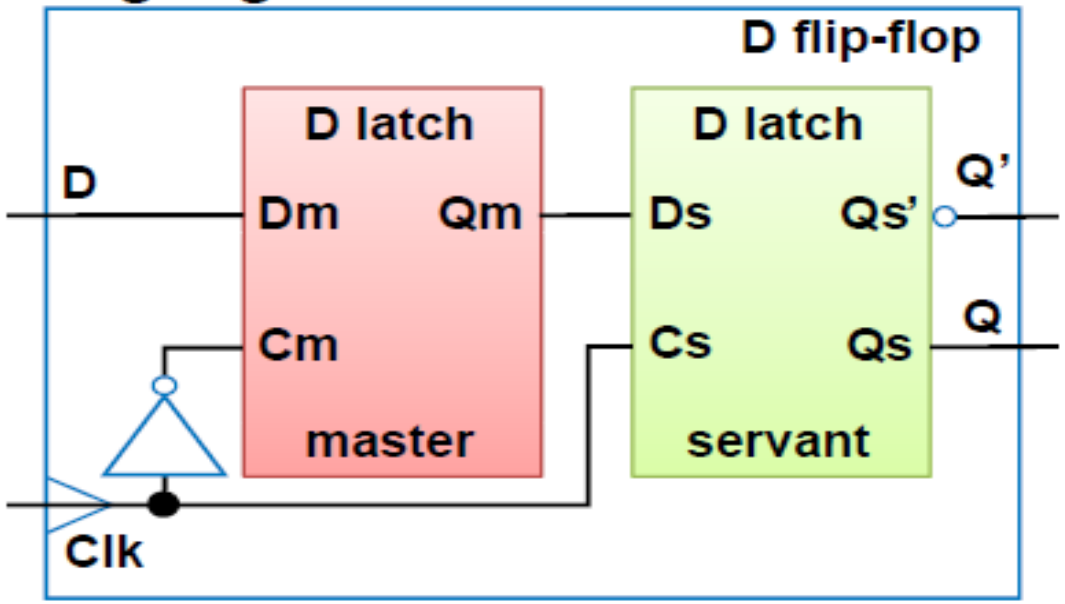
Problem with Level-Sensitive D Latch

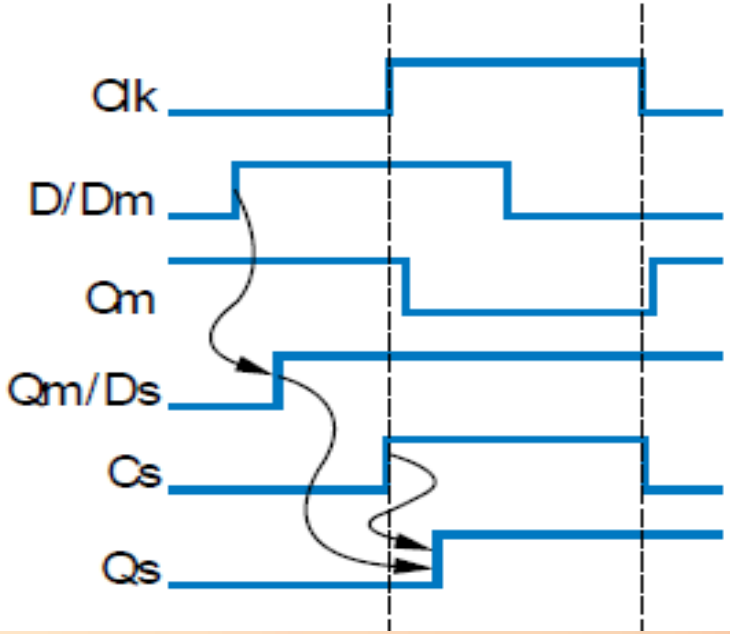
- D latch still has problem (as does SR latch)
  - When C=1, through how many latches will a signal travel?
  - Depends on for how long C=1
    - Clk_A -- signal may travel through multiple latches
    - Clk_B -- signal may travel through fewer latches
  - Hard to pick C that is just the right length
    - Can we design bit storage that only stores a value on the rising edge of a clock signal?



Master -Slave D Flip-Flop

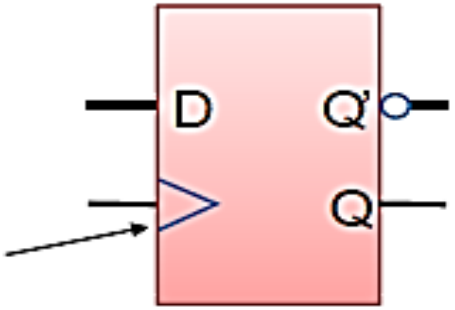
- ***Flip-flop***: stores on clock edge, not level
  - Two latches, output of first goes to input of second, master latch has inverted clock signal
  - So master loaded when C=0, then servant when C=1
  - When C changes from 0 to 1, master disabled, servant loaded with value that was at D just before C changed -- i.e., value at D during rising edge of C



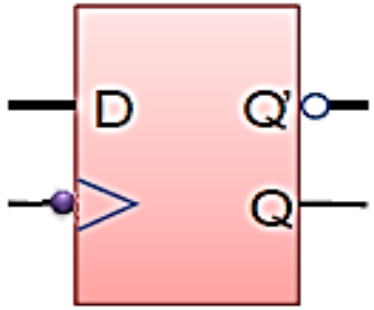


D Flip-Flop

The triangle means clock input, edge triggered

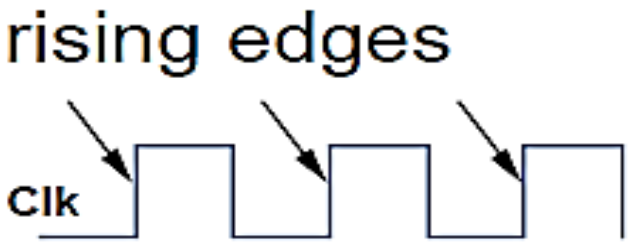


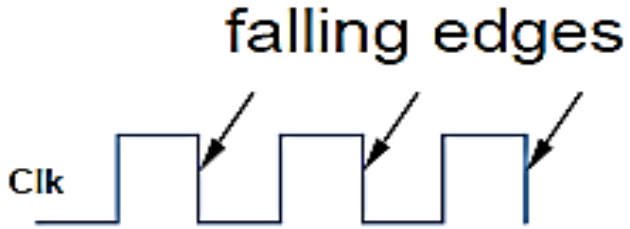
Symbol for rising-edge triggered D flip-flop



Internal design: Just invert servant clock rather than master

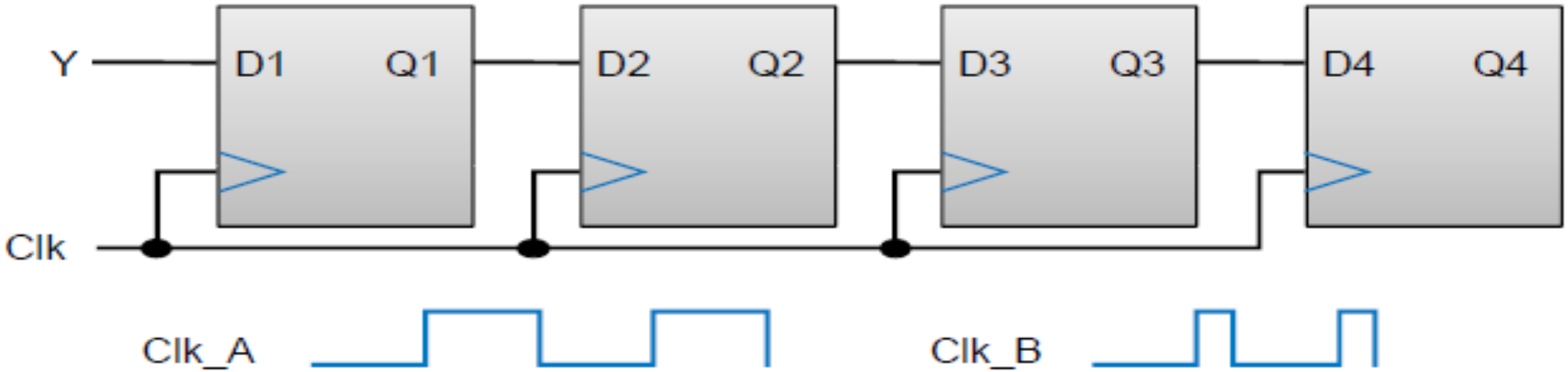
Symbol for falling-edge triggered D flip-flop





D Flip-Flop

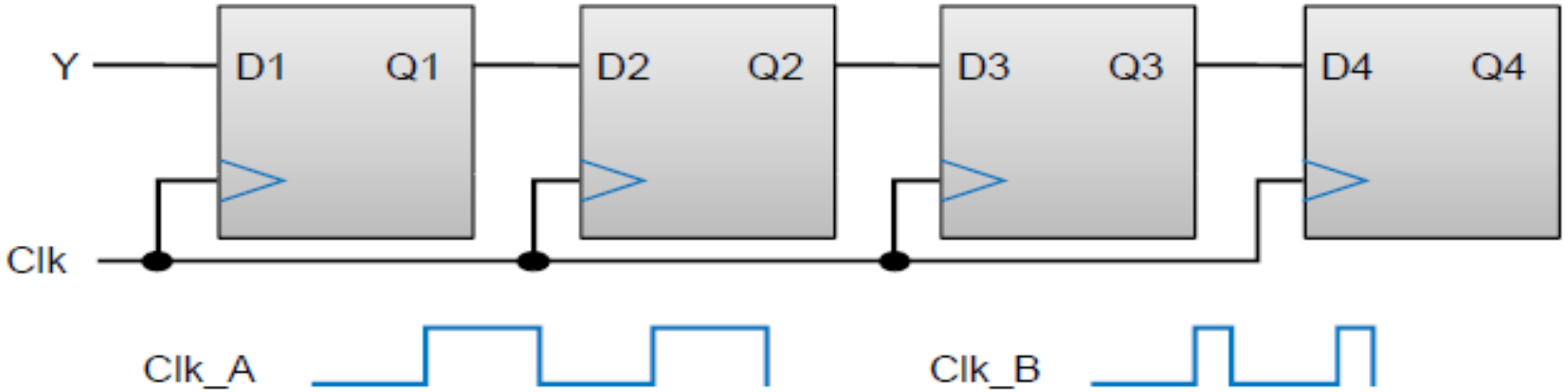
- Solves problem of not knowing through how many latches a signal travels when C=1
  - Signal travels through exactly one flip-flop, for Clk_A or Clk_B
  - Why? Because on rising edge of Clk, all four flip-flops are loaded simultaneously -- then all four no longer pay attention to their input, until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



