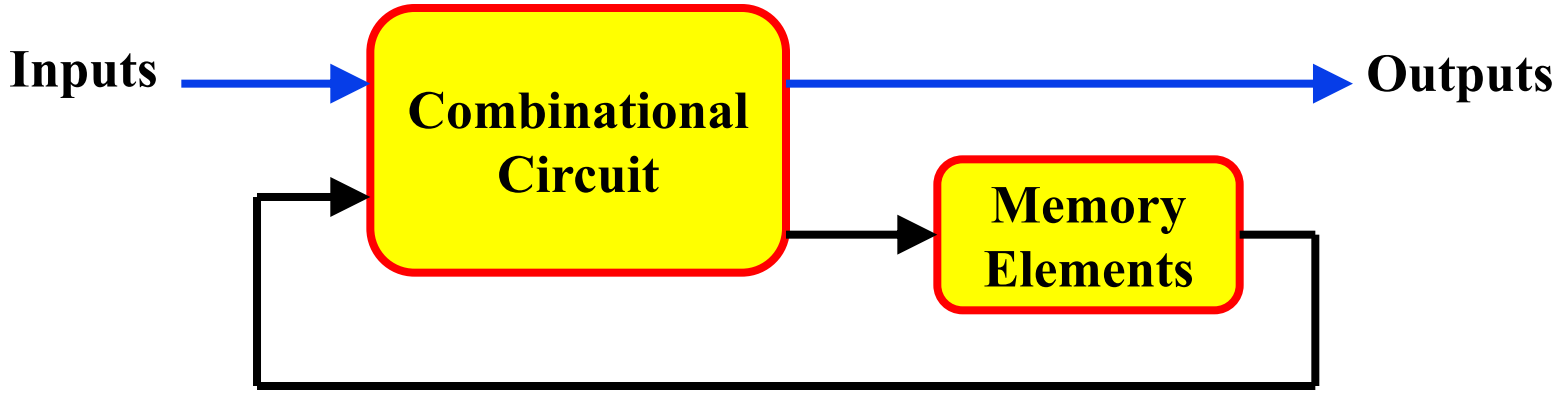
Synchronous Sequential Logic

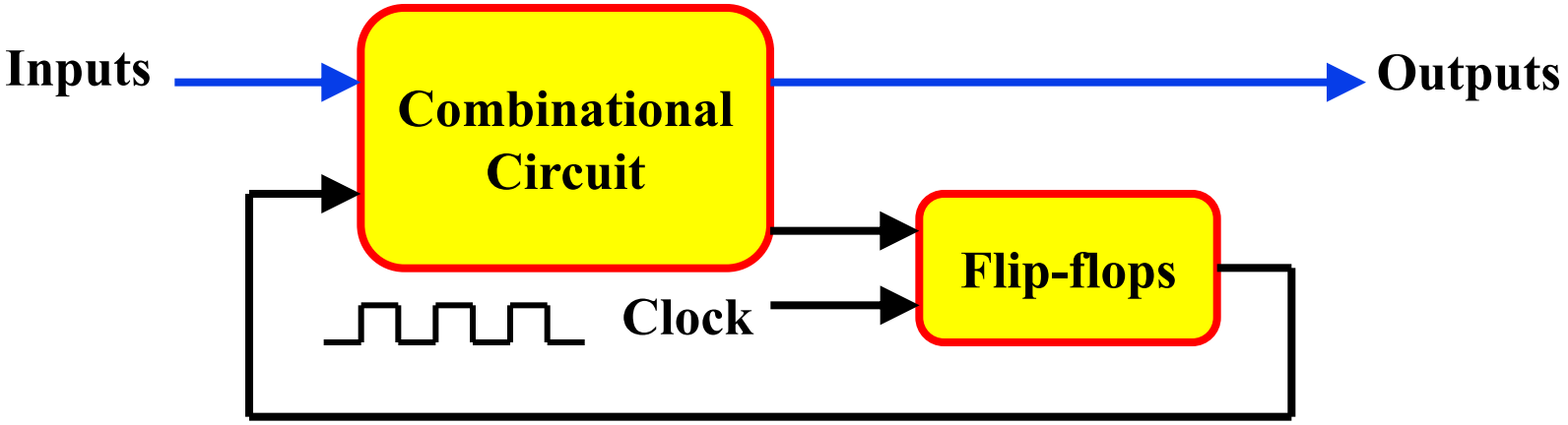
★ **UNIT-4**

Sequential Circuits

★ Asynchronous

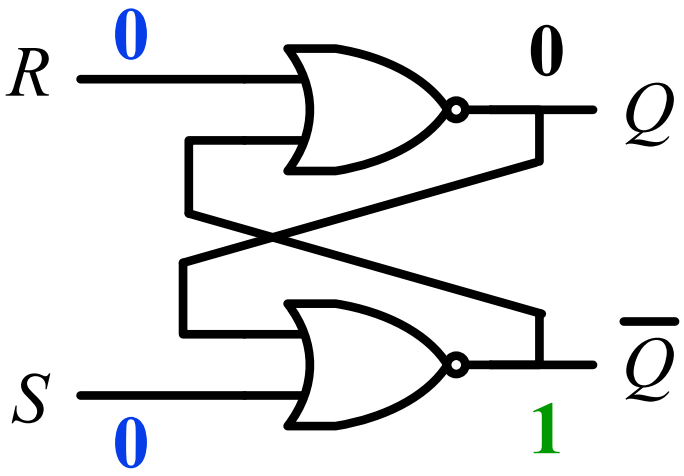


★ Synchronous



Latches

★ *SR* Latch



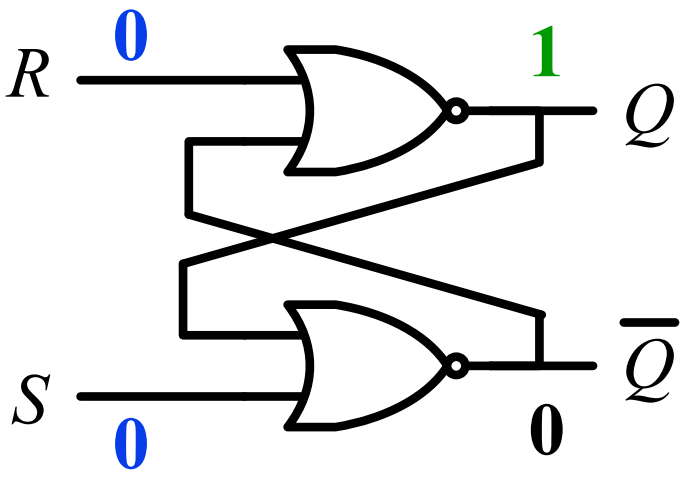
S	R	Q_0	Q	Q'
0	0	0	0	1

$Q = Q_0$

Initial Value

Latches

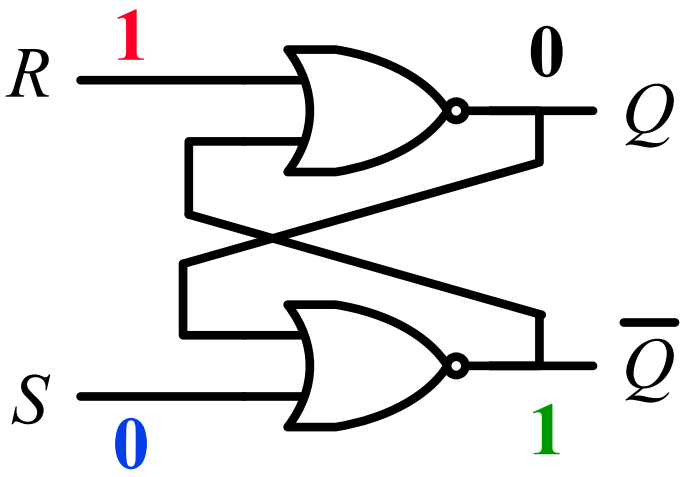
★ *SR* Latch



S R Q_0	Q	Q'	
0 0 0	0	1	$Q = Q_0$
0 0 1	1	0	$Q = Q_0$

Latches

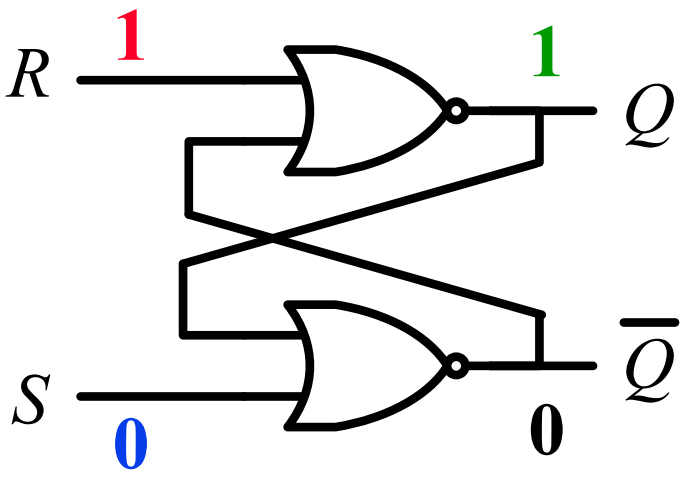
★ *SR* Latch



S	R	Q_0	Q	Q'	
0	0	0	0	1	$Q = Q_0$
0	0	1	1	0	
0	1	0	0	1	$Q = 0$

Latches

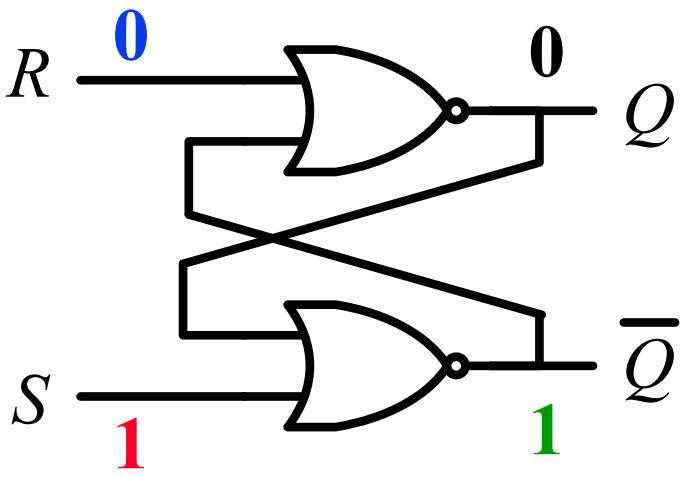
★ *SR* Latch



S	R	Q_0	Q	Q'	
0	0	0	0	1	$Q = Q_0$
0	0	1	1	0	
0	1	0	0	1	$Q = 0$
0	1	1	0	1	$Q = 0$

Latches

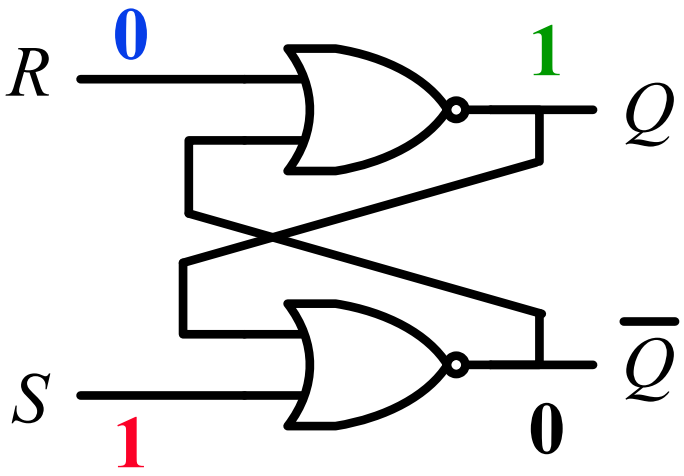
★ *SR* Latch



S	R	Q_0	Q	Q'	
0	0	0	0	1	$Q = Q_0$
0	0	1	1	0	
0	1	0	0	1	$Q = 0$
0	1	1	0	1	
1	0	0	1	0	$Q = 1$

Latches

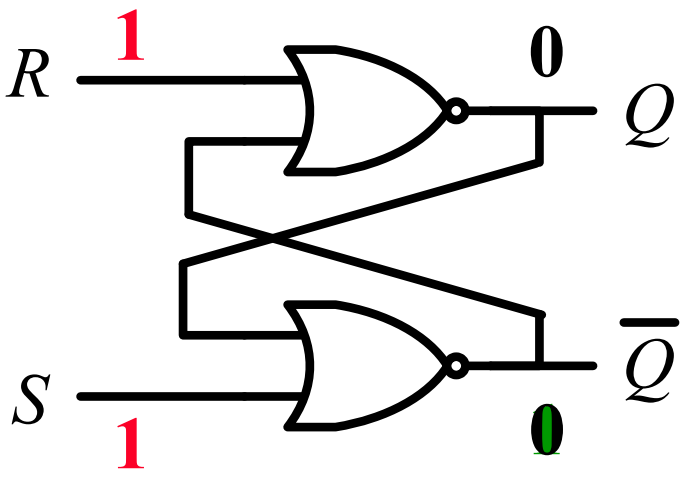
★ *SR* Latch



S	R	Q_0	Q	Q'	
0	0	0	0	1	$Q = Q_0$
0	0	1	1	0	
0	1	0	0	1	$Q = 0$
0	1	1	0	1	
1	0	0	1	0	$Q = 1$
1	0	1	1	0	$Q = 1$

Latches

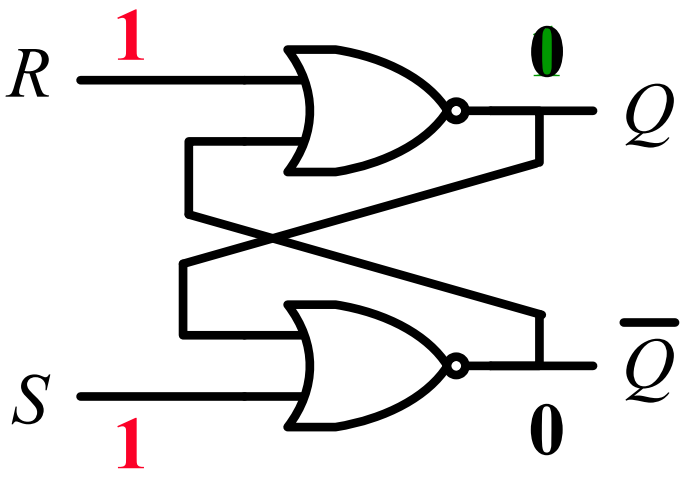
★ *SR* Latch



S	R	Q_0	Q	Q'	
0	0	0	0	1	$Q = Q_0$
0	0	1	1	0	
0	1	0	0	1	$Q = 0$
0	1	1	0	1	
1	0	0	1	0	$Q = 1$
1	0	1	1	0	
1	1	0	0	0	$Q = Q'$