Two latches inside each flip-flop

D Flip-Flop

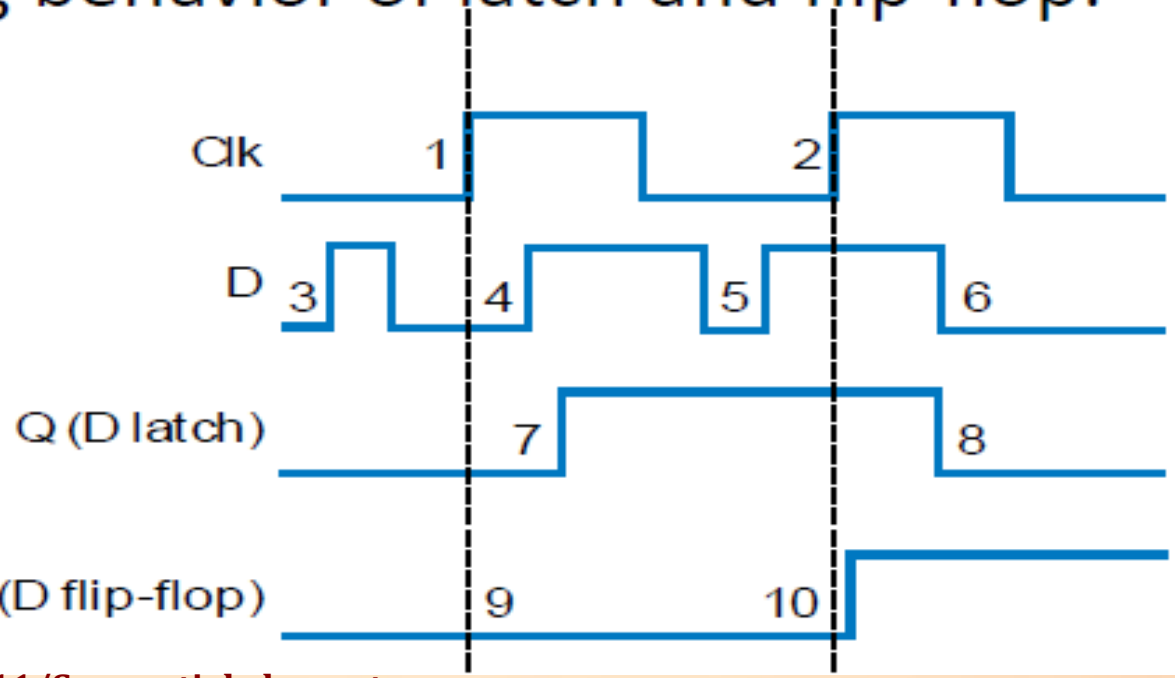
- Solves problem of not knowing through how many latches a signal travels when C=1
  - Signal travels through exactly one flip-flop, for Clk_A or Clk_B
  - Why? Because on rising edge of Clk, all four flip-flops are loaded simultaneously -- then all four no longer pay attention to their input, until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



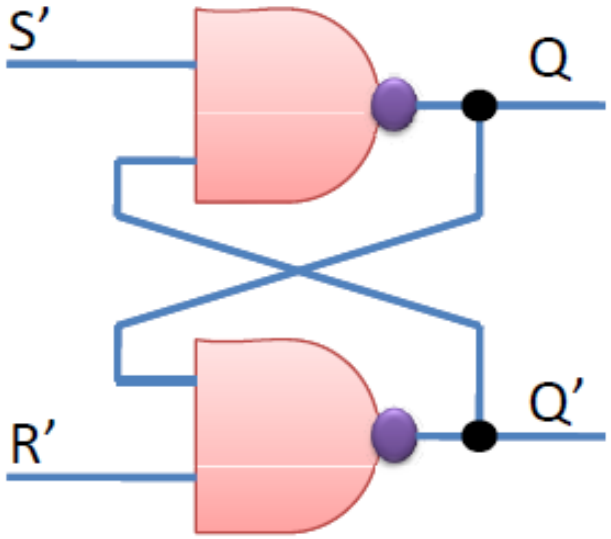
Two latches inside each flip-flop

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.
- Comparing behavior of latch and flip-flop:



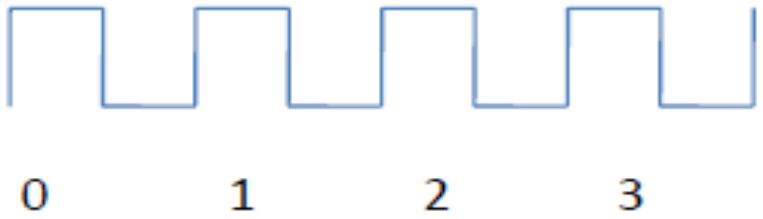
S'R' Latch

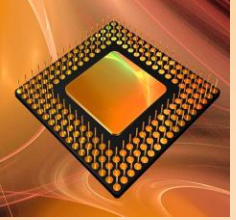


S'	R'	Q+
0	0	1*0* (Unpredictable)
0	1	1
1	0	0
1	1	Q

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



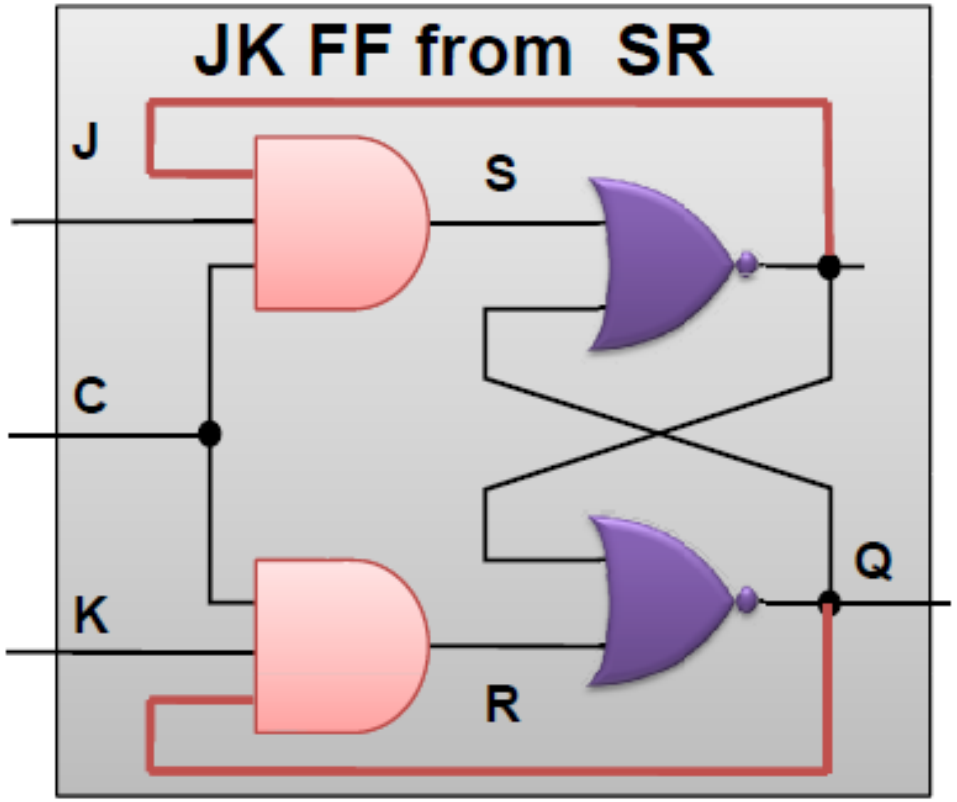


4 Type of Flip Flop

- **SR Flip Flop** : Set/Reset Flip Flop
- **D Flip Flop** : Data Flip Flop to store Bit
- **J-K Flip Flop**: SR with use of the unavoidable SR=11 state to Toggle (All input values are useful)
- **T Flip Flop**: Toggle Flip Flop

J-K Flip Flop

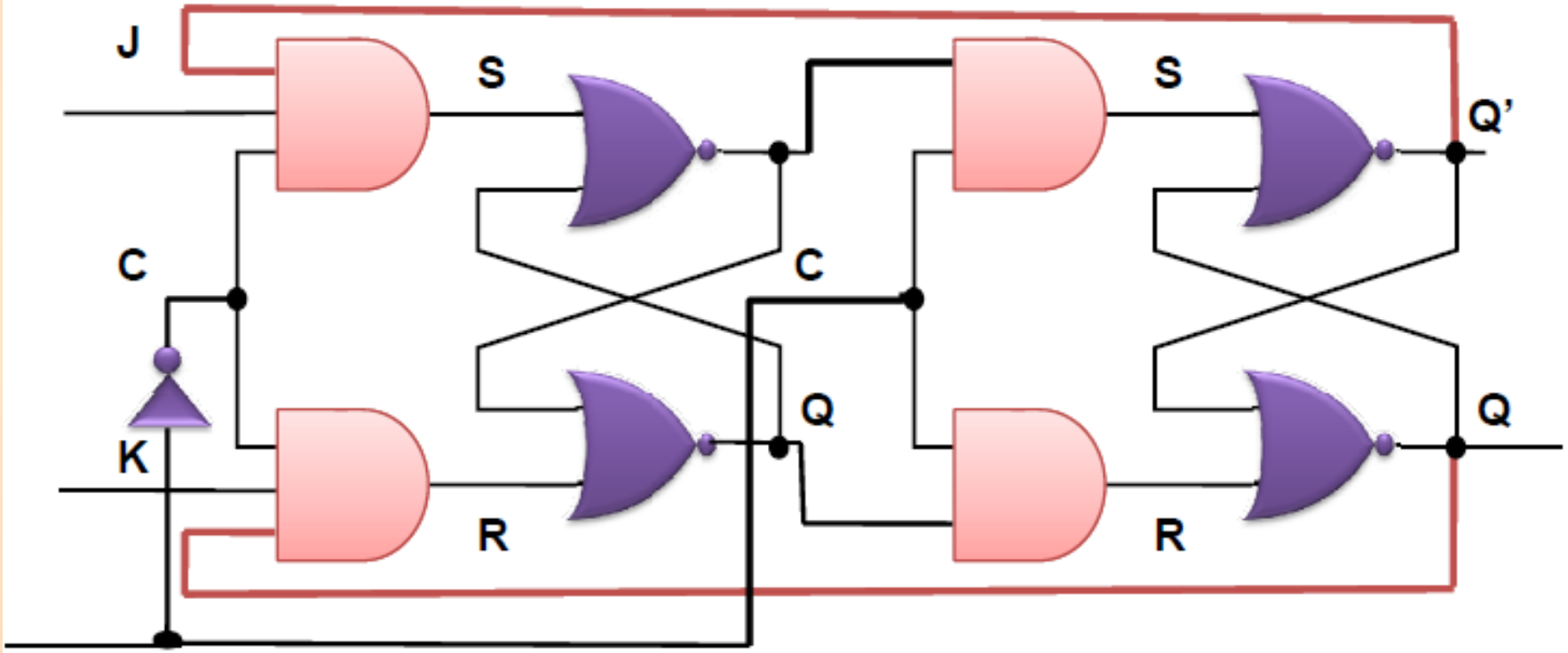
- The JK flip-flop augments the behavior of the SR flip-flop (J=Set, K=Reset) by interpreting the S = R = 1 condition as a "flip" or toggle command.



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



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

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.

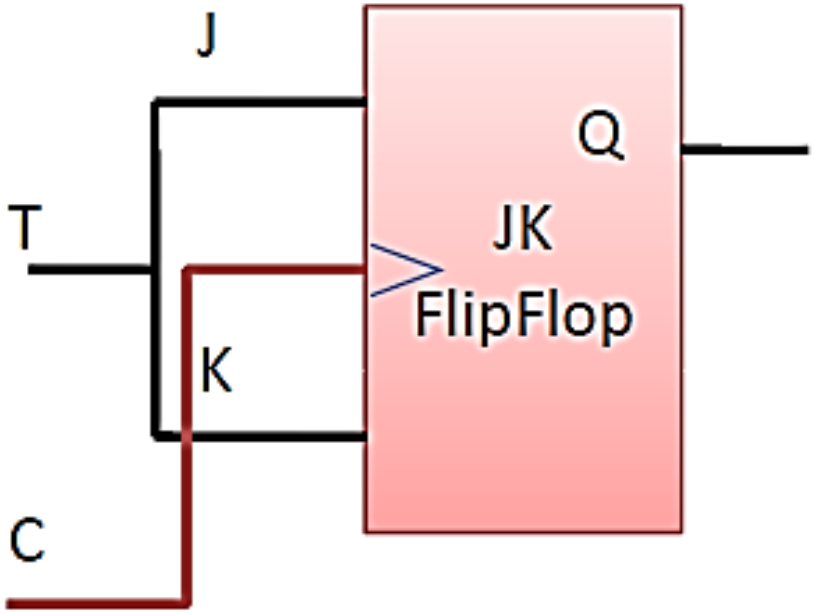
J=T	K=T	Q+
0	0	Qt
0	1	0
1	0	1
1	1	Qt'

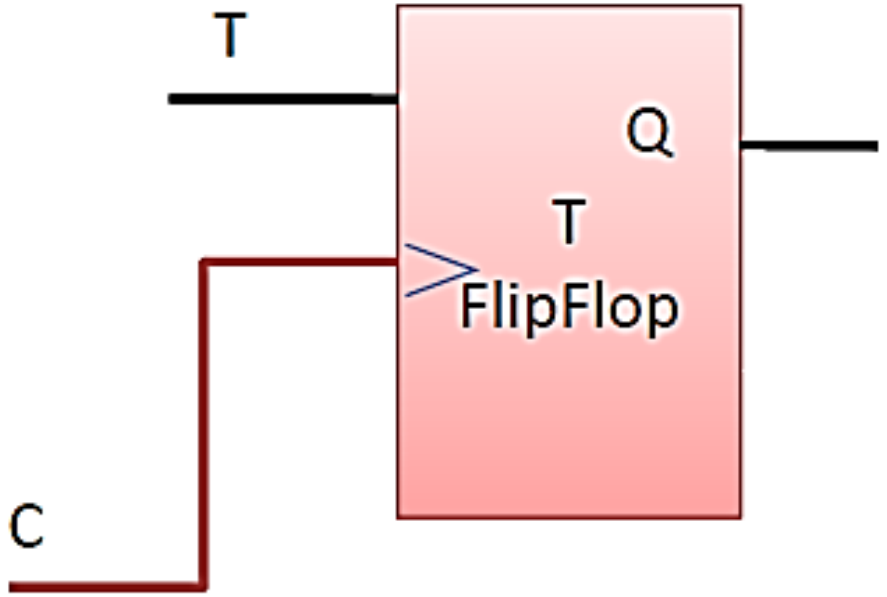
J=D	K=D'	Q+
0	0	Qt
0	1	0
1	0	1
1	1	Qt'

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, $Q^+ = Q'$





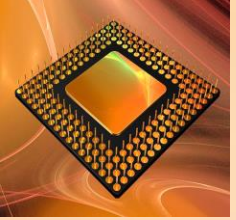
4 Types of Flip-Flops

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'



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'**

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

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

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

$Q^{+}=D$

$Q(t+1) = D$

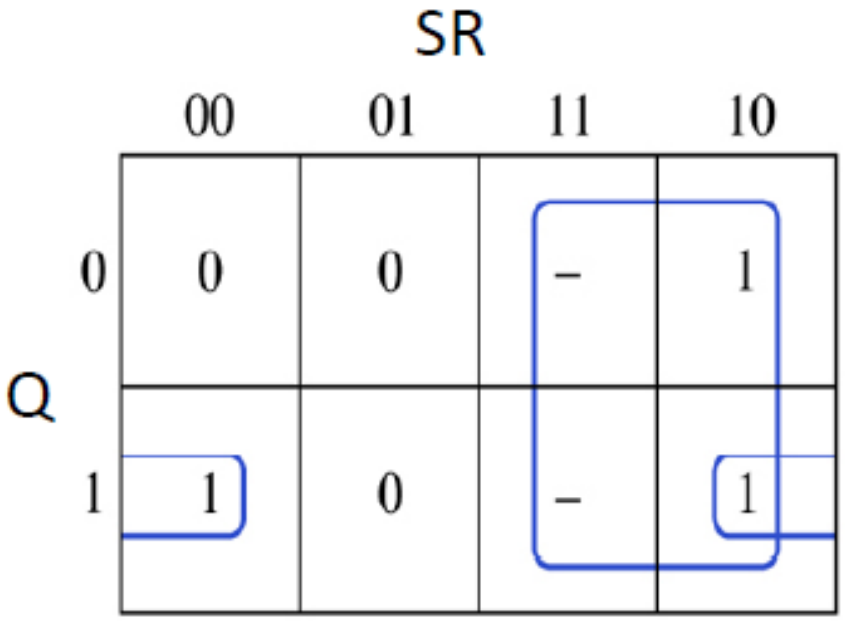
T	Q+
0	Qt
1	Qt'

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

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

$= T \oplus Q(t)$

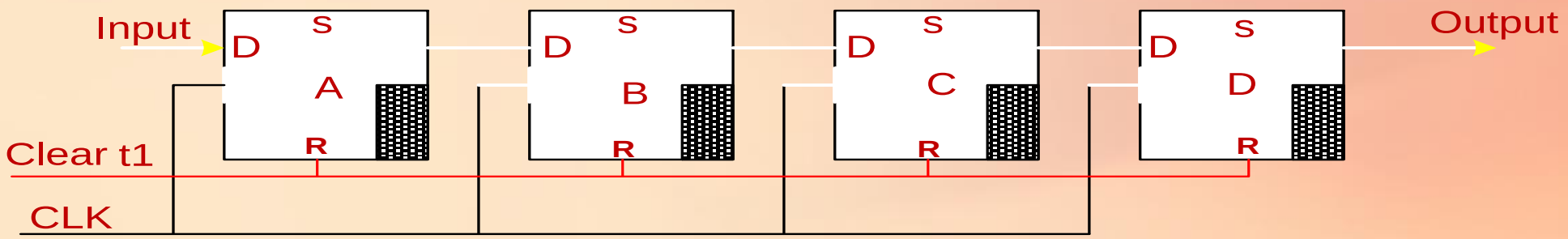
Characteristic equations

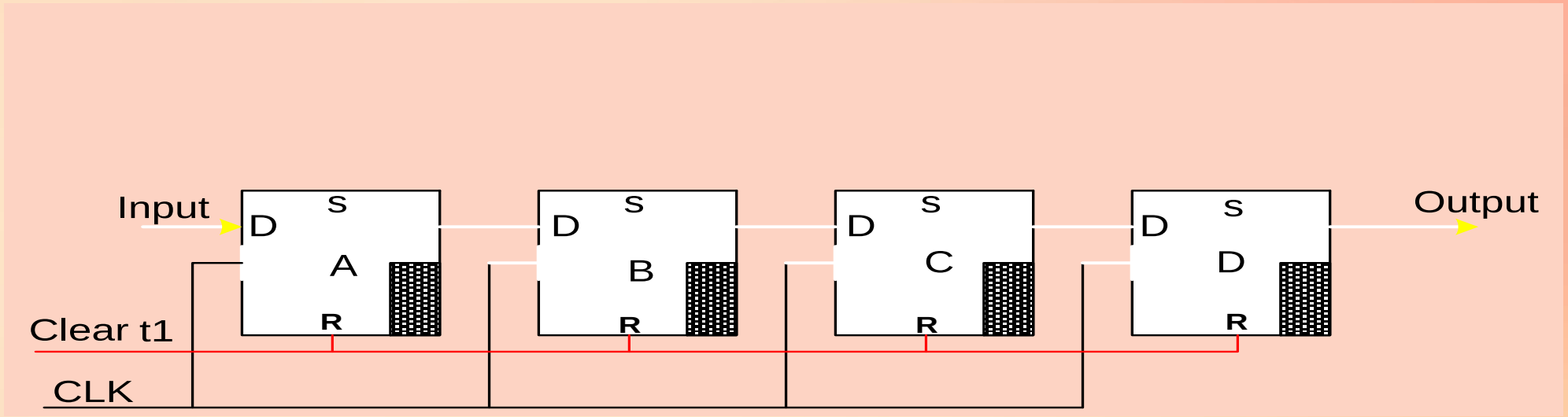


$Q^+ = S + R'Q \quad (SR=0)$

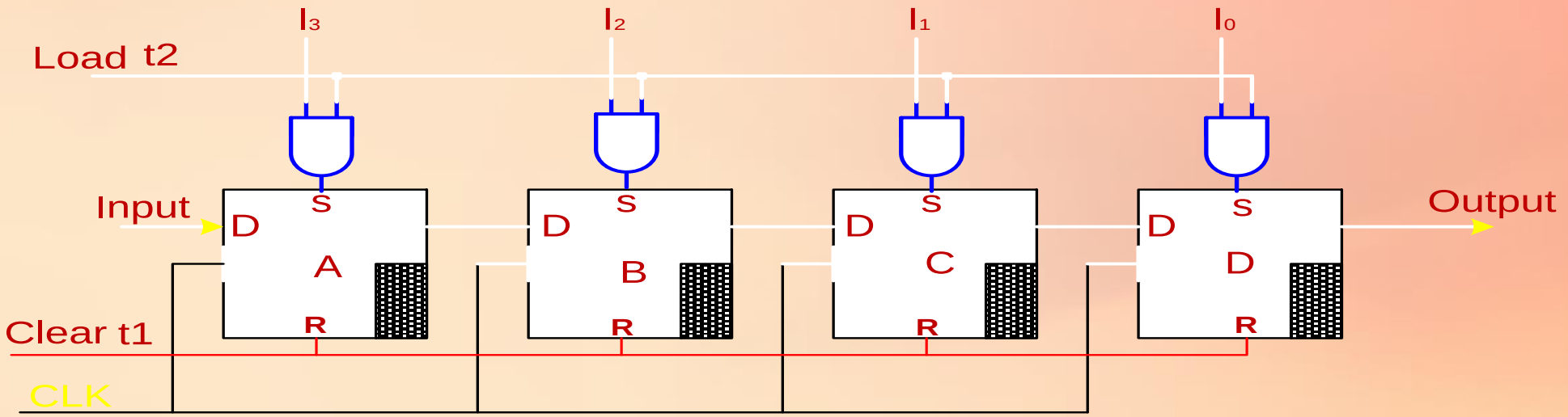
Flip Flop Type	Characteristic Equation
SR	$Q^+ = S + R'Q \quad (SR=0)$
JK	$Q^+ = JQ' + K'Q$
D	$Q^+ = D$
T	$Q^+ = TQ' + T'Q = T \oplus Q$