Latches

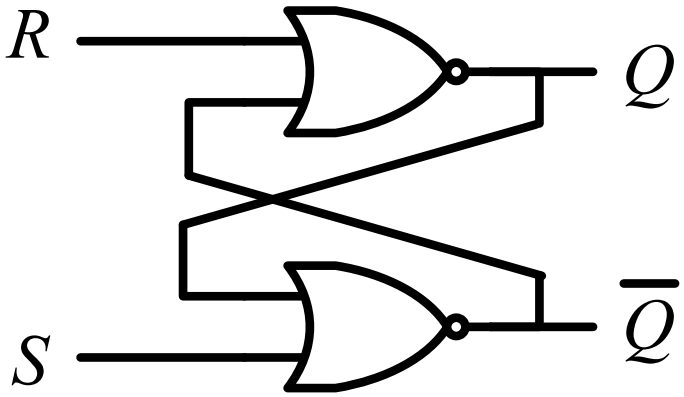
★ *SR* Latch



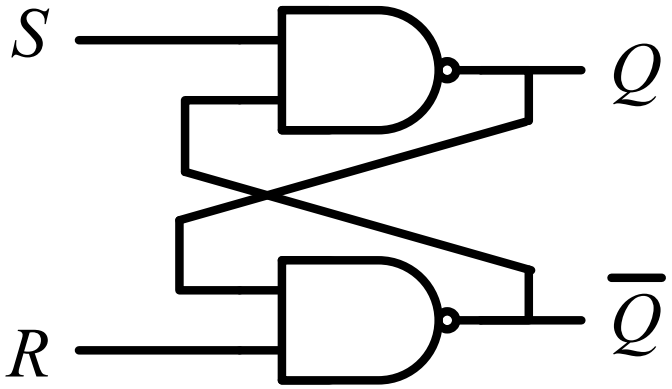
S	R	Q_0	Q	Q'	
0	0	0	0	1	$Q = Q_0$
0	0	1	1	0	
0	1	0	0	1	$Q = 0$
0	1	1	0	1	
1	0	0	1	0	$Q = 1$
1	0	1	1	0	
1	1	0	0	0	$Q = Q'$
1	1	1	0	0	$Q = Q'$

Latches

★ *SR* Latch



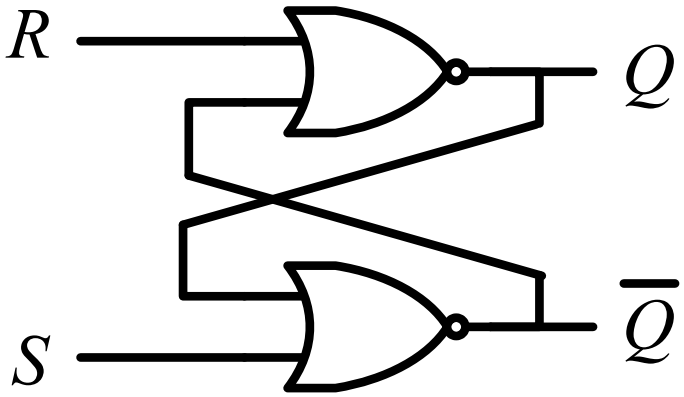
S *R*	*Q*	
0 0	Q_0	No change
0 1	0	Reset
1 0	1	Set
1 1	*Q*=*Q*'=0	Invalid



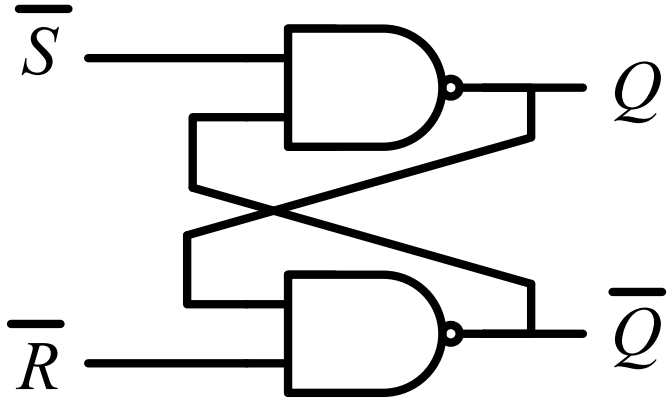
S *R*	*Q*	
0 0	*Q*=*Q*'=1	Invalid
0 1	1	Set
1 0	0	Reset
1 1	Q_0	No change

Latches

★ *SR* Latch



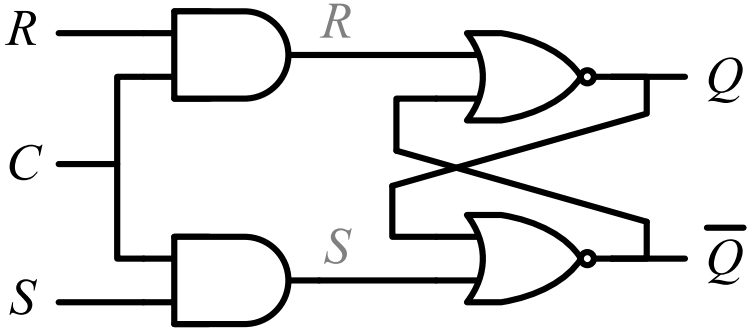
S *R*	*Q*	
0 0	Q_0	No change
0 1	0	Reset
1 0	1	Set
1 1	*Q*=*Q*'=0	Invalid

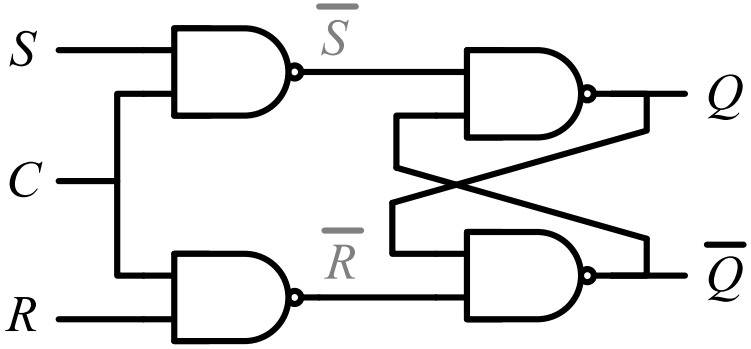


S' *R*'	*Q*	
0 0	*Q*=*Q*'=1	Invalid
0 1	1	Set
1 0	0	Reset
1 1	Q_0	No change

Controlled Latches

★ *SR* Latch with Control Input

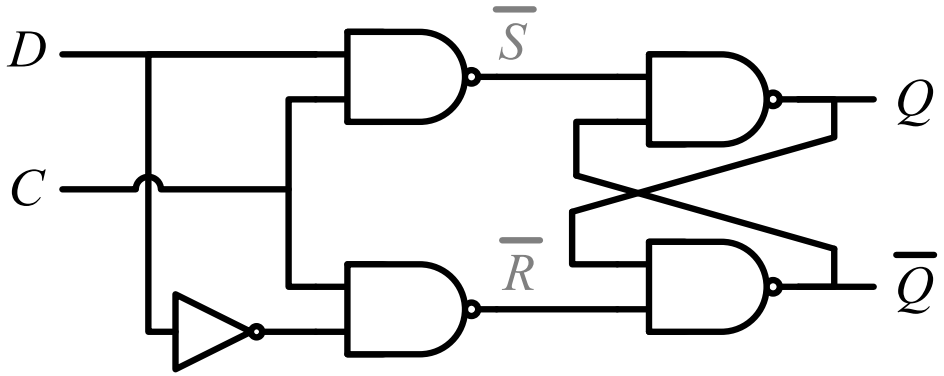




C S R	Q	
0 x x	Q_0	No change
1 0 0	Q_0	No change
1 0 1	0	Reset
1 1 0	1	Set
1 1 1	$Q=Q'$	Invalid

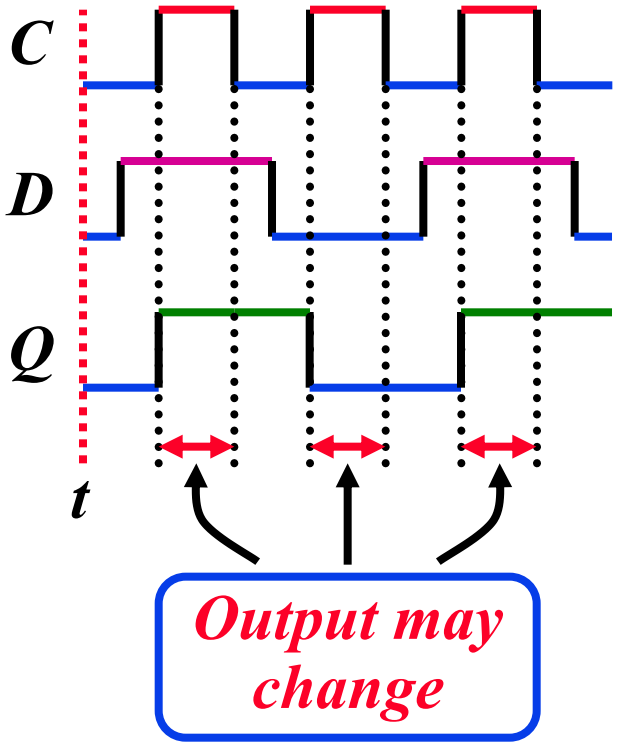
Controlled Latches

★ *D* Latch (*D* = *Data*)



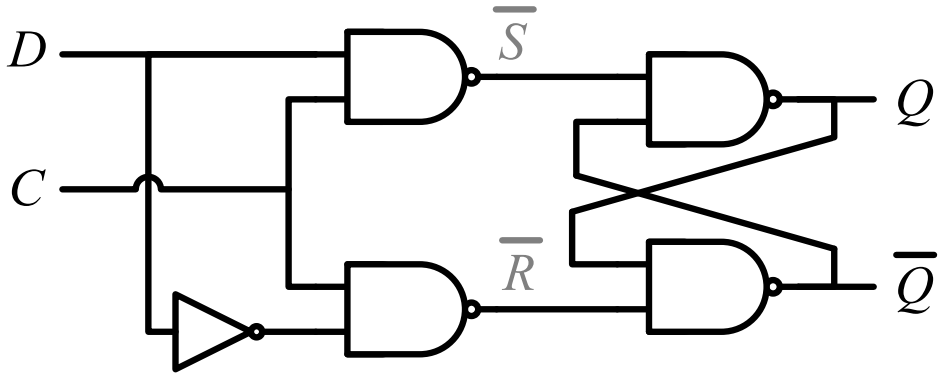
C D	Q	
0 x	Q_0	No change
1 0	0	Reset
1 1	1	Set

Timing Diagram

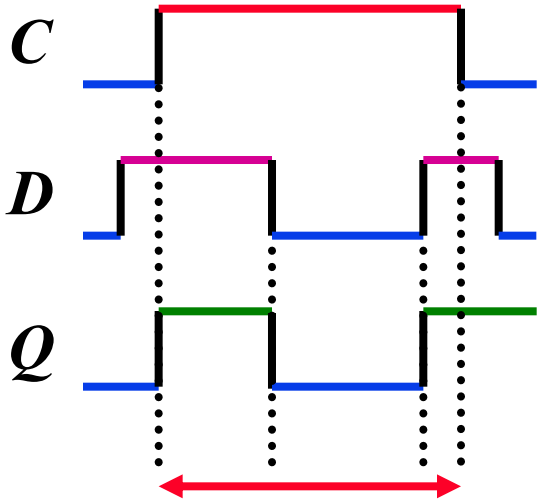


Controlled Latches

★ *D* Latch (*D* = *Data*)



Timing Diagram

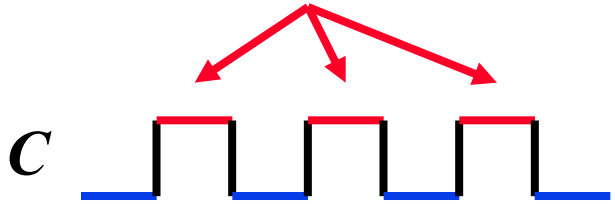


Output may change

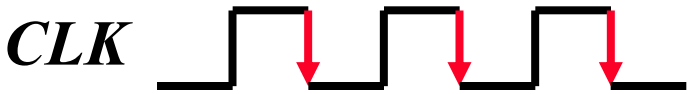
C D	Q	
0 x	Q_0	**No change**
1 0	0	**Reset**
1 1	1	**Set**