Shift Register

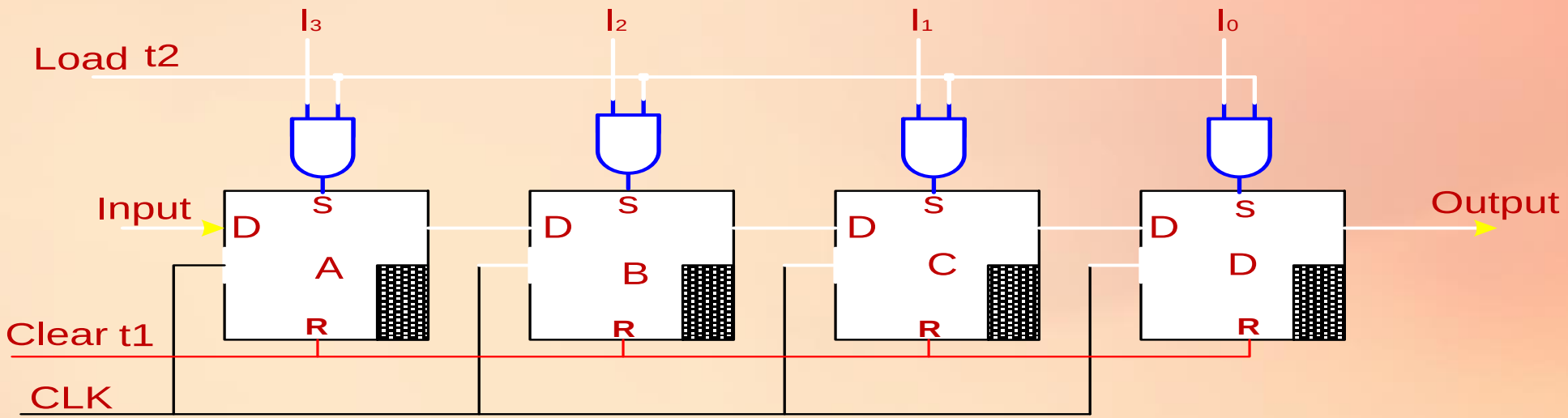



Input
D
S
A
R
D
S
B
R
D
S
C
R
D
S
D
R
Output
Clear t1
CLK

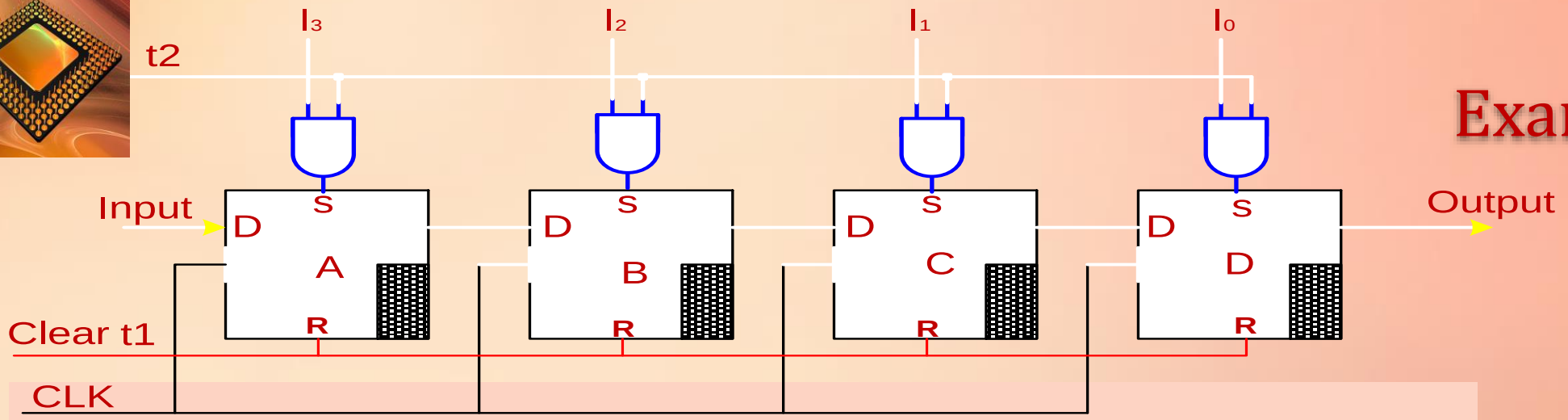
Shift Register

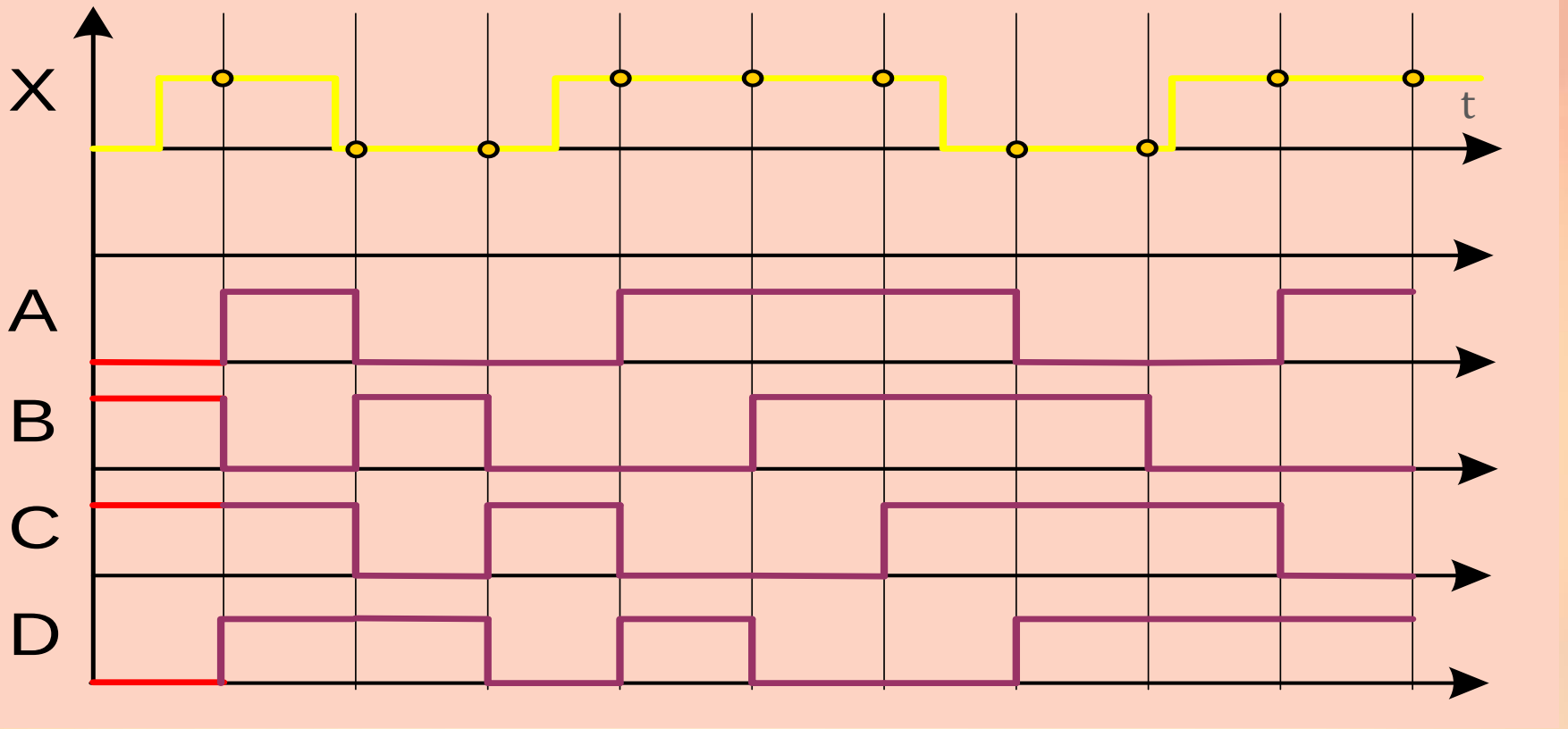


Shift Register

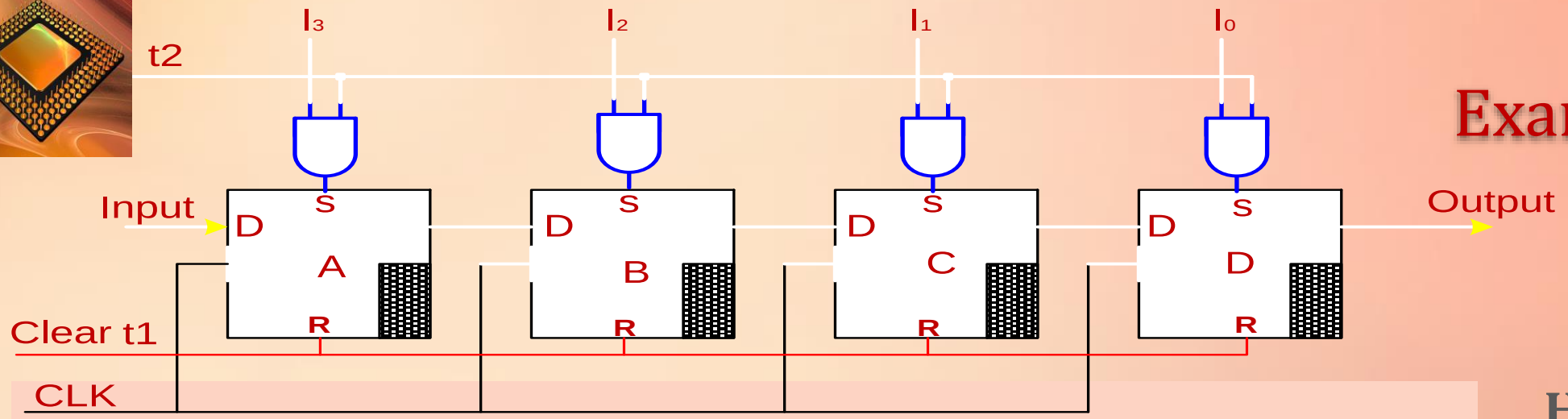


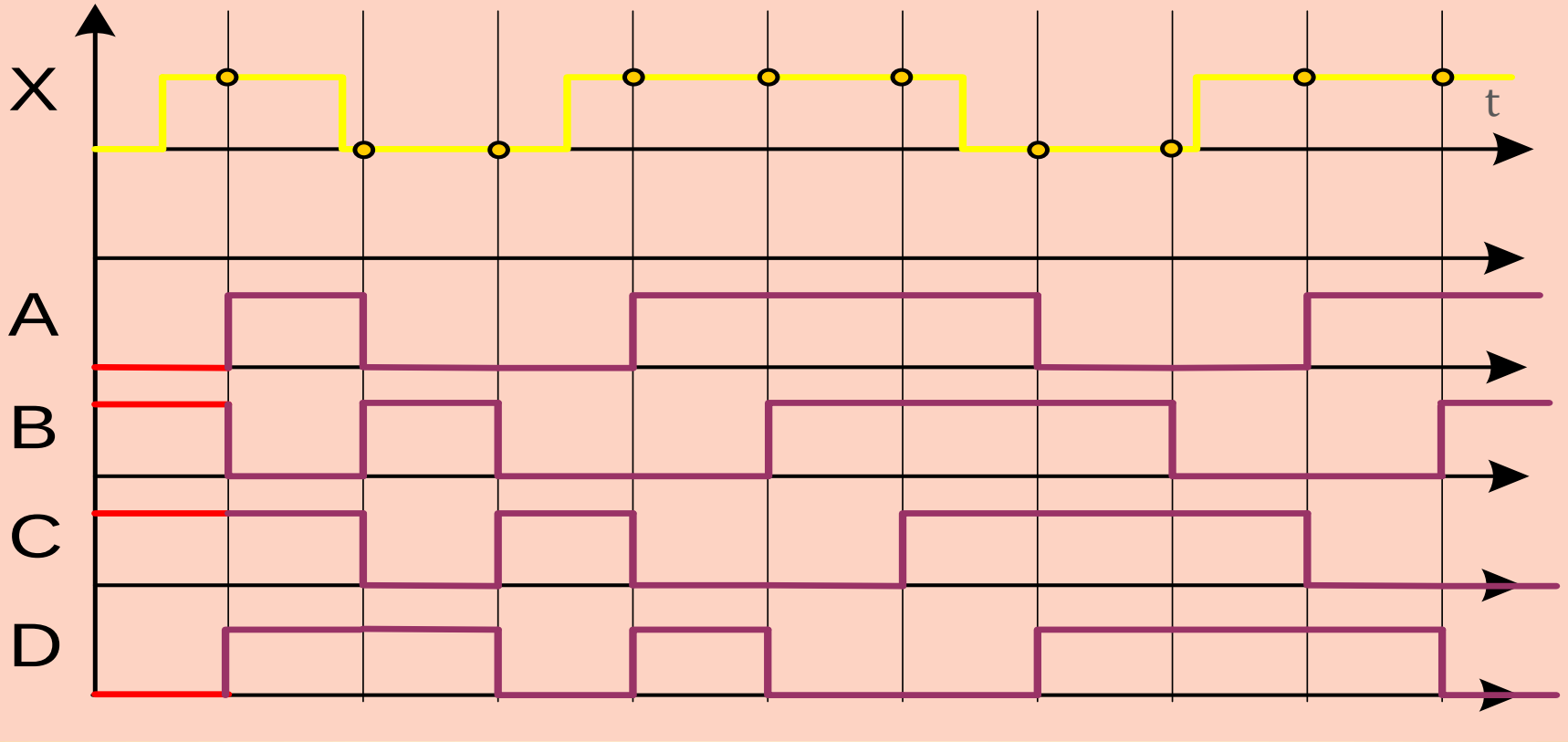
Example





Example

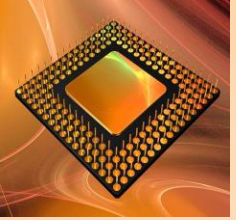




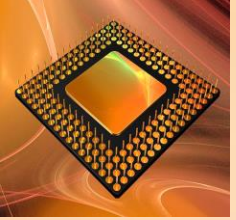
Hold time

Setup time

Maximum CLK Rate or Maximum Frequency

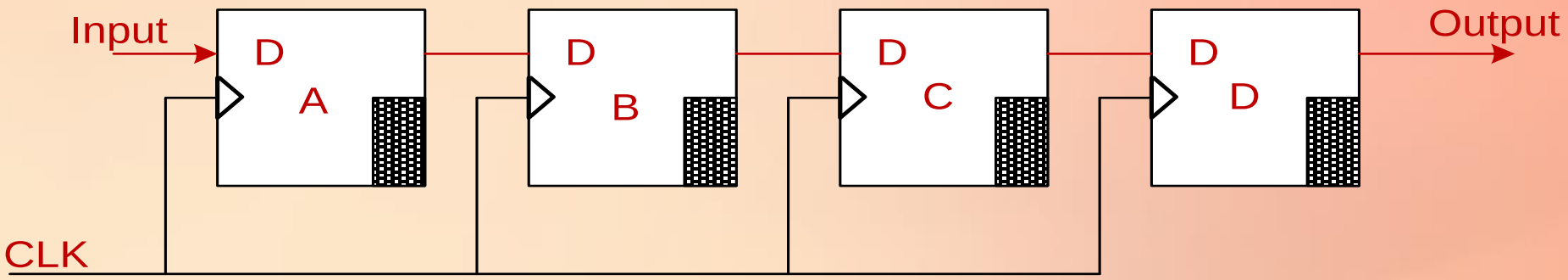


1-SR-FF
2-D-FF
3-T-FF
4-JK-MS-FF



SHIFT REGISTERS

Shift Right