Flip-Flops

★ Controlled latches are level-triggered

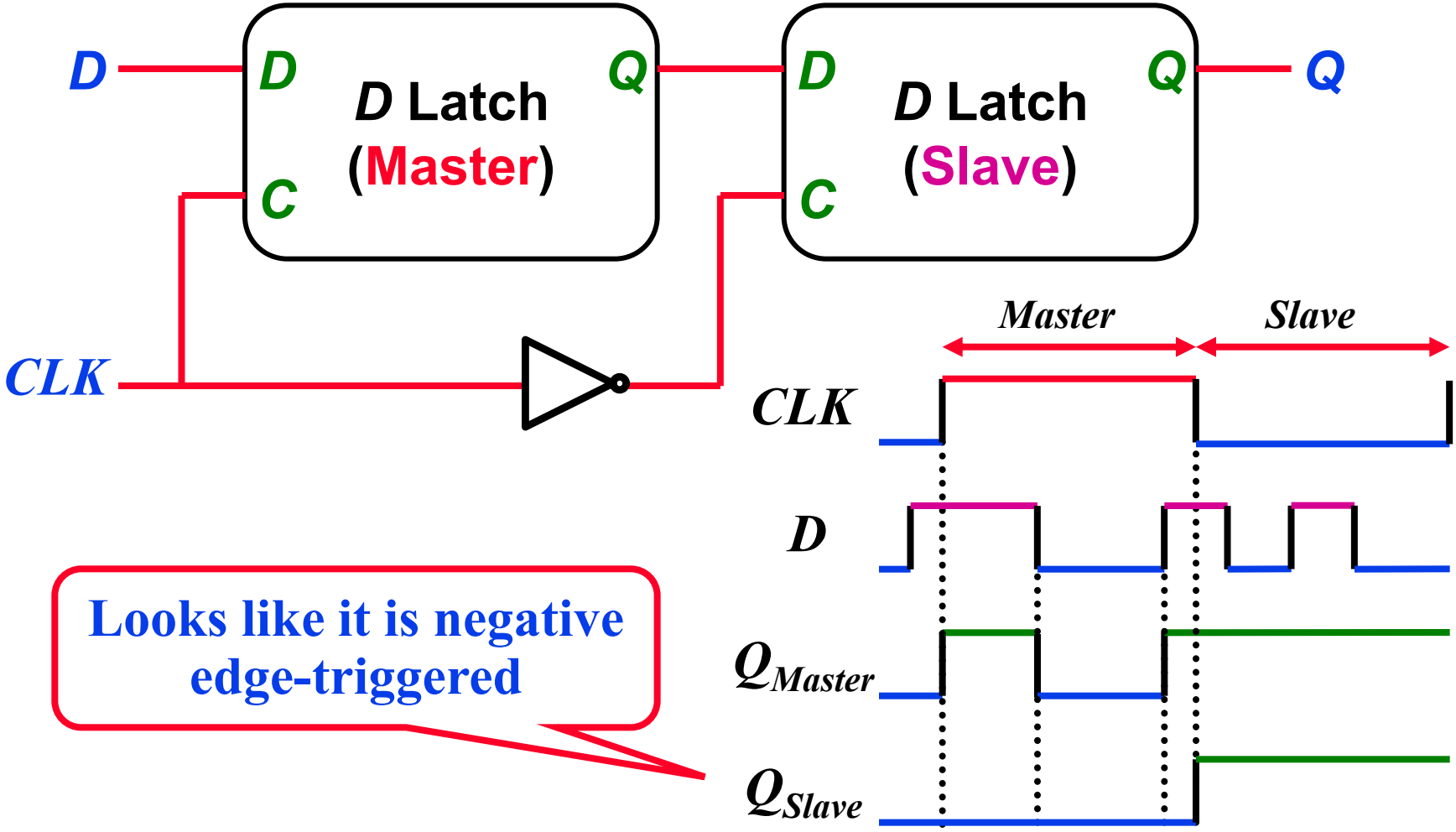


★ Flip-Flops are edge-triggered



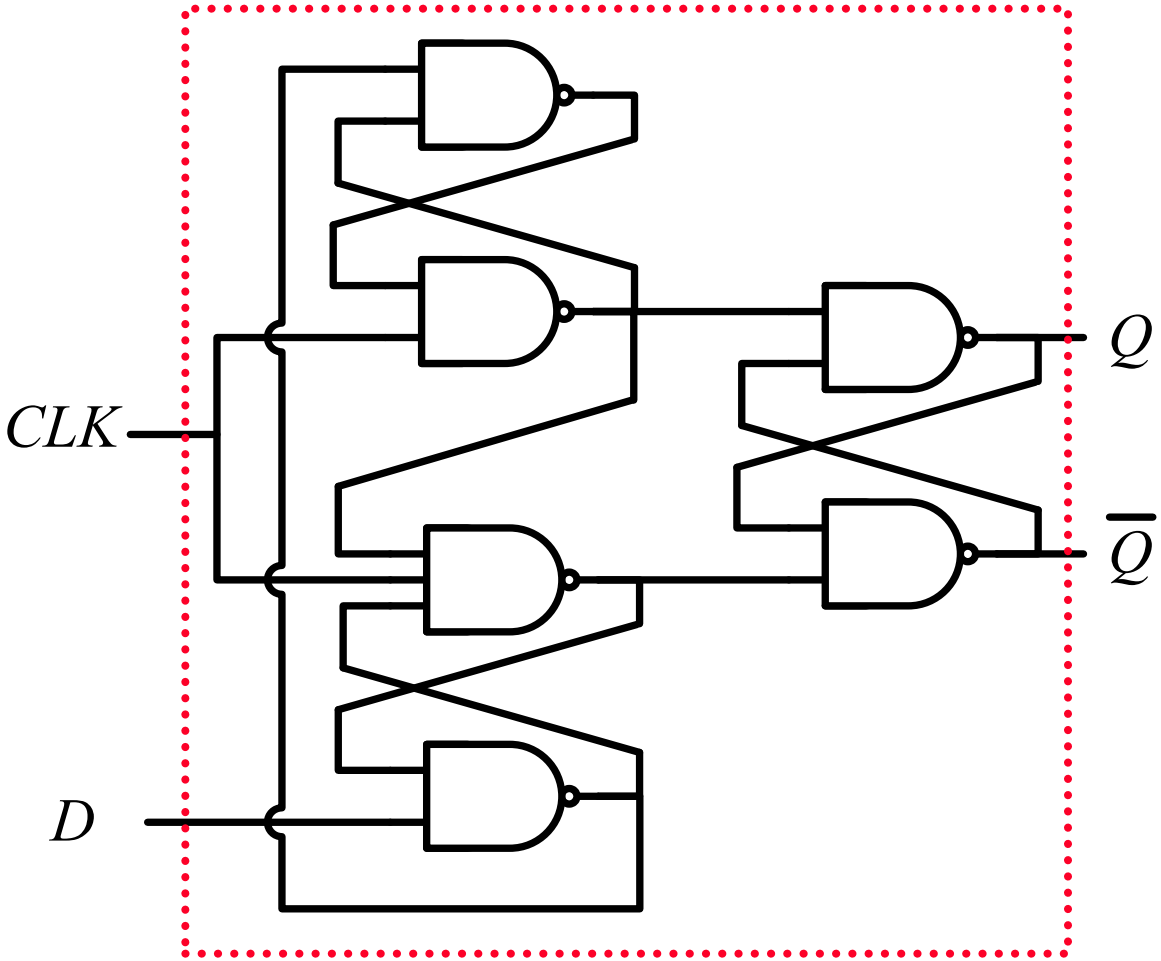
Flip-Flops

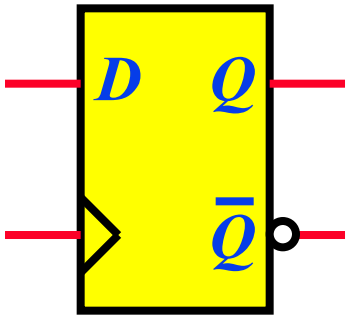
★ Master-Slave *D* Flip-Flop



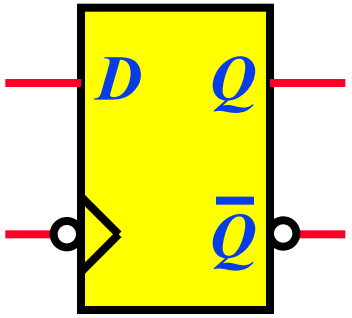
Flip-Flops

★ Edge-Triggered *D* Flip-Flop





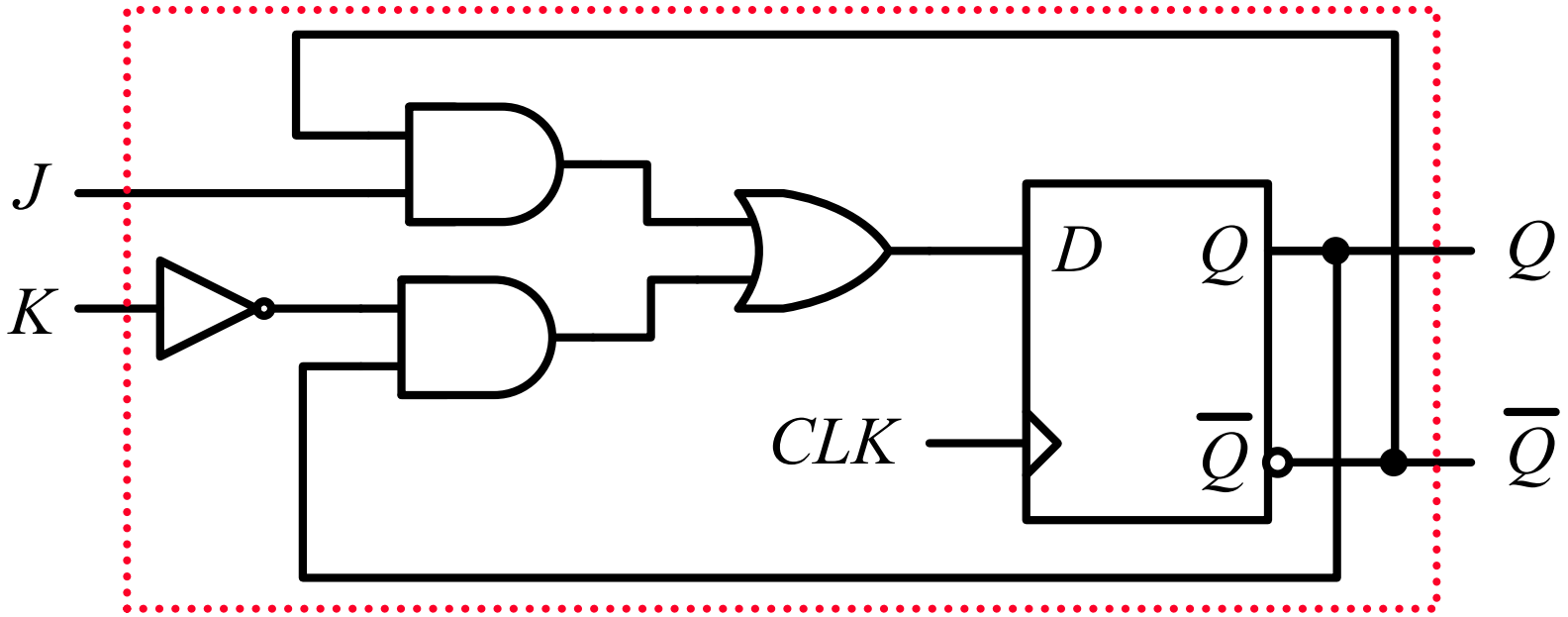
Positive Edge



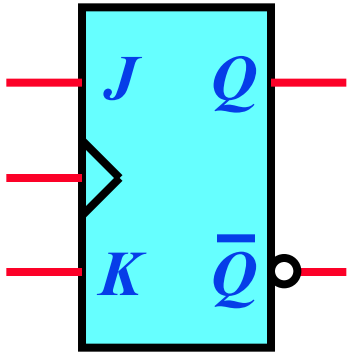
Negative Edge

Flip-Flops

★ *JK* Flip-Flop

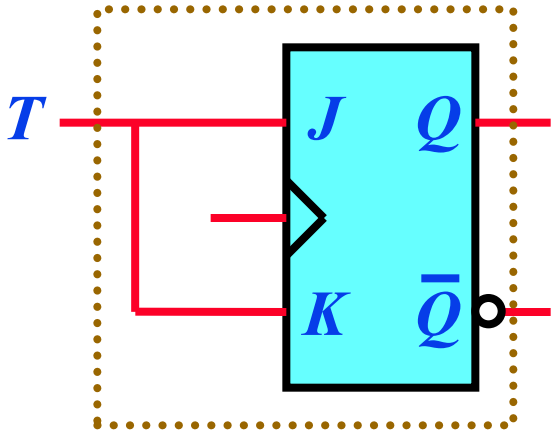


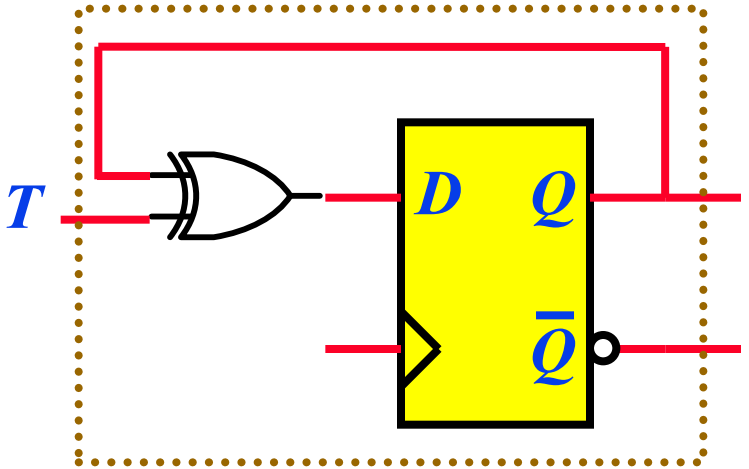
$D = JQ' + K'Q$



Flip-Flops

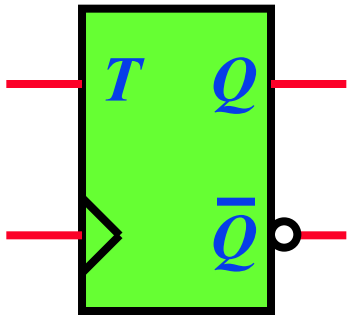
★ *T* Flip-Flop



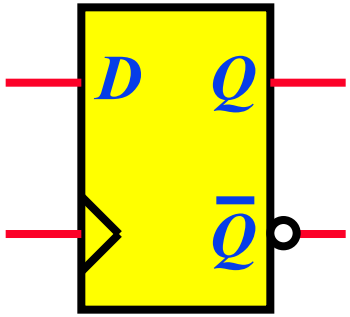


$$D = JQ' + K'Q$$

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



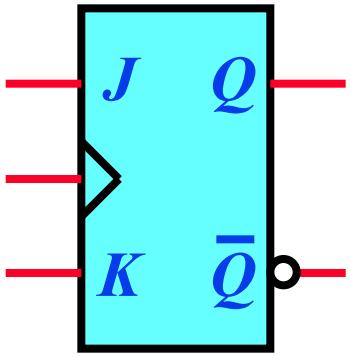
Flip-Flop Characteristic Tables



D	$Q(t+1)$
0	0
1	1

Reset

Set



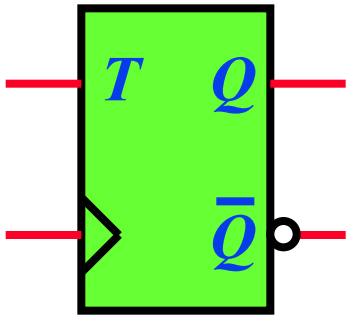
J	K	$Q(t+1)$
0	0	$Q(t)$
0	1	0
1	0	1
1	1	$Q'(t)$

No change

Reset

Set

Toggle

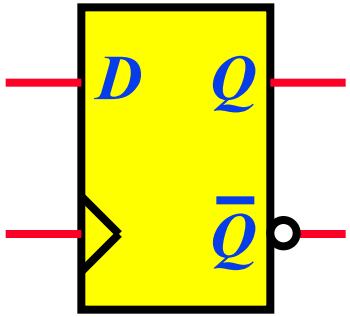


T	$Q(t+1)$
0	$Q(t)$
1	$Q'(t)$

No change

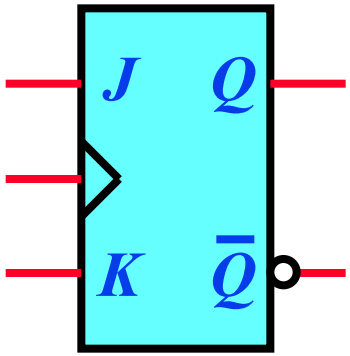
Toggle

Flip-Flop Characteristic Equations



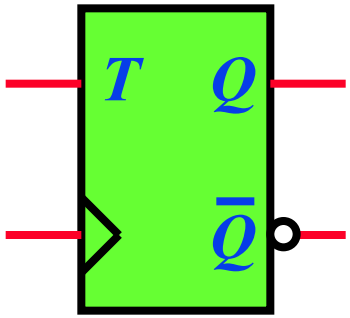
D	Q(t+1)
0	0
1	1

$$Q(t+1) = D$$



J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q'(t)

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

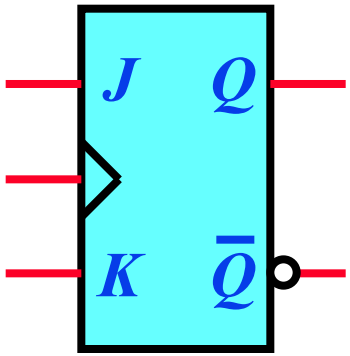


T	Q(t+1)
0	Q(t)
1	Q'(t)

$$Q(t+1) = T \oplus Q$$

Flip-Flop Characteristic Equations

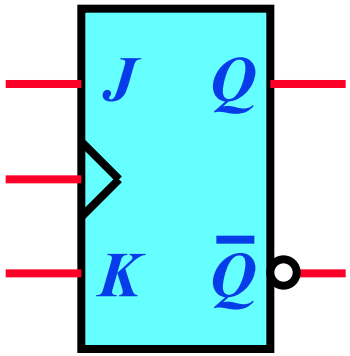
★ Analysis / Derivation



J	K	$Q(t)$	$Q(t+1)$	
0	0	0	0	No change
0	0	1	1	
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Flip-Flop Characteristic Equations