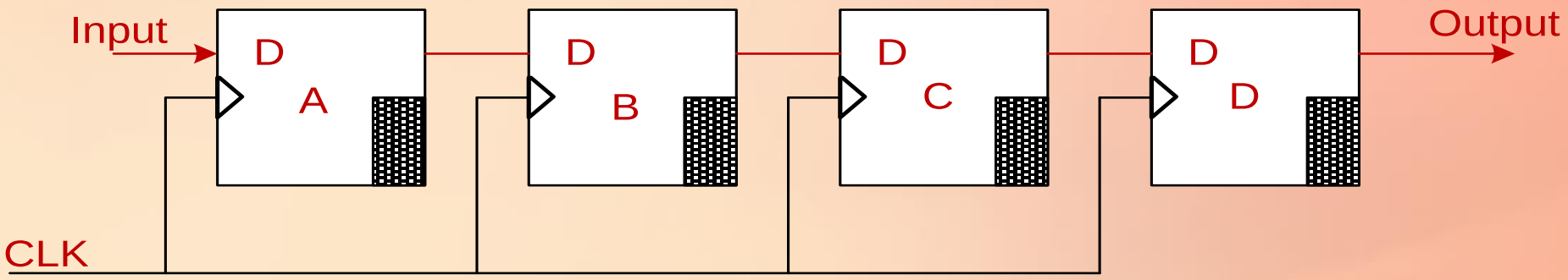


High to Low Edge

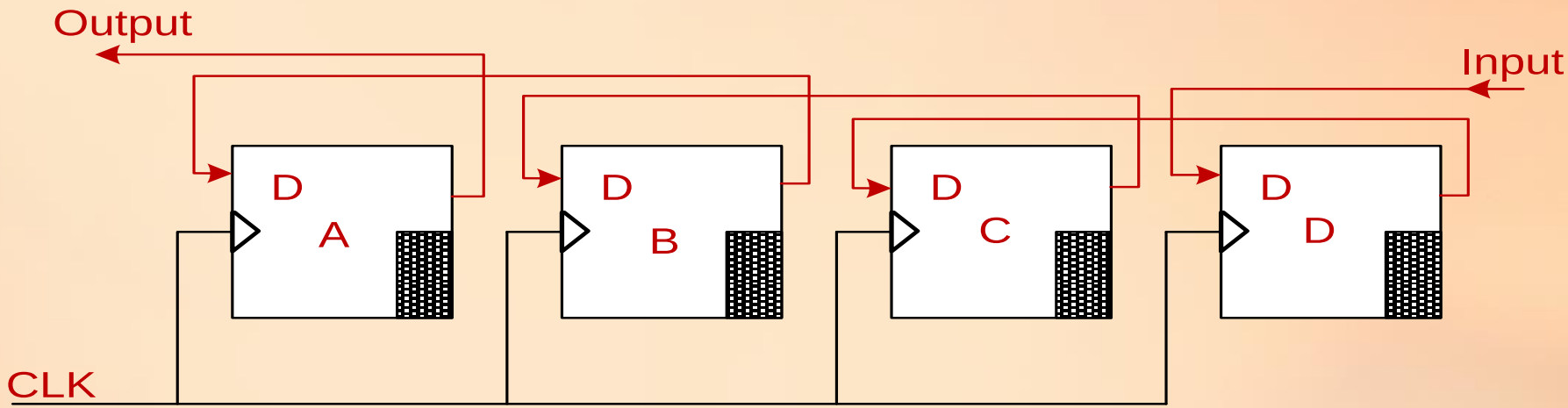
Low to High Edge

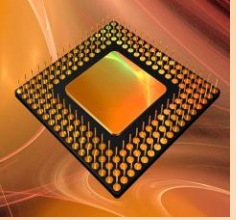
Efective Edge

Shift Right



Shift Left



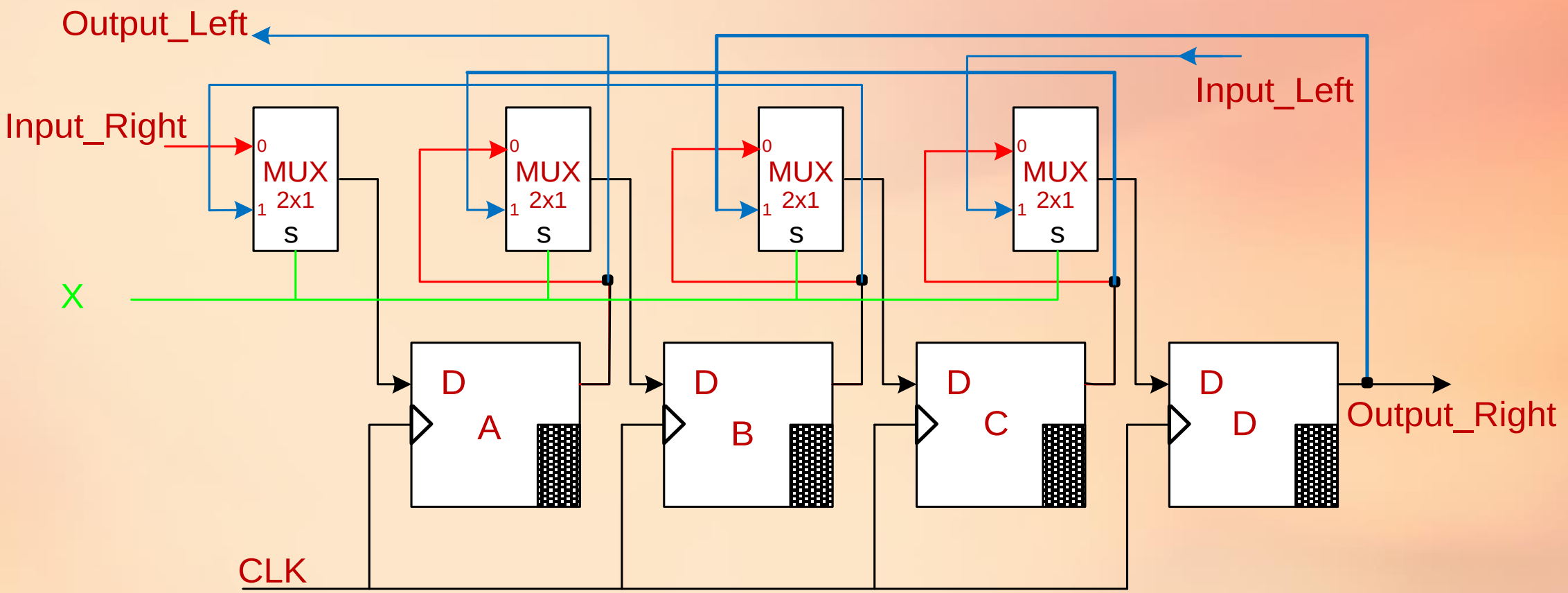


Shift Right and Left

X	
0	Shift _Right
1	Shift_Left

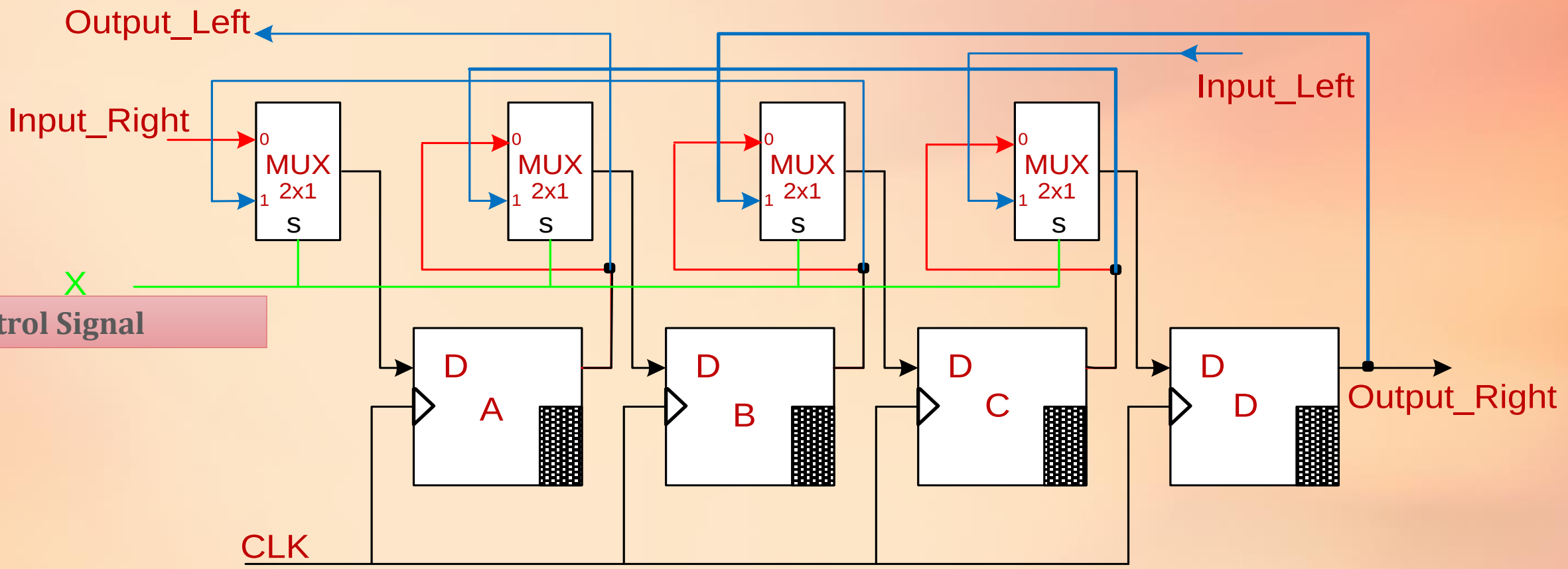
X	
0	Shift _Right
1	Shift_Left

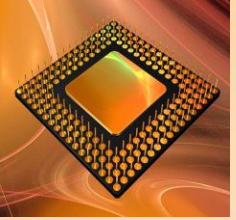
Shift Right and Left



Shift Right and Left

X	
0	Shift _Right
1	Shift_Left

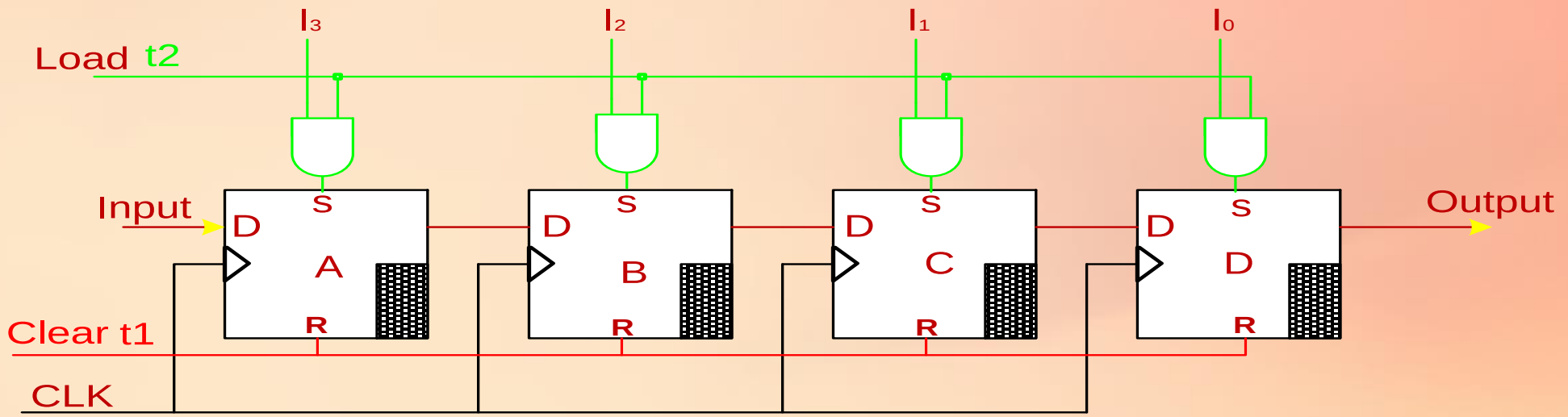




- **Parallel Load**
  (with Asynchronous inputs)