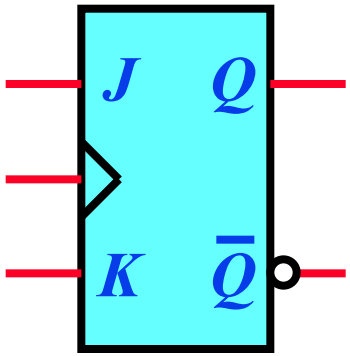
★ Analysis / Derivation



J	K	$Q(t)$	$Q(t+1)$	
0	0	0	0	No change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Flip-Flop Characteristic Equations

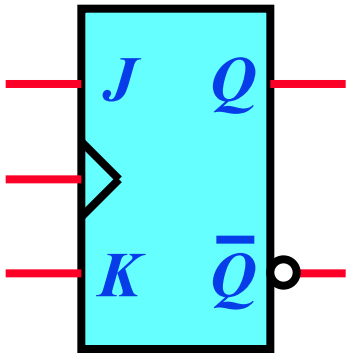
★ Analysis / Derivation



J	K	$Q(t)$	$Q(t+1)$	
0	0	0	0	No change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0		
1	1	1		

Flip-Flop Characteristic Equations

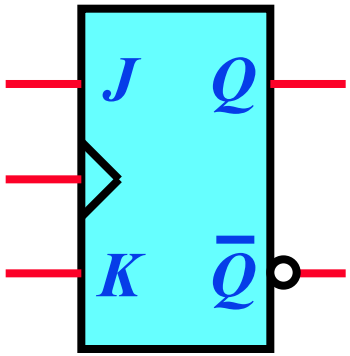
★ Analysis / Derivation



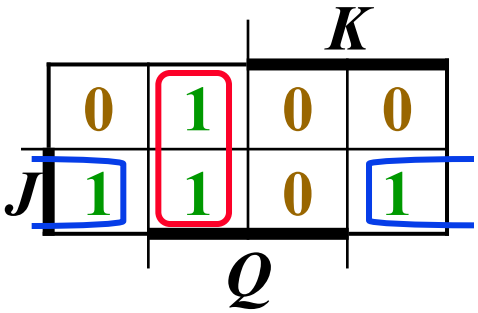
J	K	$Q(t)$	$Q(t+1)$	
0	0	0	0	No change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	1	Toggle
1	1	1	0	

Flip-Flop Characteristic Equations

★ Analysis / Derivation



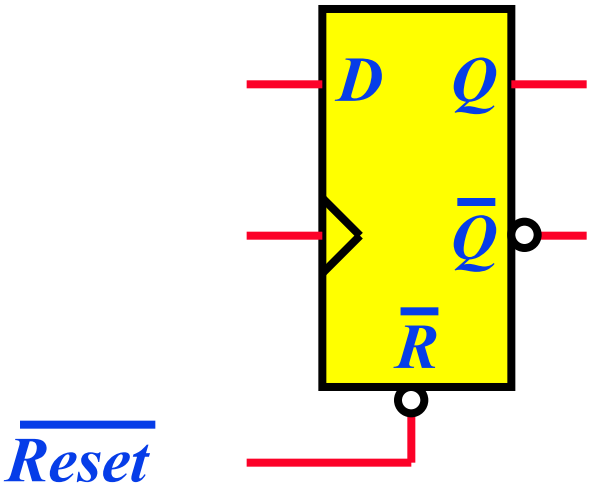
J	K	Q(t)	Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



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

Flip-Flops with Direct Inputs

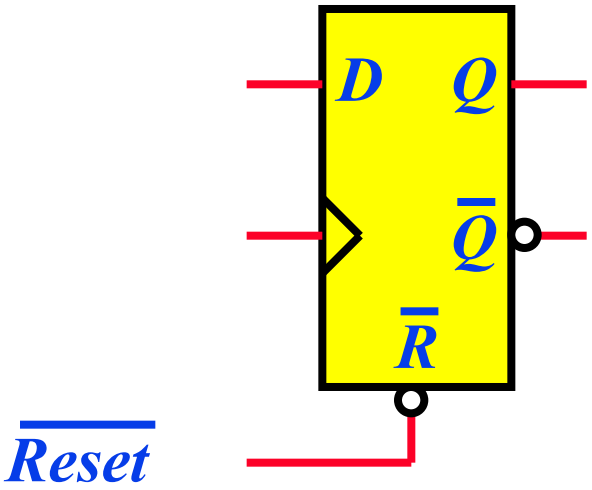
★ **Asynchronous Reset**



R'	*D*	*CLK*	*Q*(*t*+1)
0	x	x	0

Flip-Flops with Direct Inputs

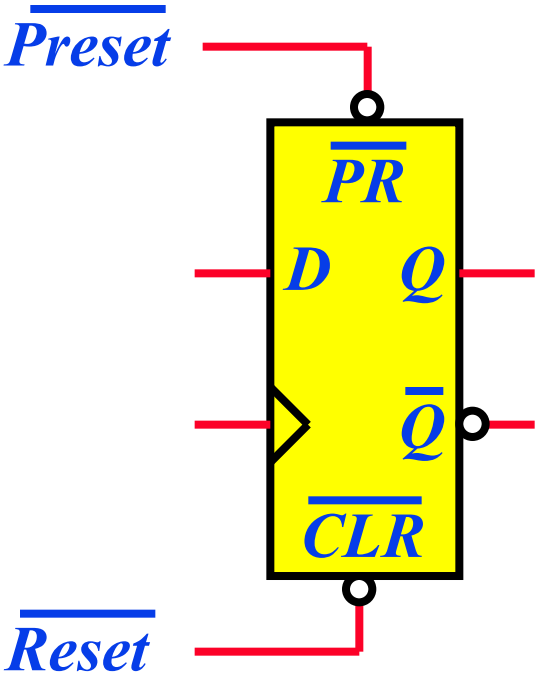
★ **Asynchronous Reset**



R'	*D*	*CLK*	*Q*(*t*+1)
0	x	x	0
1	0	↑	0
1	1	↑	1

Flip-Flops with Direct Inputs

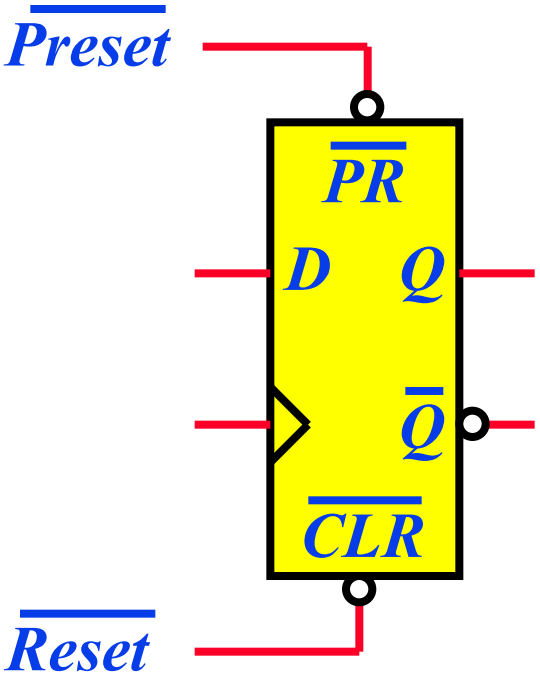
★ **Asynchronous Preset and Clear**



PR'	CLR'	D	CLK	Q(t+1)
1	0	x	x	0

Flip-Flops with Direct Inputs

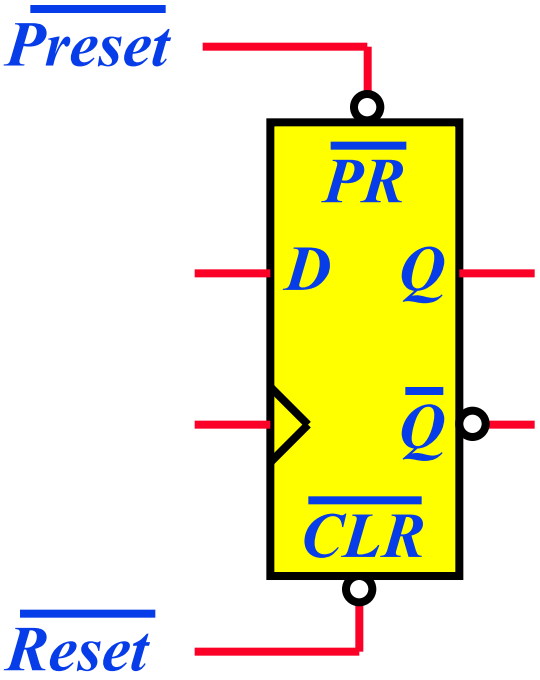
★ **Asynchronous Preset and Clear**



PR'	CLR'	D	CLK	Q(t+1)
1	0	x	x	0
0	1	x	x	1

Flip-Flops with Direct Inputs

★ **Asynchronous Preset and Clear**



PR'	CLR'	D	CLK	Q(t+1)
1	0	x	x	0
0	1	x	x	1
1	1	0	↑	0
1	1	1	↑	1

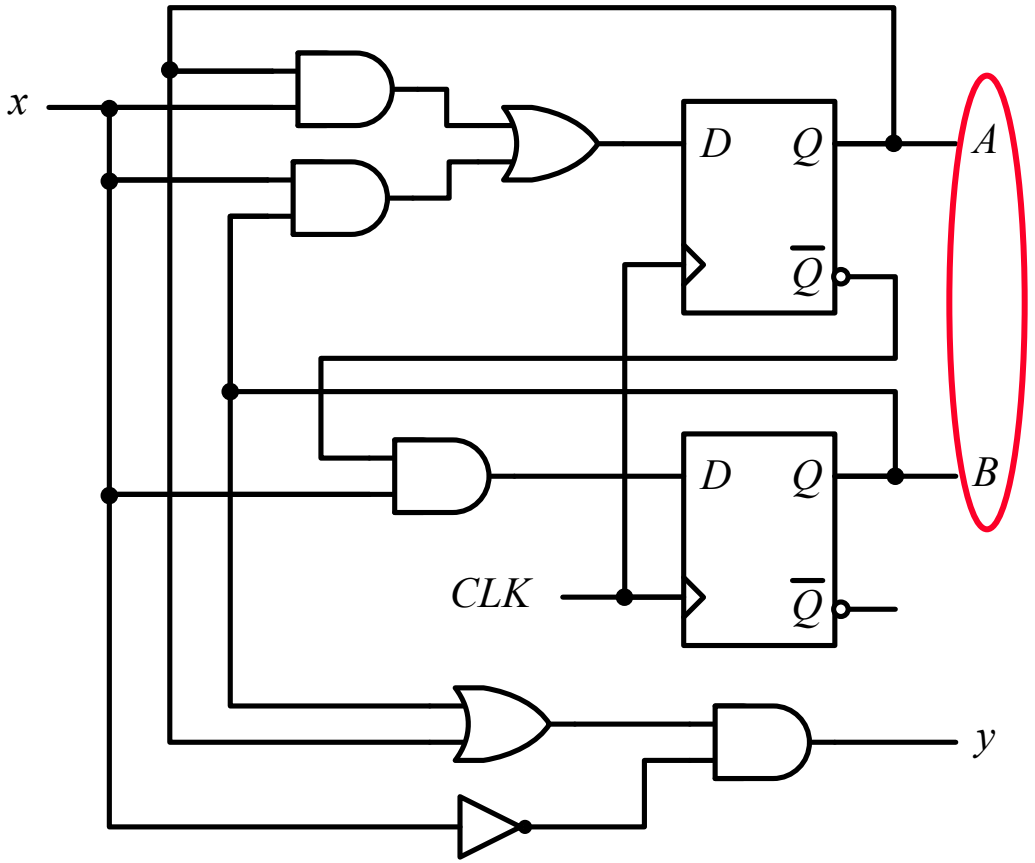
Analysis of Clocked Sequential Circuits

★ The State

- **State = Values of all Flip-Flops**

Example

$A\ B = 0\ 0$



Analysis of Clocked Sequential Circuits

★ State Equations

$$A(t+1) = D_A$$

$$= A(t)\,x(t) + B(t)\,x(t)$$

$$= A\,x + B\,x$$

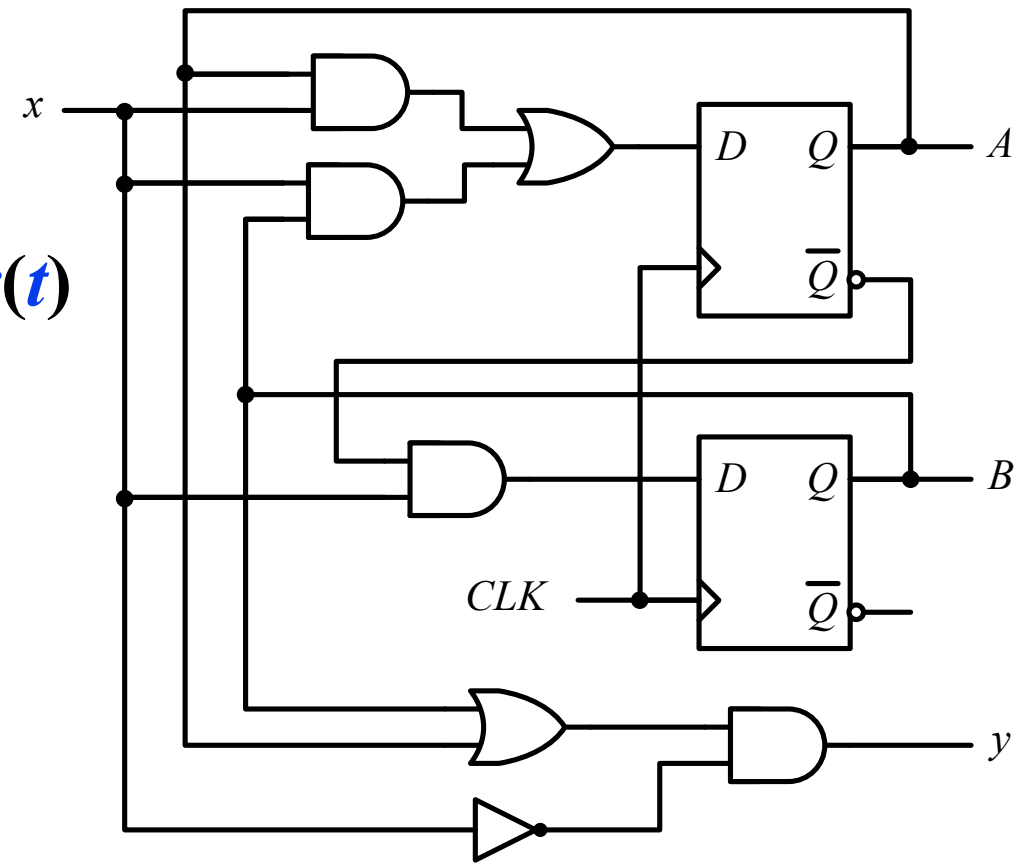
$$B(t+1) = D_B$$

$$= A'(t)\,x(t)$$

$$= A'\,x$$

$$y(t) = [A(t) + B(t)]\,x'(t)$$

$$= (A + B)\,x'$$

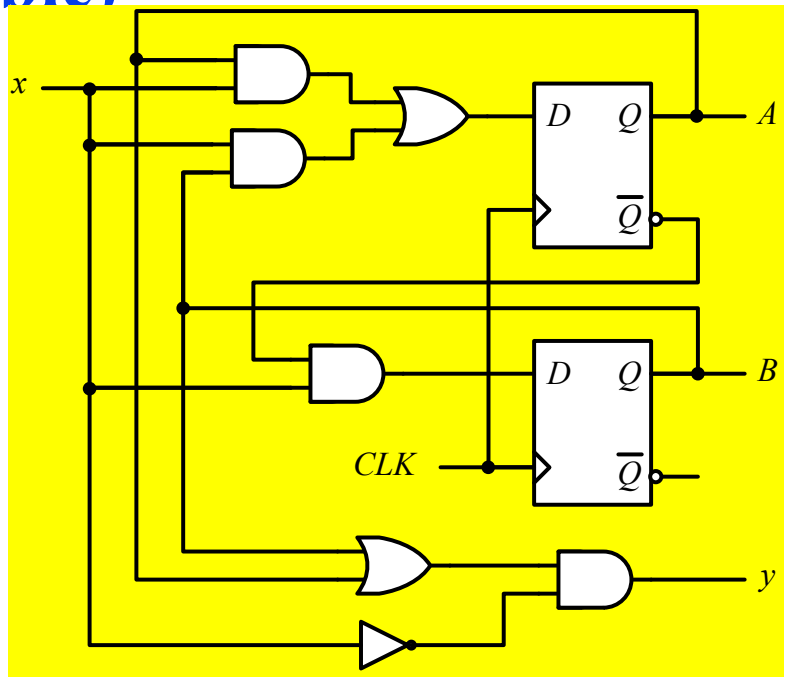


Analysis of Clocked Sequential Circuits

★ State Table (Transition Table)

Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

t *t*+1 *t*



$$A(t+1) = A\,x + B\,x$$

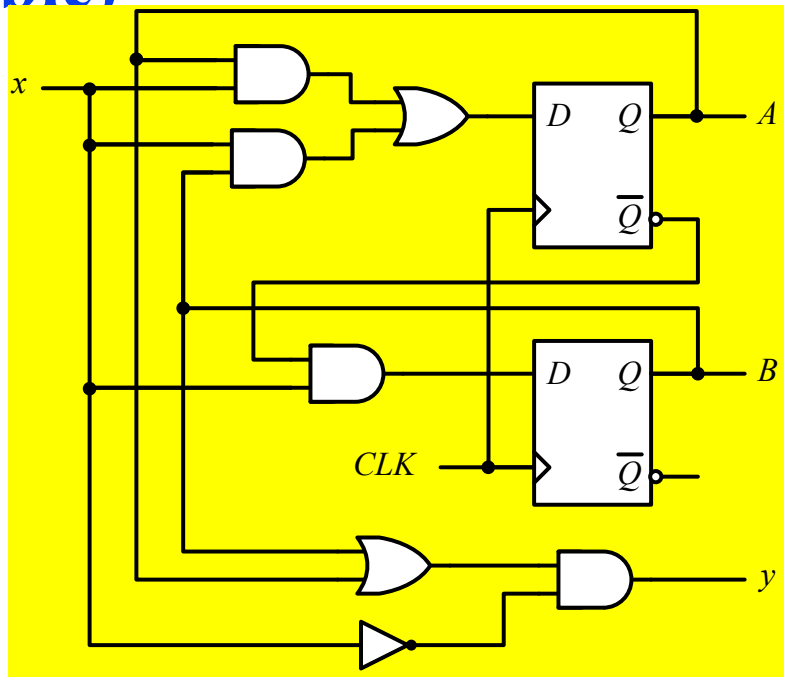
$$B(t+1) = A'\,x$$

$$y(t) = (A + B)\,x'$$

Analysis of Clocked Sequential Circuits

★ State Table (Transition Table)

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
$A\ B$	$A\ B$	$A\ B$	y	y
0 0	0 0	0 1	0	0
0 1	0 0	1 1	1	0
1 0	0 0	1 0	1	0
1 1	0 0	1 0	1	0
t	$t+1$		t	



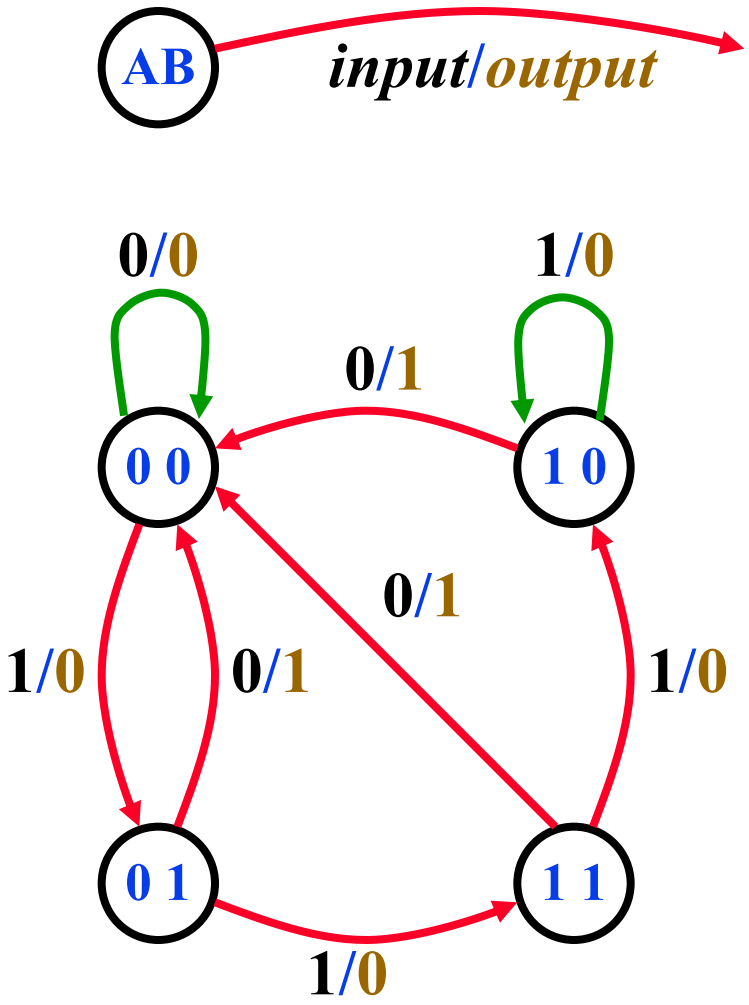
$$A(t+1) = A\,x + B\,x$$

$$B(t+1) = A'\,x$$

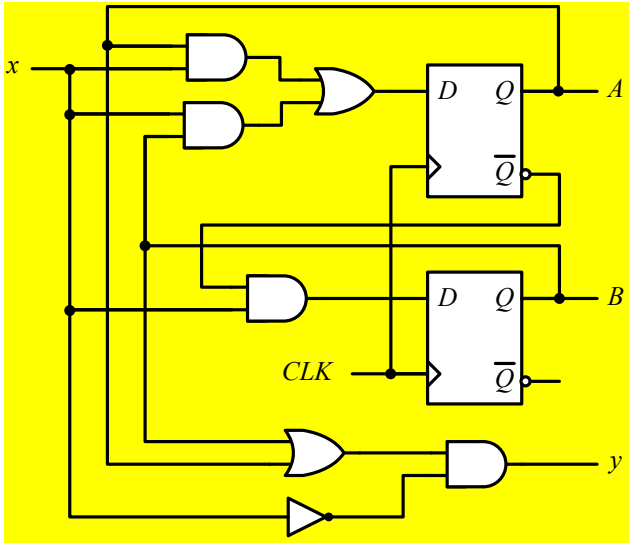
$$y(t) = (A + B)\,x'$$

Analysis of Clocked Sequential Circuits

★ State Diagram



Present State	Next State				Output	
	$x = 0$		$x = 1$		$x = 0$	$x = 1$
A B	A	B	A	B	y	y
0 0	0	0	0	1	0	0
0 1	0	0	1	1	1	0
1 0	0	0	1	0	1	0
1 1	0	0	1	0	1	0

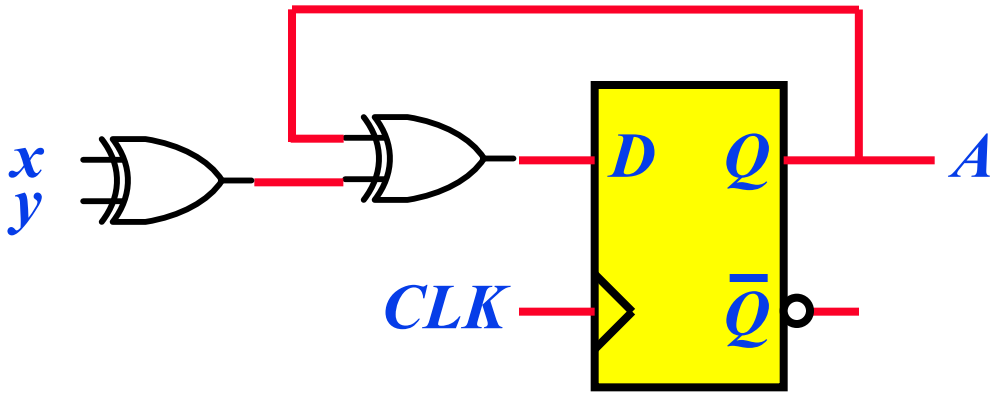


Analysis of Clocked Sequential Circuits

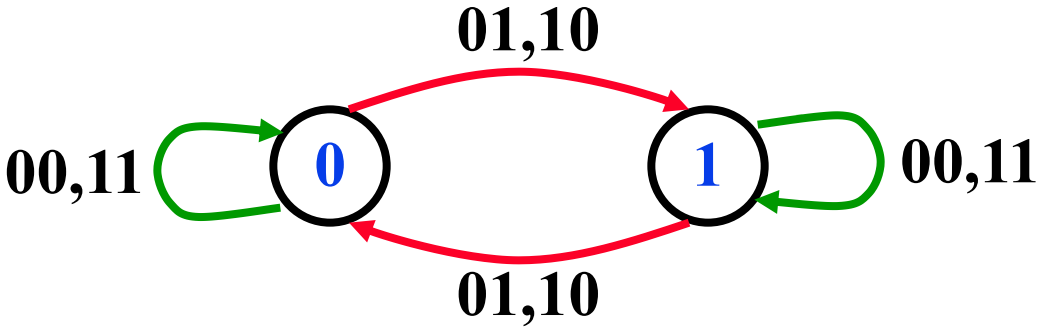
★ *D* Flip-Flops

***Example*:**

Present State	Input		Next State
A	*x*	*y*	*A*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



$$A(t+1) = D_A = A \oplus x \oplus y$$

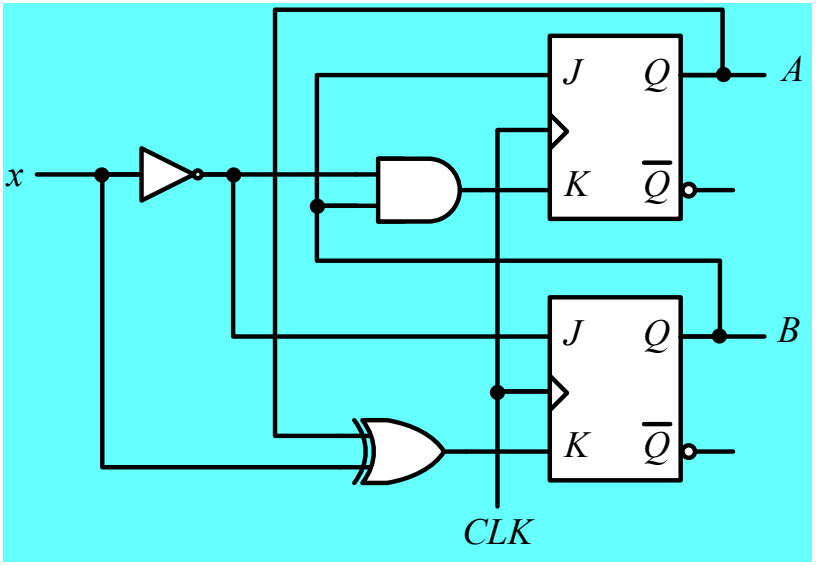


Analysis of Clocked Sequential Circuits

★ *JK* Flip-Flops

***Example*:**

Present State		I/P	Next State		Flip-Flop Inputs			
A	*B*	*x*	*A*	*B*	J_A	K_A	J_B	K_B
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0



$J_A = B$ $\quad K_A = B\,x'$

$J_B = x'$ $\quad K_B = A \oplus x$

$$A(t+1) = J_A Q'_A + K'_A Q_A$$
$$= A'B + AB' + Ax$$

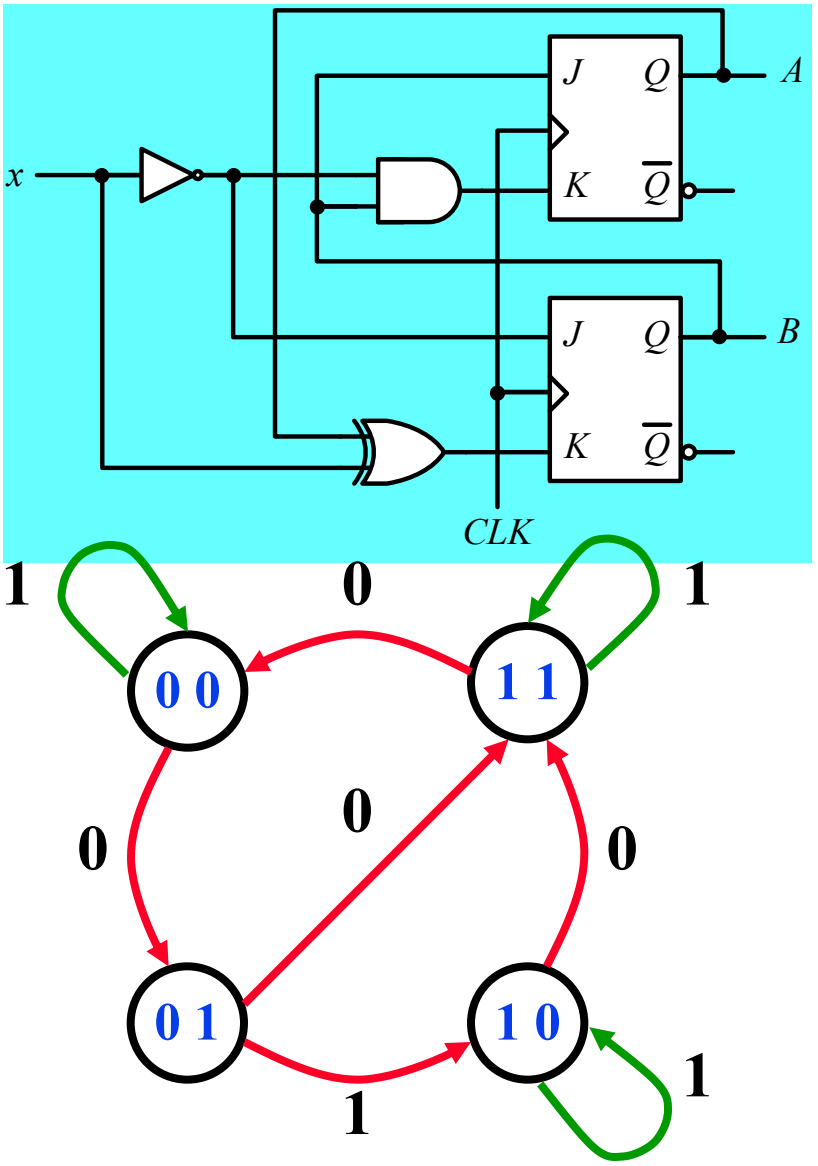
$$B(t+1) = J_B Q'_B + K'_B Q_B$$
$$= B'x' + ABx + A'Bx'$$