- **Parallel Load**
  (with Synchronous inputs)

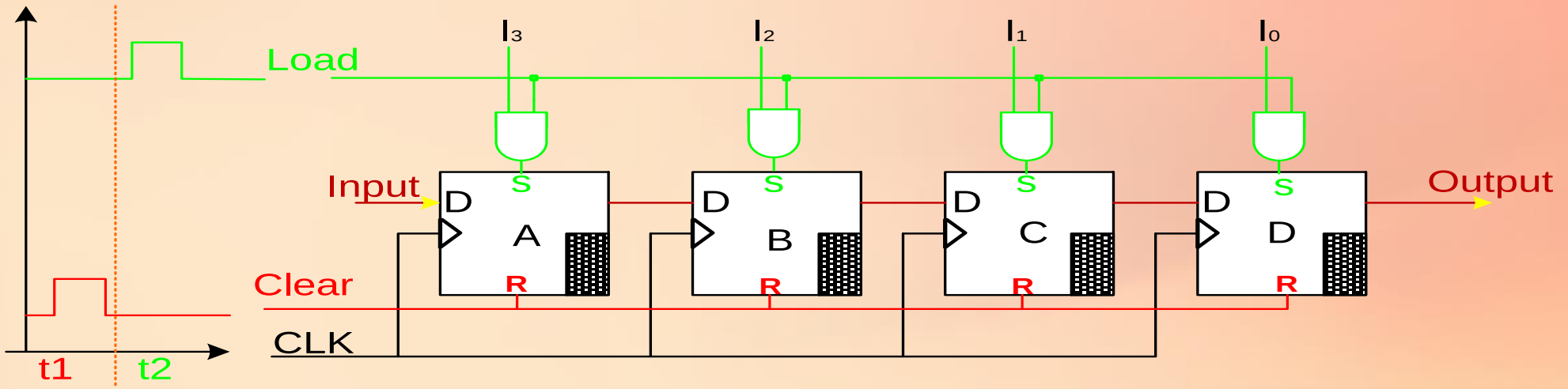
❑ Parallel Load (Asynchronous pins)



S: Preset , Primary Set

R: Clear , Primary Reset

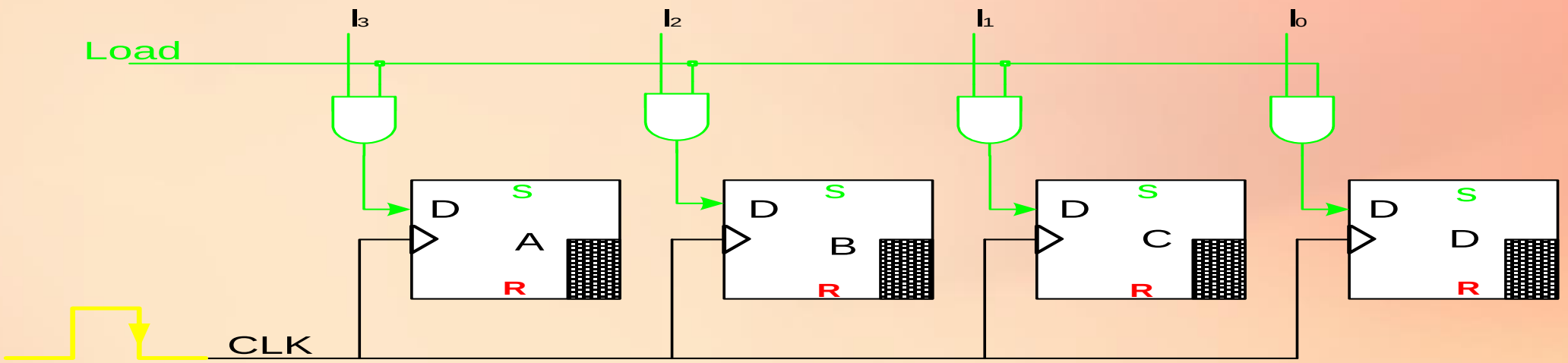
❑ Parallel Load (Asynchronous pins)



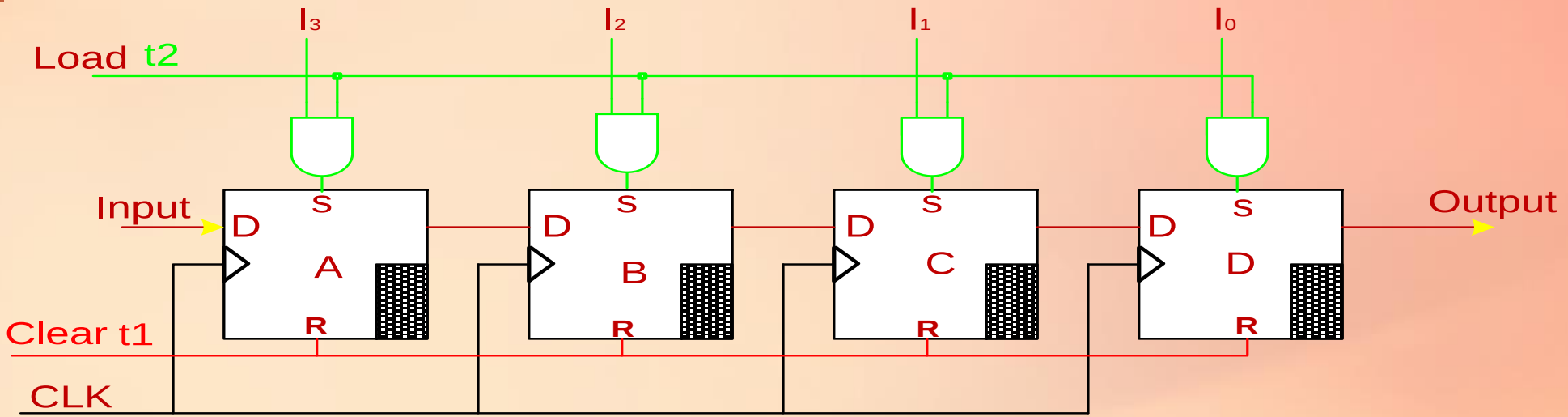
S: Preset , Primary Set

R: Clear , Primary Reset

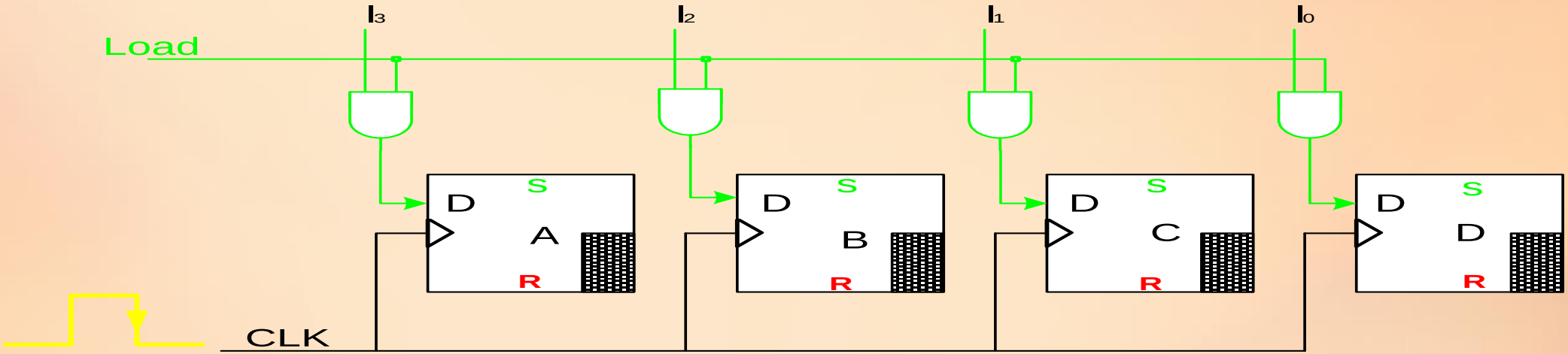
❑ Parallel Load (Synchronous pins)

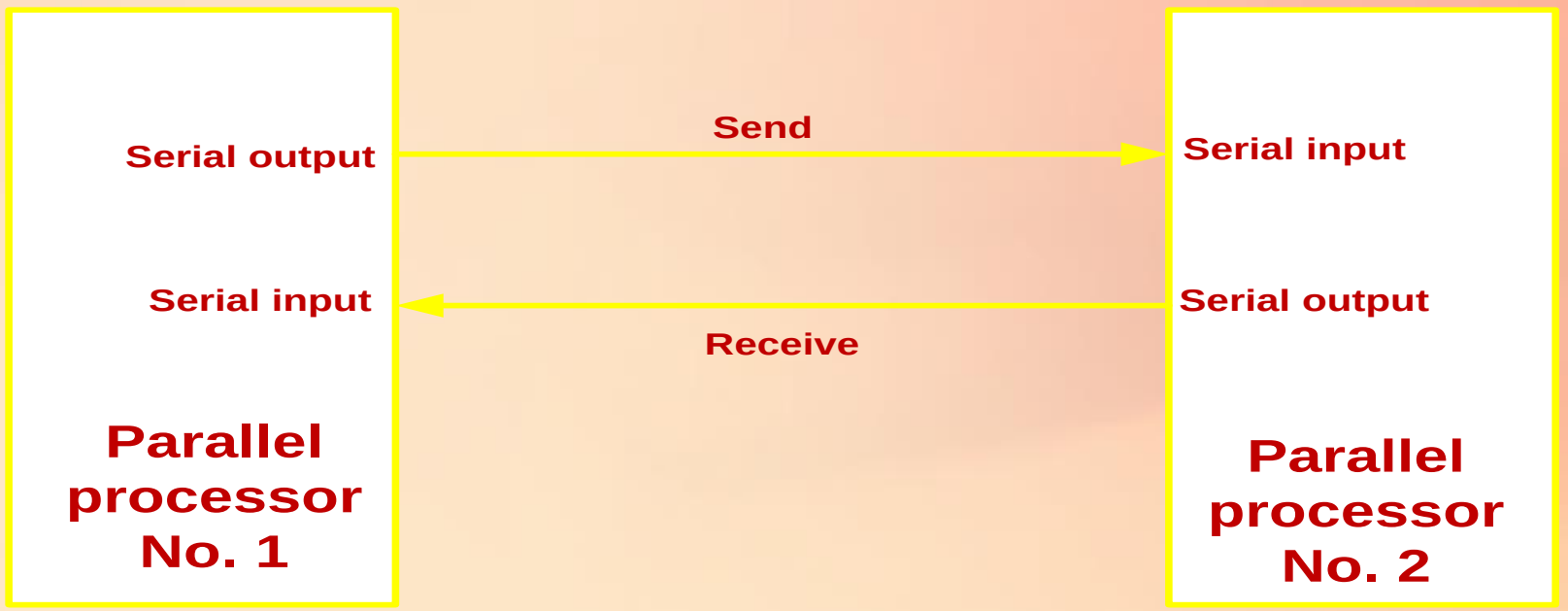


❑ Parallel Load (Asynchronous pins)

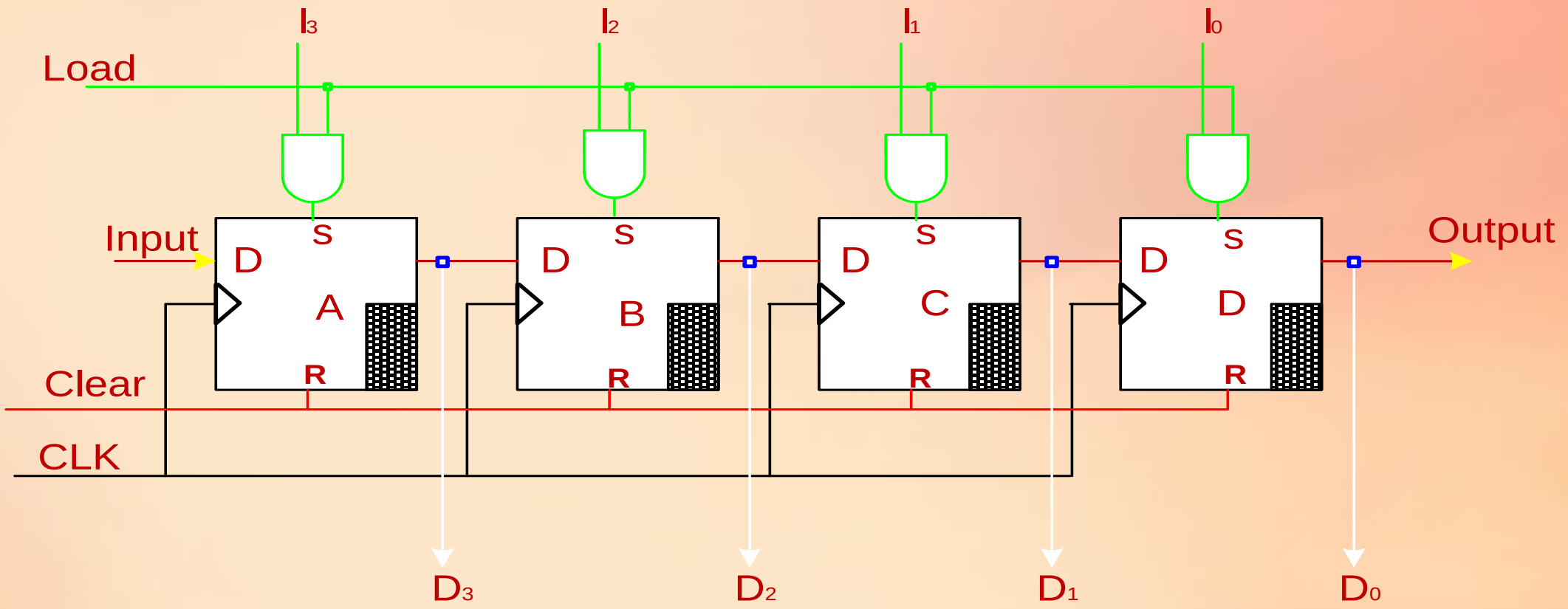


❑ Parallel Load (Synchronous pins)

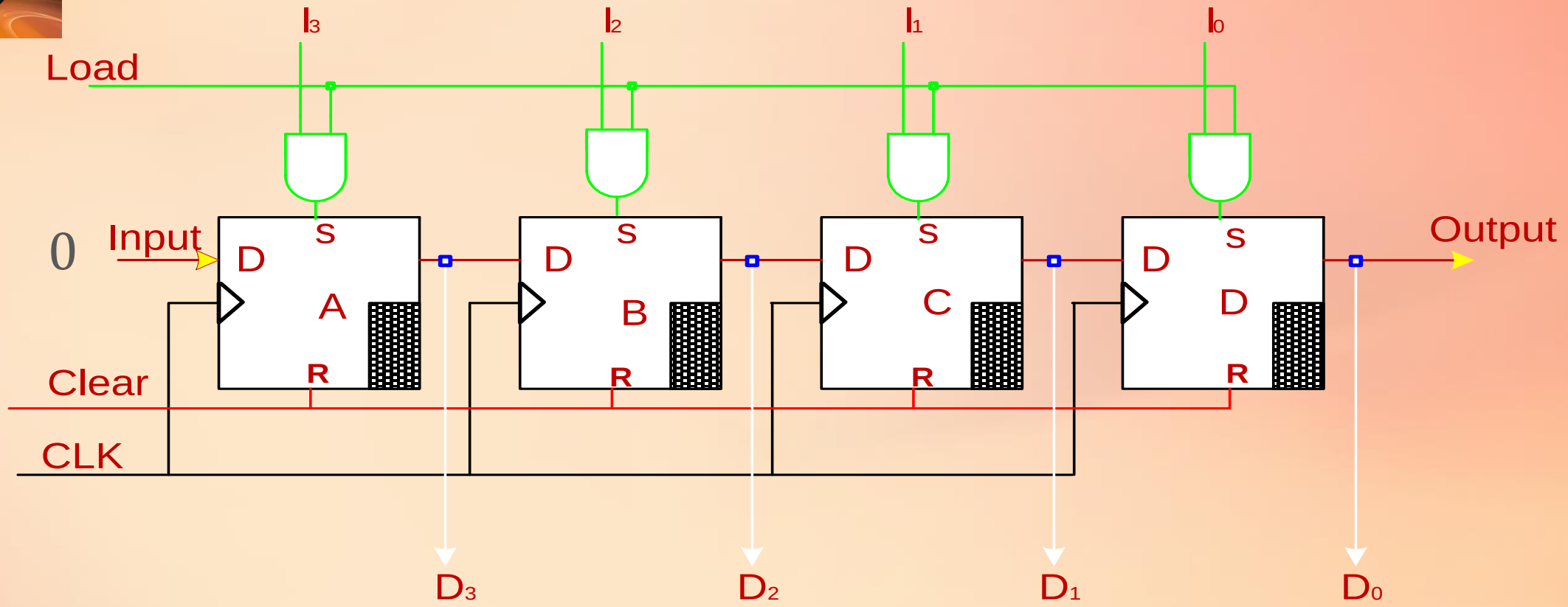


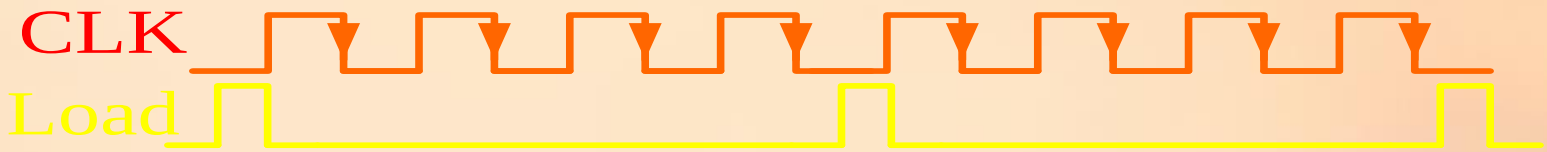

Serial output
Send
Serial input
Serial input
Serial output
Receive
Parallel processor No. 1
Parallel processor No. 2