Analysis of Clocked Sequential Circuits

★ *JK* Flip-Flops

***Example*:**

Present State		I/P	Next State		Flip-Flop Inputs			
A	*B*	*x*	*A*	*B*	J_A	K_A	J_B	K_B
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0

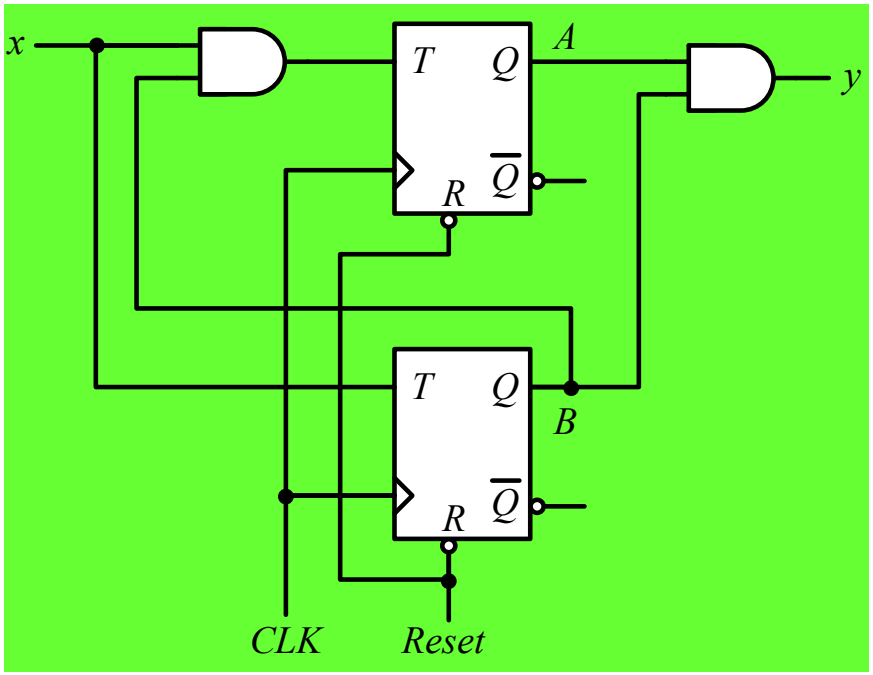


Analysis of Clocked Sequential Circuits

★ *T* Flip-Flops

***Example*:**

Present State		I/P	Next State		F.F Inputs		O/P
A	*B*	*x*	*A*	*B*	T_A	T_B	*y*
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	1	0	0	0
0	1	1	1	0	1	1	0
1	0	0	1	0	0	0	0
1	0	1	1	1	0	1	0
1	1	0	1	1	0	0	1
1	1	1	0	0	1	1	1



$$T_A = B\,x \qquad T_B = x$$

$$y = A\,B$$

$$A(t+1) = T_A Q'_A + T'_A Q_A$$
$$= AB' + Ax' + A'Bx$$

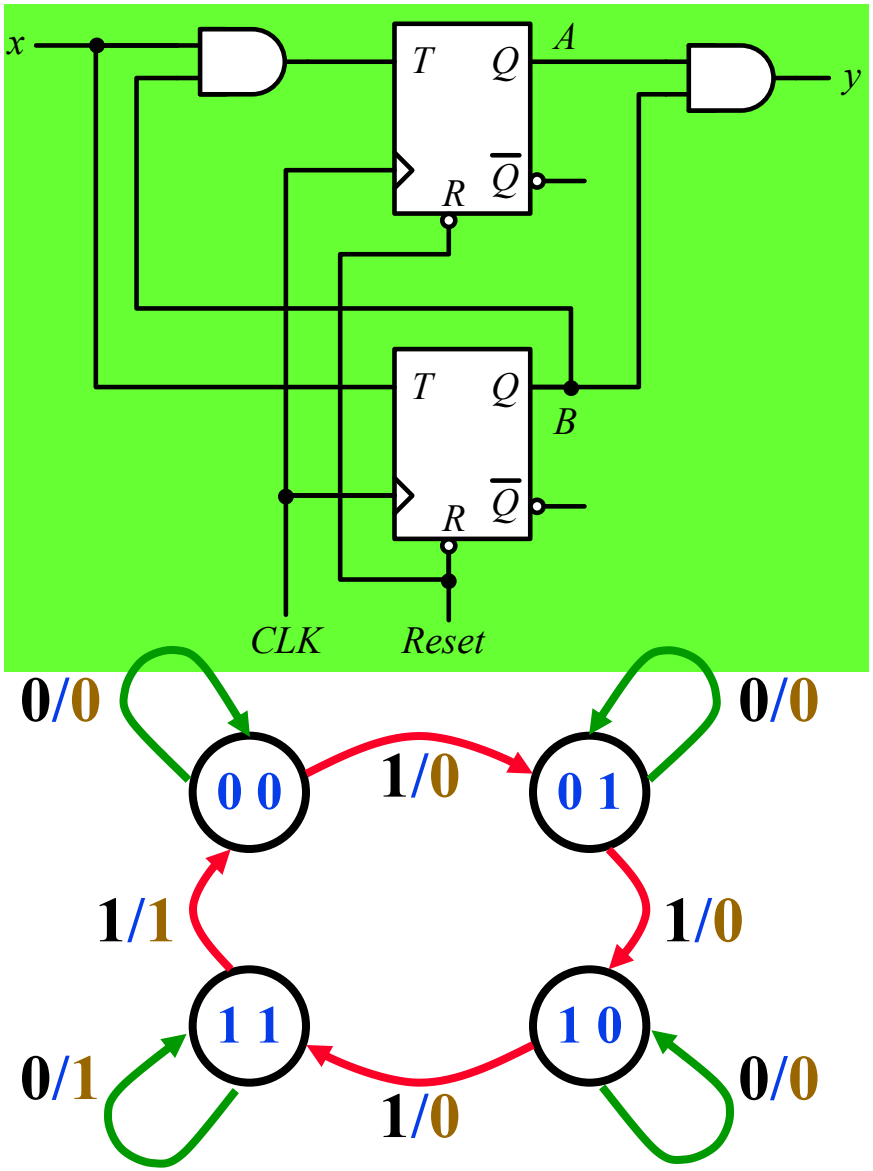
$$B(t+1) = T_B Q'_B + T'_B Q_B$$
$$= x \oplus B$$

Analysis of Clocked Sequential Circuits

★ *T* Flip-Flops

Example:

Present State		I/P	Next State		F.F Inputs		O/P
A	*B*	*x*	*A*	*B*	T_A	T_B	*y*
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	1	0	0	0
0	1	1	1	0	1	1	0
1	0	0	1	0	0	0	0
1	0	1	1	1	0	1	0
1	1	0	1	1	0	0	1
1	1	1	0	0	1	1	1



State Reduction and Assignment

★ **State Reduction**
Reductions on the number of flip-flops and the number of gates.

- **A reduction in the number of states may result in a reduction in the number of flip-flops.**
- **An example state diagram showing in Fig. 5.25.**

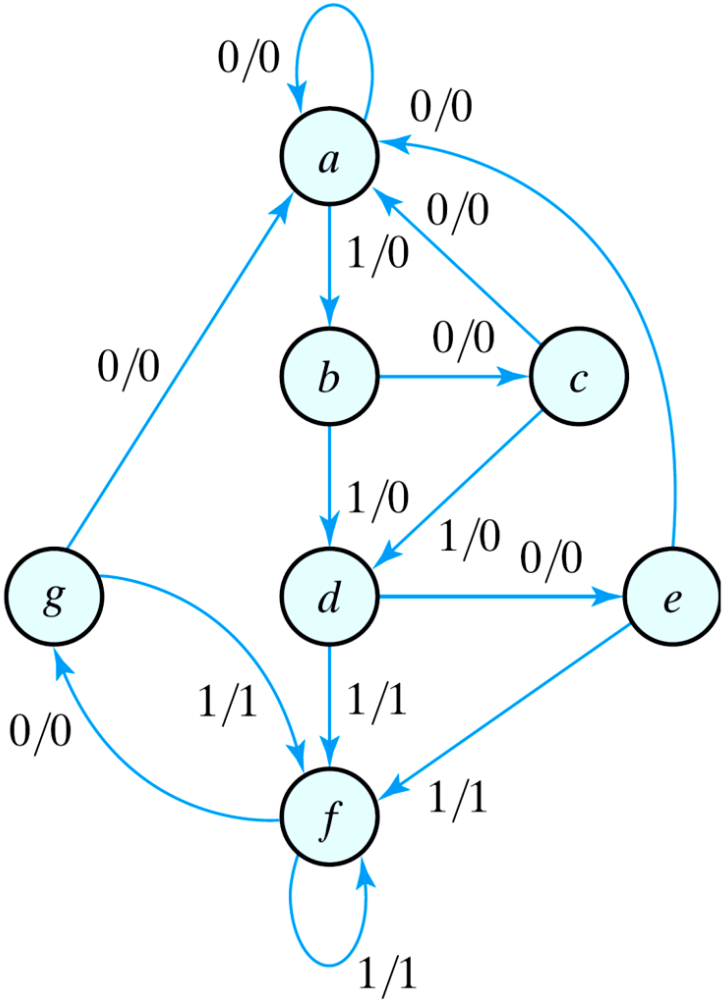


Fig. 5.25 State diagram

State Reduction

State:	a	a	b	c	d	e	f	f	g	f	g	a
Input:	0	1	0	1	0	1	1	0	1	0	0	
Output:	0	0	0	0	0	1	1	0	1	0	0	

- **Only the input-output sequences are important.**
- **Two circuits are equivalent**
  - **Have identical outputs for all input sequences;**
  - **The number of states is not important.**

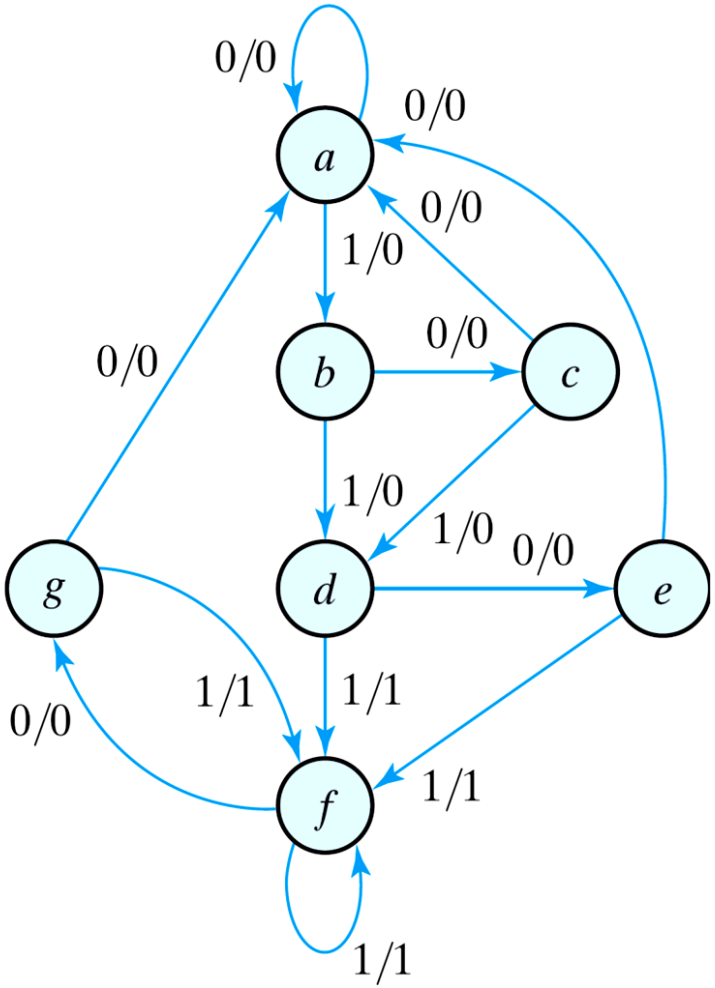


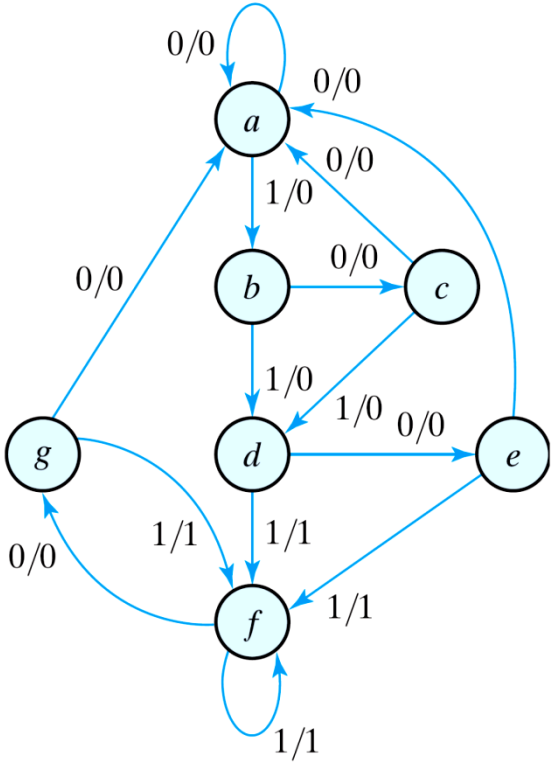
Fig. 5.25 State diagram

★ Equivalent states

- **Two states are said to be equivalent**
  - **For each member of the set of inputs, they give exactly the same output and send the circuit to the same state or to an equivalent state.**

Table 5.6
State Table

	Next State		Output	
Present State	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1



★ Reducing the state table

- $e = g$ (remove g);
- $d = f$ (remove f);

Table 5.7
Reducing the State Table

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	e	f	0	1

- **The reduced finite state machine**

Table 5.8
Reduced State Table

	Next State		Output	
Present State	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

State:	a	a	b	c	d	e	d	d	e	d	e	a
Input:	0	1	0	1	0	1	1	0	1	0	0	
Output:	0	0	0	0	0	1	1	0	1	0	0	

- **The checking of each pair of states for possible equivalence can be done systematically using Implication Table.**
- **The unused states are treated as don't-care condition ⇒ fewer combinational gates.**

Table 5.8
Reduced State Table

	Next State		Output	
Present State	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

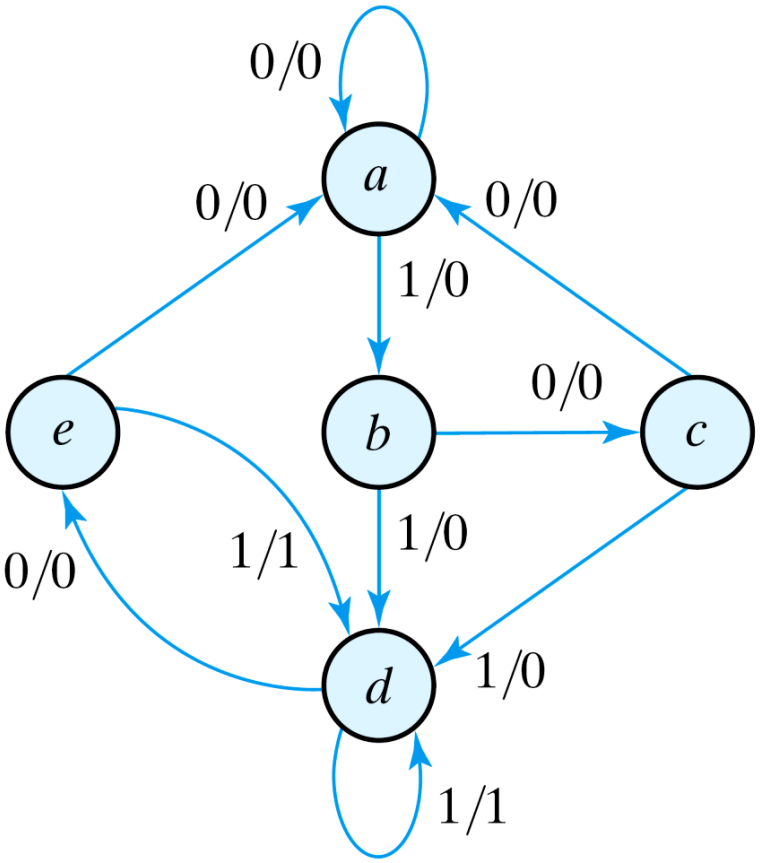


Fig. 5.26 Reduced State diagram

Implication Table

- The state-reduction procedure for completely specified state tables is based on the algorithm that two states in a state table can be combined into one if they can be shown to be equivalent. There are occasions when a pair of states do not have the same next states, but, nonetheless, go to equivalent next states. Consider the following state table:

Present State	Next State		Output	
	x = 0	x = 1	x = 0	x = 1
a	*c*	*b*	0	1
b	*d*	*a*	0	1
c	*a*	*d*	1	0
d	*b*	*d*	1	0

- (*a*, *b*) *imply* (*c*, *d*) and (*c*, *d*) *imply* (*a*, *b*). Both pairs of states are equivalent; i.e., *a* and *b* are equivalent as well as *c* and *d*.

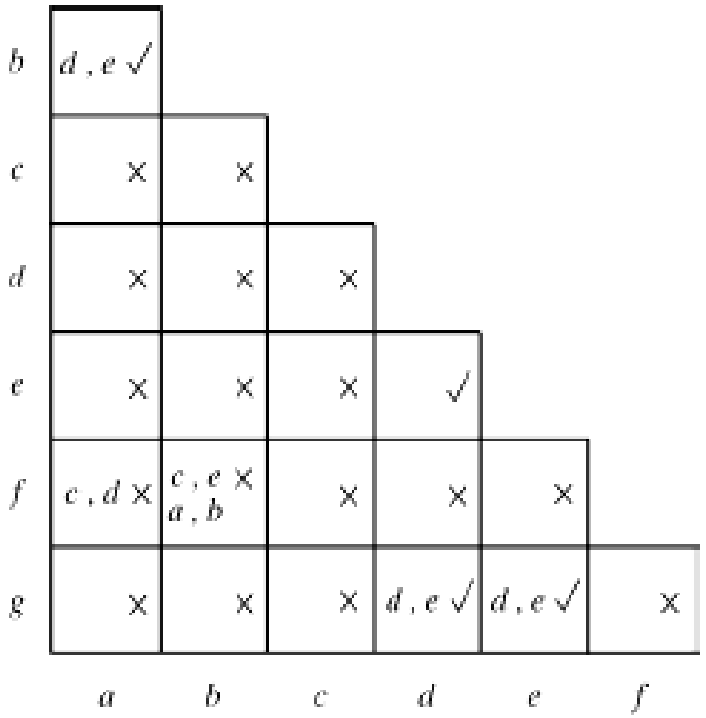
Implication Table

- The checking of each pair of states for possible equivalence in a table with a large number of states can be done systematically by means of an *implication table*. This a chart that consists of squares, one for every possible pair of states, that provide spaces for listing any possible implied states. Consider the following state table:

Present State	Next State		Output	
	x = 0	x = 1	x = 0	x = 1
a	*d*	6	0	0
b	*e*	*a*	0	0
c	*g*	*f*	0	1
d	*a*	*d*	1	0
e	*a*	*d*	1	0
f	*c*	*b*	0	0
g	*a*	*e*	1	0

Implication Table

★ The implication table is:



Implication Table

- On the left side along the vertical are listed all the states defined in the state table except the last, and across the bottom horizontally are listed all the states except the last.
- The states that are not equivalent are marked with a ‘x’ in the corresponding square, whereas their equivalence is recorded with a ‘√’.
- Some of the squares have entries of implied states that must be further investigated to determine whether they are equivalent or not.
- The *step-by-step* procedure of filling in the squares is as follows:

1. Place a cross in any square corresponding to a pair of states whose outputs are not equal for every input.
2. Enter in the remaining squares the pairs of states that are implied by the pair of states representing the squares. We do that by starting from the top square in the left column and going down and then proceeding with the next column to the right.

Implication Table

3. Make successive passes through the table to determine whether any additional squares should be marked with a 'x'. A square in the table is crossed out if it contains at least one implied pair that is not equivalent.

4. Finally, all the squares that have no crosses are recorded with check marks. The equivalent states are: (a, b), (d, e), (d, g), (e, g).

We now combine pairs of states into larger groups of equivalent states. The last three pairs can be combined into a set of three equivalent states (d, e, g) because each one of the states in the group is equivalent to the other two. The final partition of these states consists of the equivalent states found from the implication table, together with all the remaining states in the state table that are not equivalent to any other state:

$$(a, b)\ (c)\ (d, e, g)\ (f)$$

The reduced state table is:

Implication Table

Present State	Next State		Output	
	x = 0	x = 1	x = 0	x = 1
a	a	c	0	0
b	a	d	0	1
c	c	d	1	0
d	c	c	0	0

State Assignment

★ State Assignment

★ To minimize the cost of the combinational circuits.

- Three possible binary state assignments. (*m* states need *n*-bits, where $2^n > m$)

Table 5.9

Three Possible Binary State Assignments

State	Assignment 1, Binary	Assignment 2, Gray Code	Assignment 3, One-Hot
a	000	000	00001
b	001	001	00010
c	010	011	00100
d	011	010	01000
e	100	110	10000

- **Any binary number assignment is satisfactory as long as each state is assigned a unique number.**
- **Use binary assignment 1.**

Table 5.10
Reduced State Table with Binary Assignment 1

	Next State		Output	
Present State	$x = 0$	$x = 1$	$x = 0$	$x = 1$
000	000	001	0	0
001	010	011	0	0
010	000	011	0	0
011	100	011	0	1
100	000	011	0	1

Design Procedure

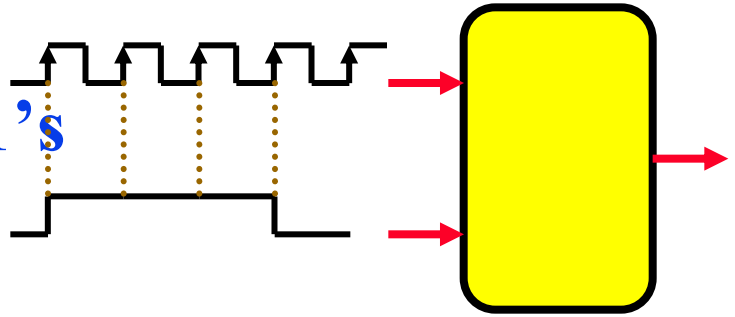
★ Design Procedure for sequential circuit

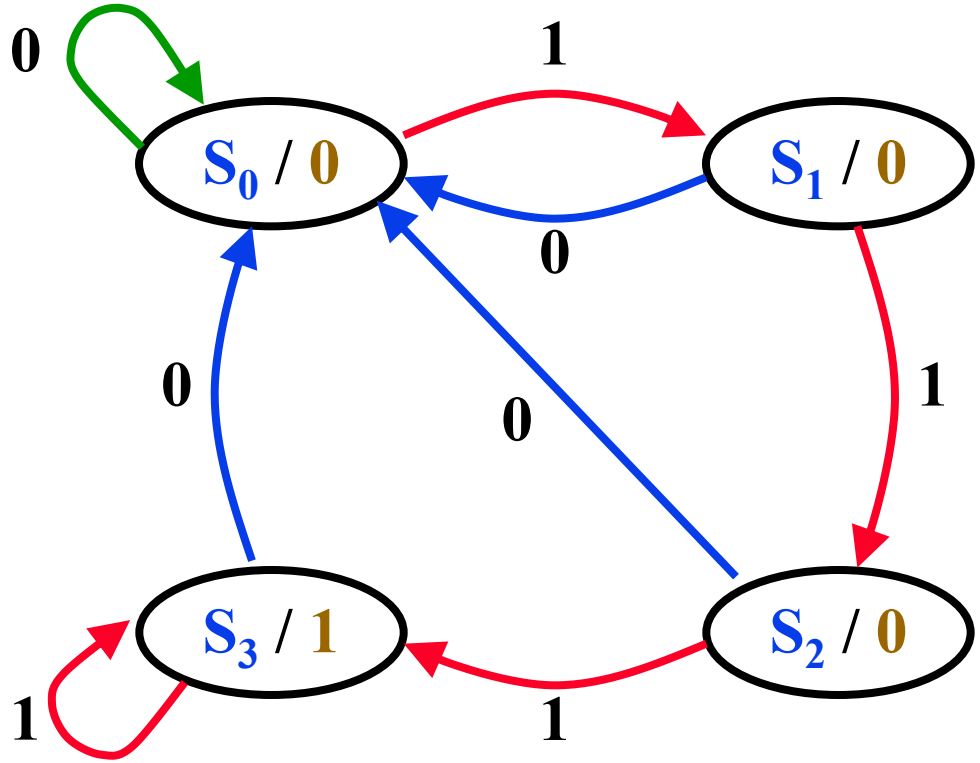
- The word description of the circuit behavior to get a state diagram;
- State reduction if necessary;
- Assign binary values to the states;
- Obtain the binary-coded state table;
- Choose the type of flip-flops;
- Derive the simplified flip-flop input equations and output equations;
- Draw the logic diagram;

Design of Clocked Sequential Circuits

★ *Example*:

Detect 3 or more consecutive 1's



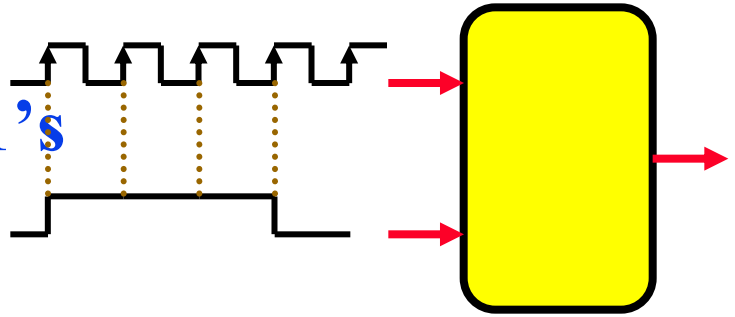


State	*A* *B*
S_0	0 0
S_1	0 1
S_2	1 0
S_3	1 1

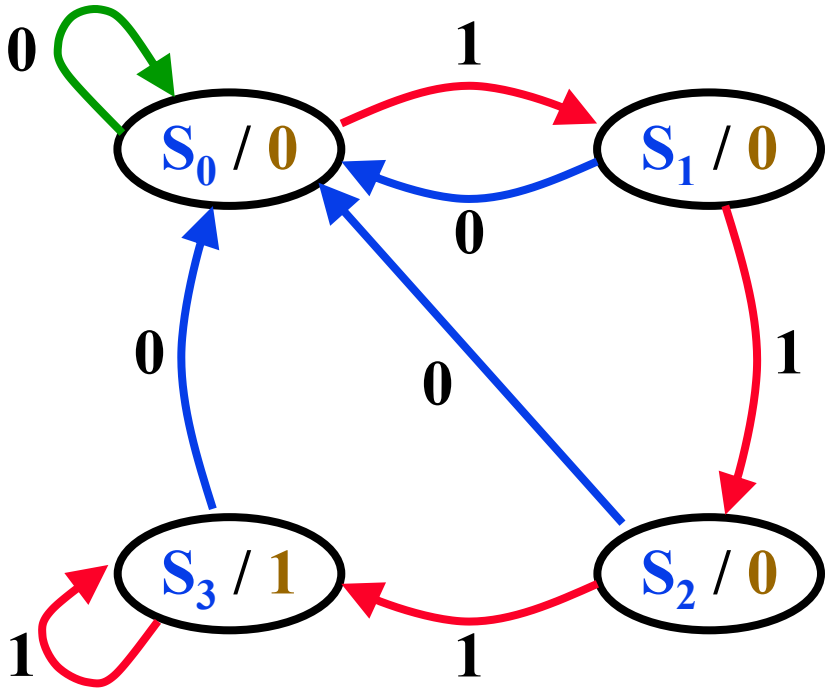
Design of Clocked Sequential Circuits

★ *Example*:

Detect 3 or more consecutive 1's



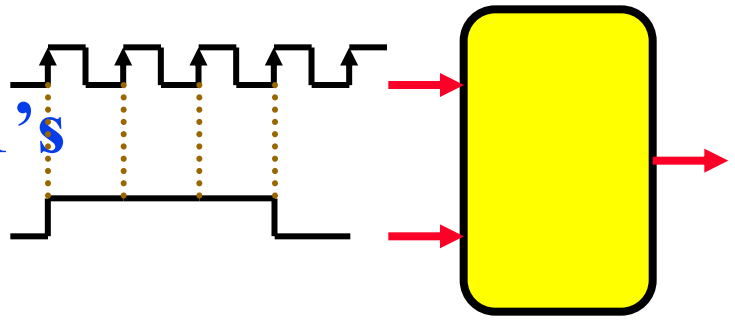
Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1



Design of Clocked Sequential Circuits

★ *Example*:

Detect 3 or more consecutive 1's



Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

Synthesis using *D* Flip-Flops

$$A(t+1) = D_A(A, B, x)$$

$$= \sum (3, 5, 7)$$

$$B(t+1) = D_B(A, B, x)$$

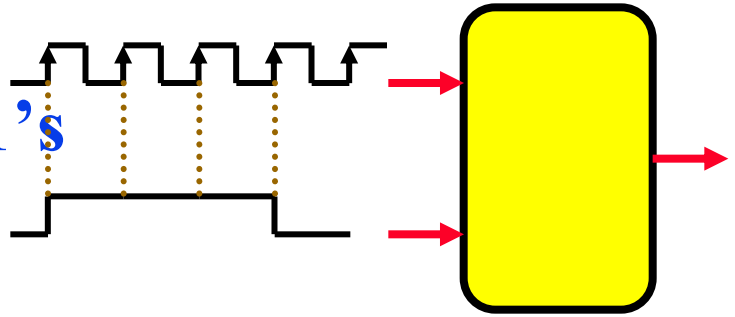
$$= \sum (1, 5, 7)$$

$$y(A, B, x) = \sum (6, 7)$$

Design of Clocked Sequential Circuits with *D* F.F.