Serial to parallel & Parallel to serial convertor

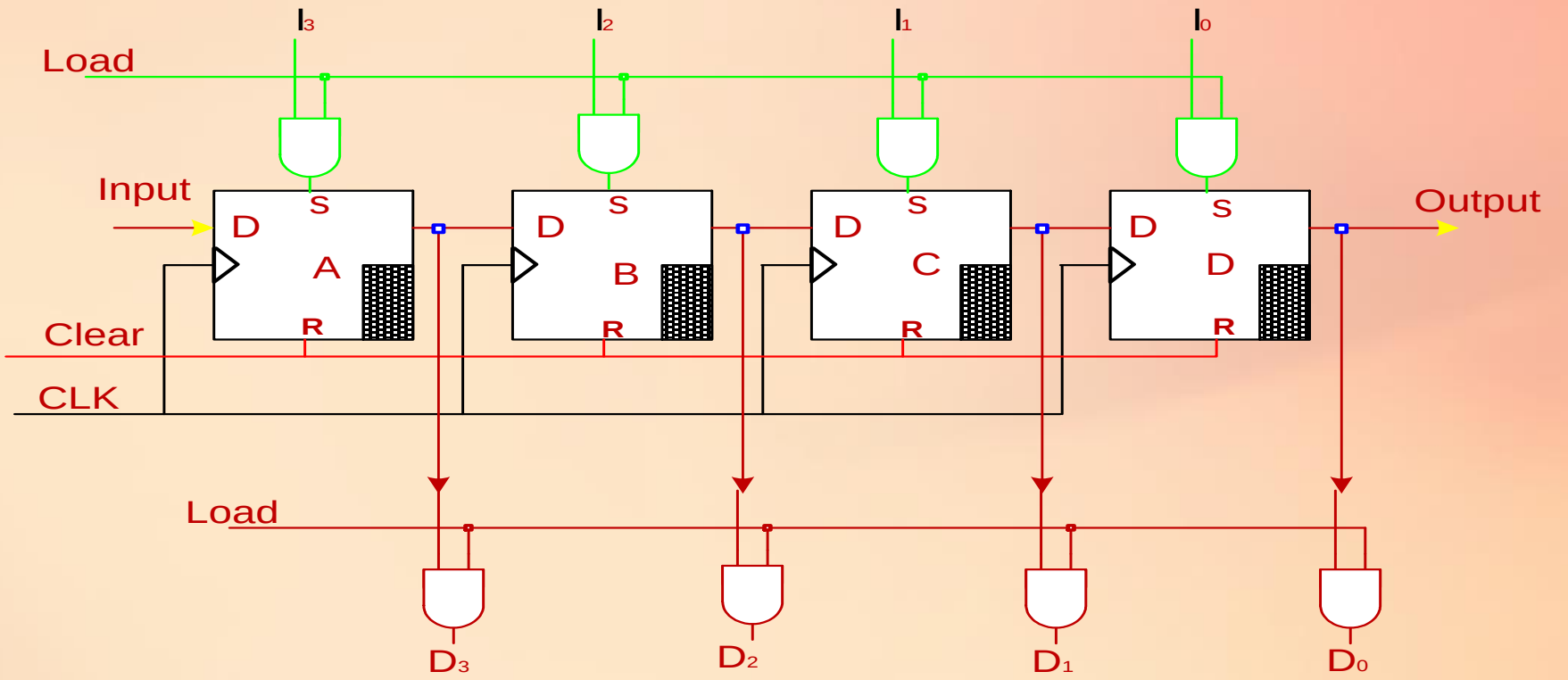


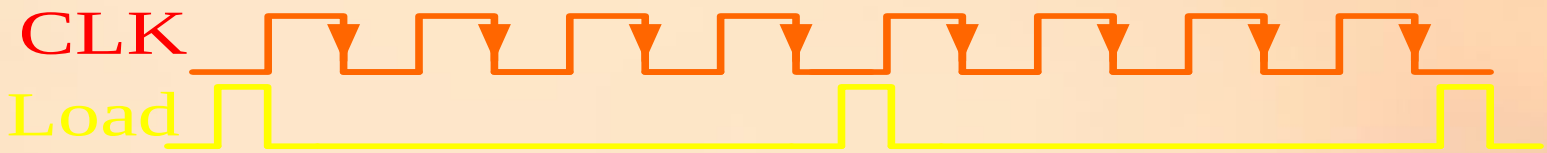
Parallel to serial convertor



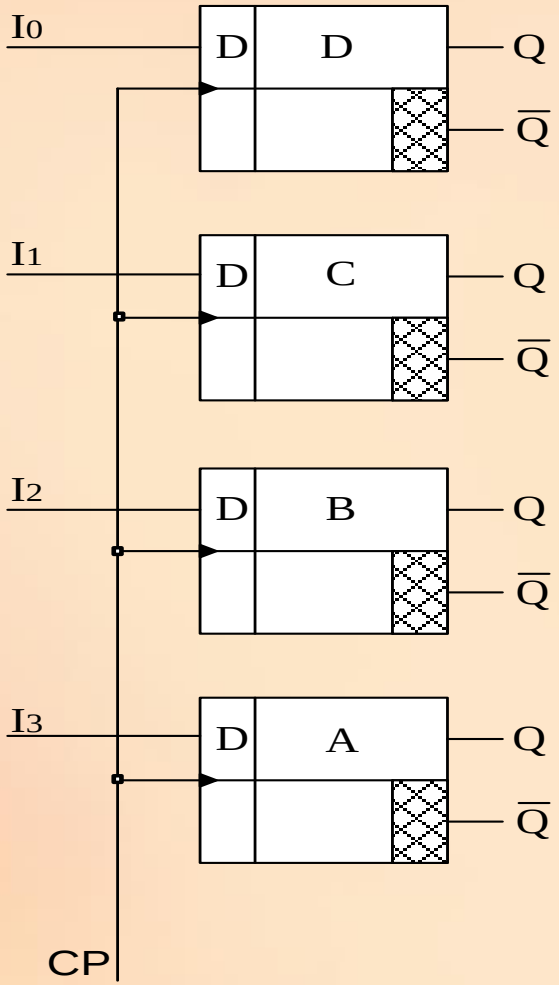


Serial to parallel convertor

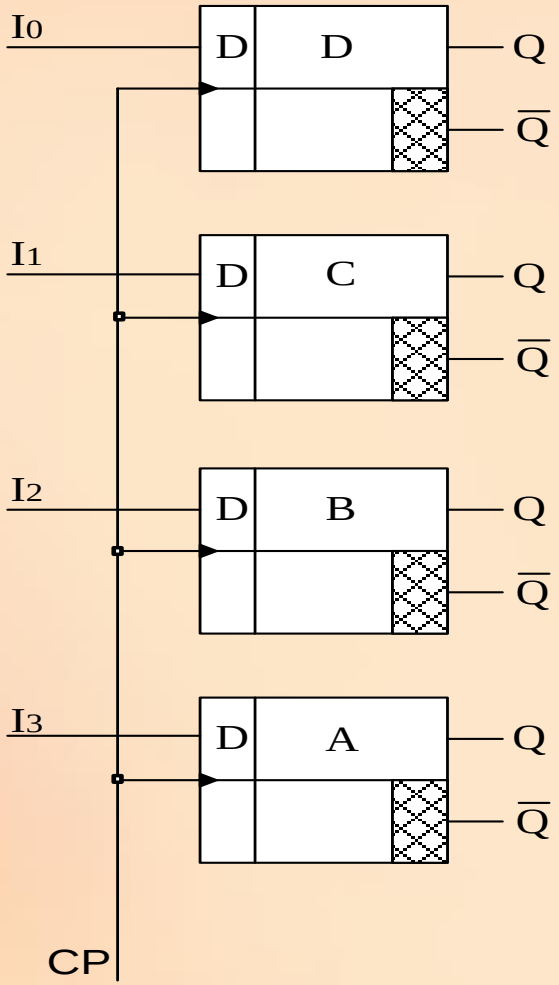


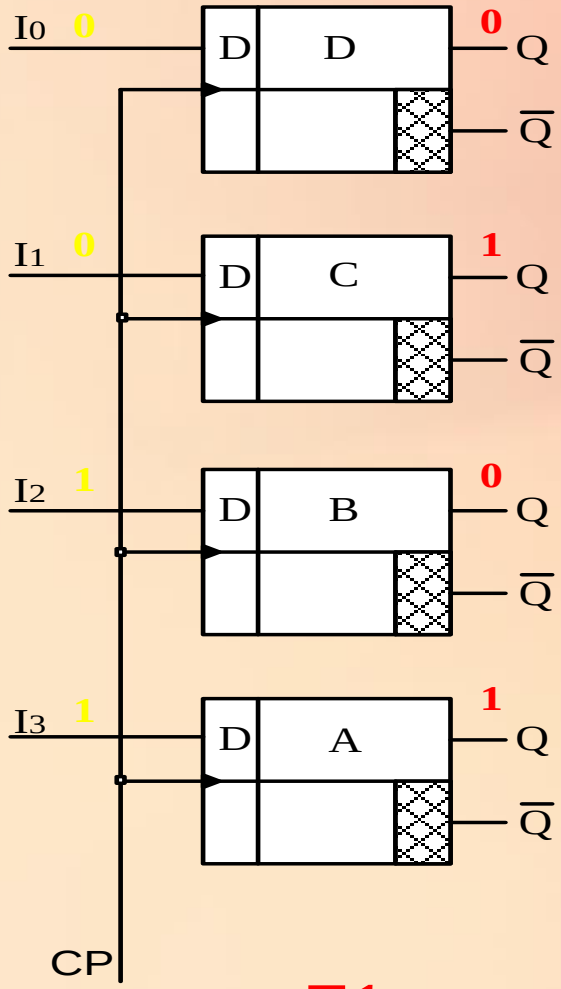


Buffer



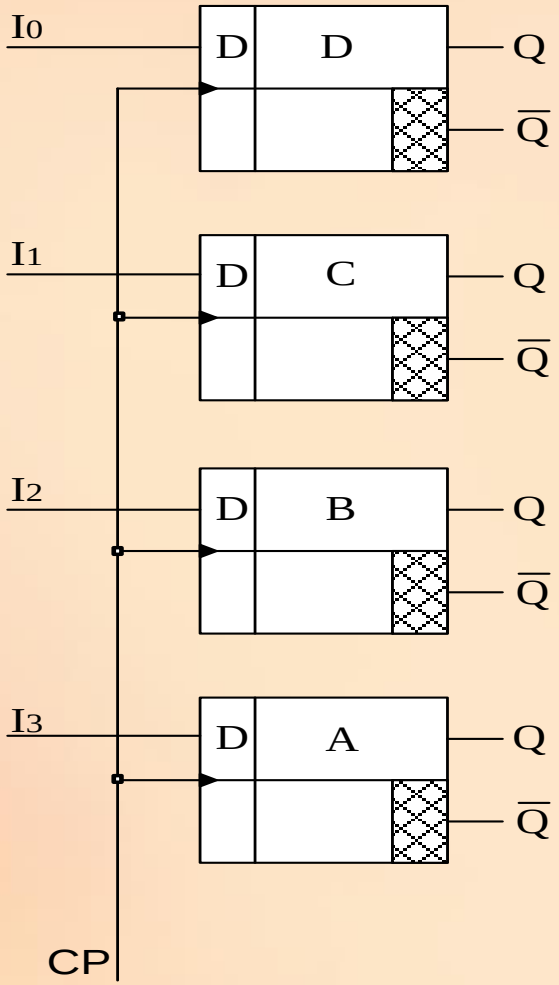
Buffer

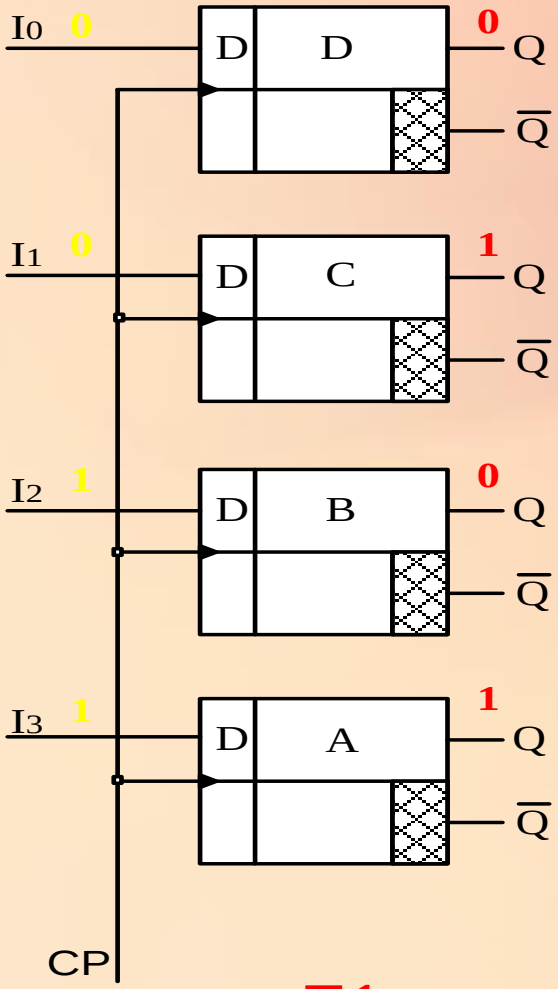

I0
D
D
Q
Q
I1
D
C
Q
Q
I2
D
B
Q
Q
I3
D
A
Q
Q
CP


I0 0
D
D
0
Q
Q
I1 0
D
C
1
Q
Q
I2 1
D
B
0
Q
Q
I3 1
D
A
1
Q
Q
CP

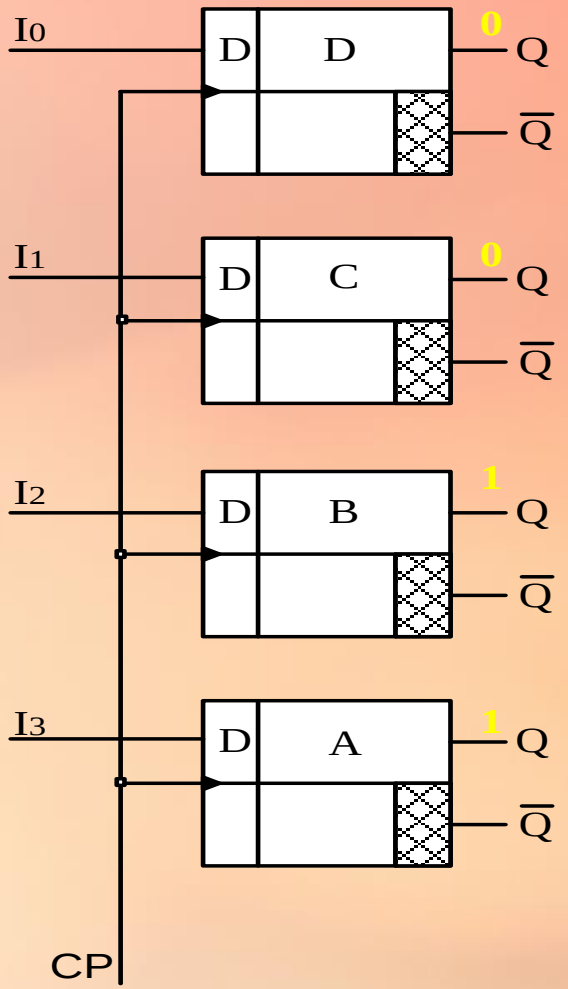
T1

Buffer


I0
D
D
Q
Q
I1
D
C
Q
Q
I2
D
B
Q
Q
I3
D
A
Q
Q
CP


I0 0
D
D
0
Q
Q
I1 0
D
C
1
Q
Q
I2 1
D
B
0
Q
Q
I3 1
D
A
1
Q
Q
CP

T1


I0
D
D
0
Q
Q
I1
D
C
0
Q
Q
I2
D
B
1
Q
Q
I3
D
A
1
Q
Q
CP

T2

THANK YOU