★ *Example*:

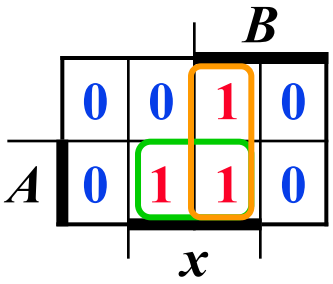
Detect 3 or more consecutive 1's



Synthesis using *D* Flip-Flops

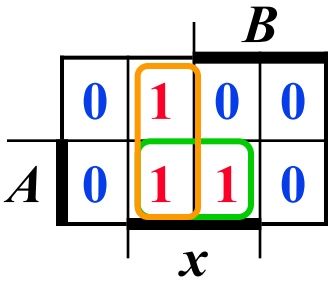
$$D_A(A, B, x) = \sum(3, 5, 7)$$

$$= A\,x + B\,x$$



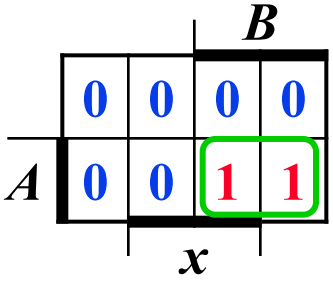
$$D_B(A, B, x) = \sum(1, 5, 7)$$

$$= A\,x + B'x$$



$$y(A, B, x) = \sum(6, 7)$$

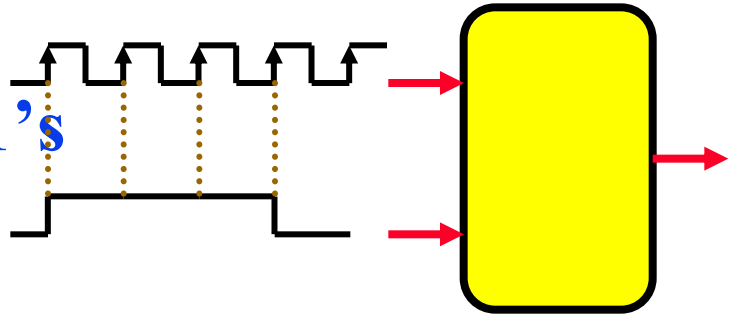
$$= A\,B$$



Design of Clocked Sequential Circuits with *D* F.F.

★ ***Example*:**

Detect 3 or more consecutive 1's

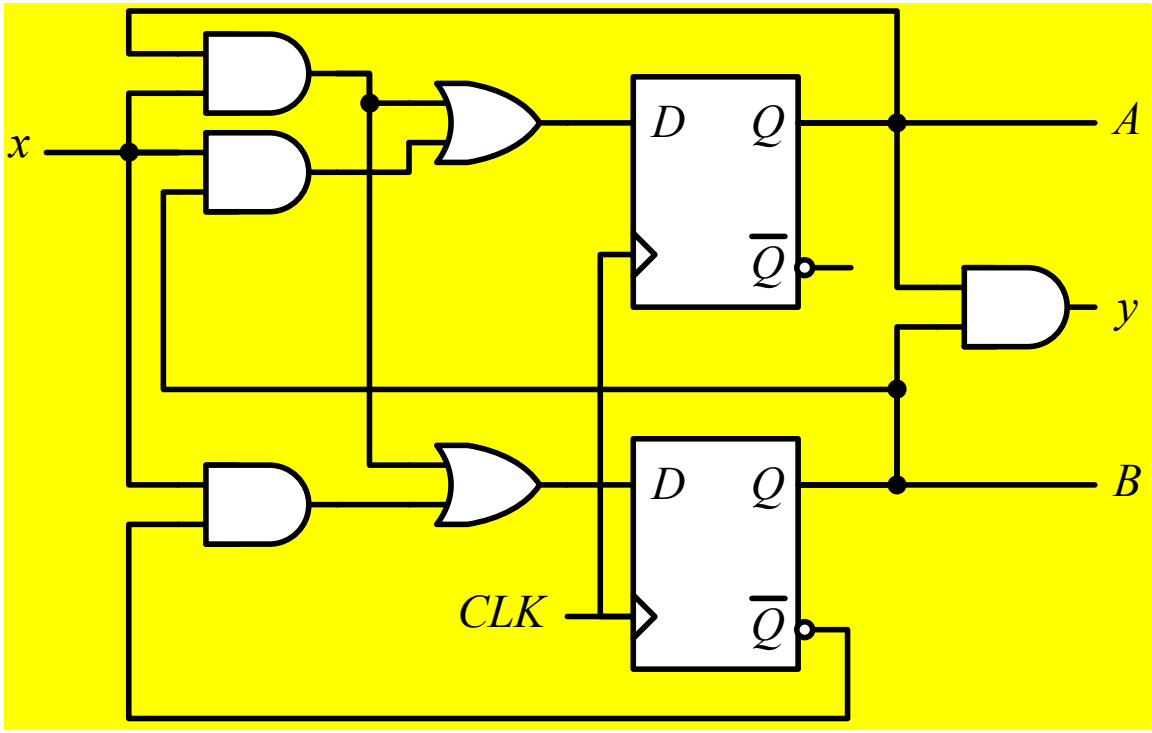


Synthesis using *D* Flip-Flops

$$D_A = A\,x + B\,x$$

$$D_B = A\,x + B'\,x$$

$$y = A\,B$$



Flip-Flop Excitation Tables

Present State	Next State	F.F. Input
$Q(t)$	$Q(t+1)$	D
0	0	0
0	1	1
1	0	0
1	1	1

Present State	Next State	F.F. Input	
$Q(t)$	$Q(t+1)$	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

0 0 (No change)
0 1 (Reset)

1 0 (Set)
1 1 (Toggle)

0 1 (Reset)
1 1 (Toggle)

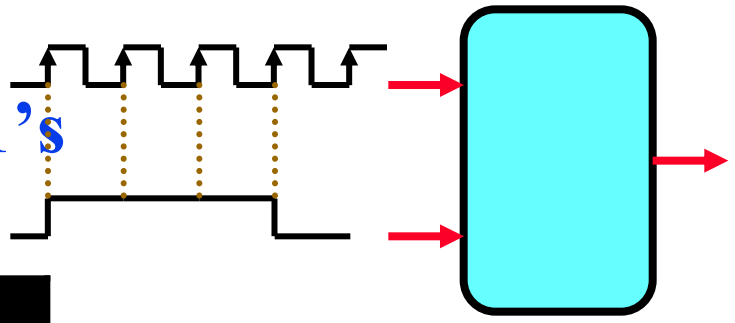
0 0 (No change)
1 0 (Set)

$Q(t)$	$Q(t+1)$	T
0	0	0
0	1	1
1	0	1
1	1	0

Design of Clocked Sequential Circuits with *JK* F.F.

★ *Example*:

Detect 3 or more consecutive 1's



Present State		Input	Next State		Flip-Flop Inputs			
A	*B*	*x*	*A*	*B*	J_A	K_A	J_B	K_B
0	0	0	0	0	0	x	0	x
0	0	1	0	1	0	x	1	x
0	1	0	0	0	0	x	x	1
0	1	1	1	0	1	x	x	1
1	0	0	0	0	x	1	0	x
1	0	1	1	1	x	0	1	x
1	1	0	0	0	x	1	x	1
1	1	1	1	1	x	0	x	0

Synthesis using *JK* F.F.

$J_A(A, B, x) = \sum(3)$

$d_{JA}(A, B, x) = \sum(4,5,6,7)$

$K_A(A, B, x) = \sum(4, 6)$

$d_{KA}(A, B, x) = \sum(0,1,2,3)$

$J_B(A, B, x) = \sum(1, 5)$

$d_{JB}(A, B, x) = \sum(2,3,6,7)$

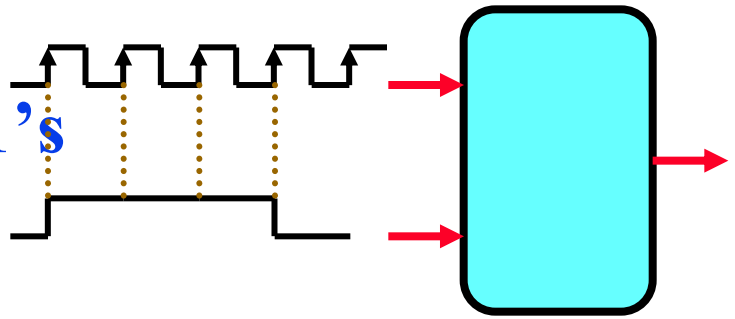
$K_B(A, B, x) = \sum(2, 3, 6)$

$d_{KB}(A, B, x) = \sum(0,1,4,5)$

Design of Clocked Sequential Circuits with *JK* F.F.

★ ***Example*:**

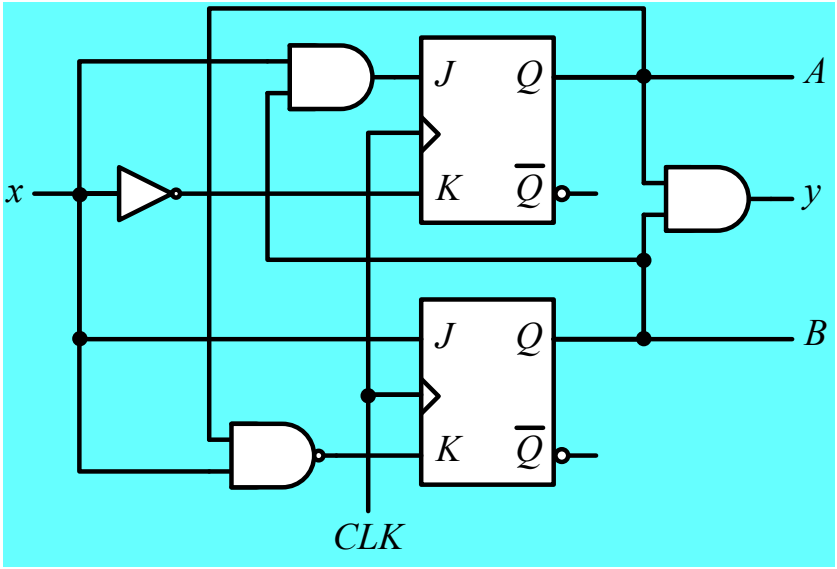
Detect 3 or more consecutive 1's

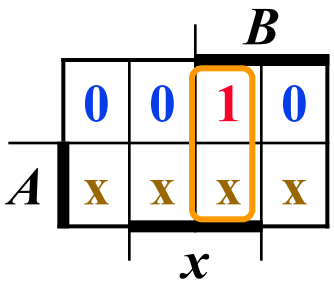


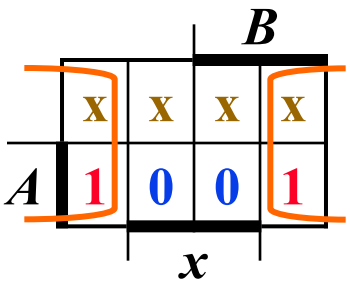
Synthesis using *JK* Flip-Flops

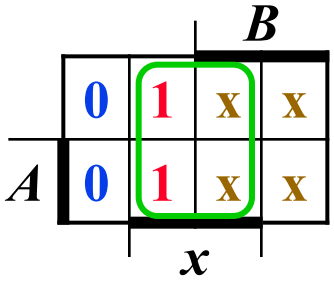
$J_A = B\,x$ $\quad K_A = x'$

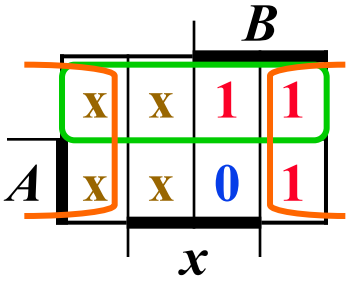
$J_B = x$ $\quad K_B = A' + x'$







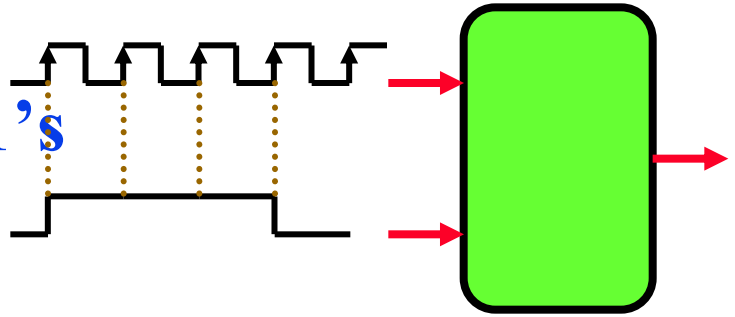




Design of Clocked Sequential Circuits with *T* F.F.

★ ***Example*:**

Detect 3 or more consecutive 1's



Present State		Input	Next State		F.F. Input	
A	*B*	*x*	*A*	*B*	T_A	T_B
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	0	0	1
0	1	1	1	0	1	1
1	0	0	0	0	1	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	1	1	1	0	0

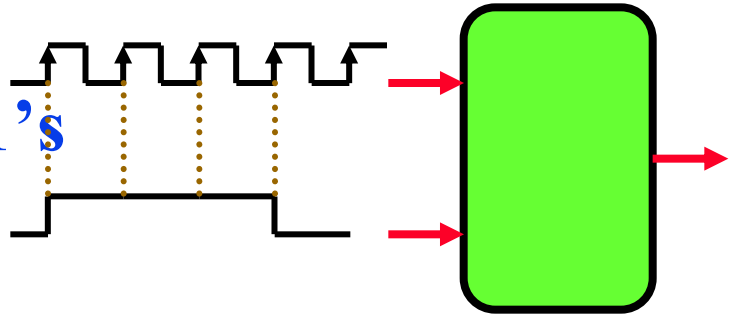
Synthesis using *T* Flip-Flops

$$T_A(A, B, x) = \sum(3, 4, 6)$$
$$T_B(A, B, x) = \sum(1, 2, 3, 5, 6)$$

Design of Clocked Sequential Circuits with *T* F.F.

★ *Example*:

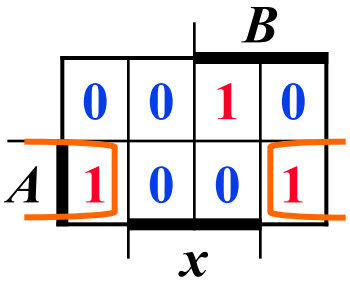
Detect 3 or more consecutive 1's

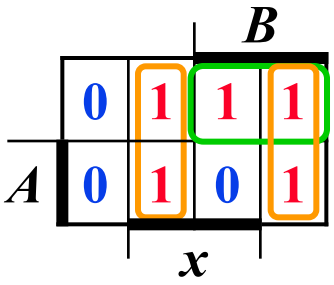


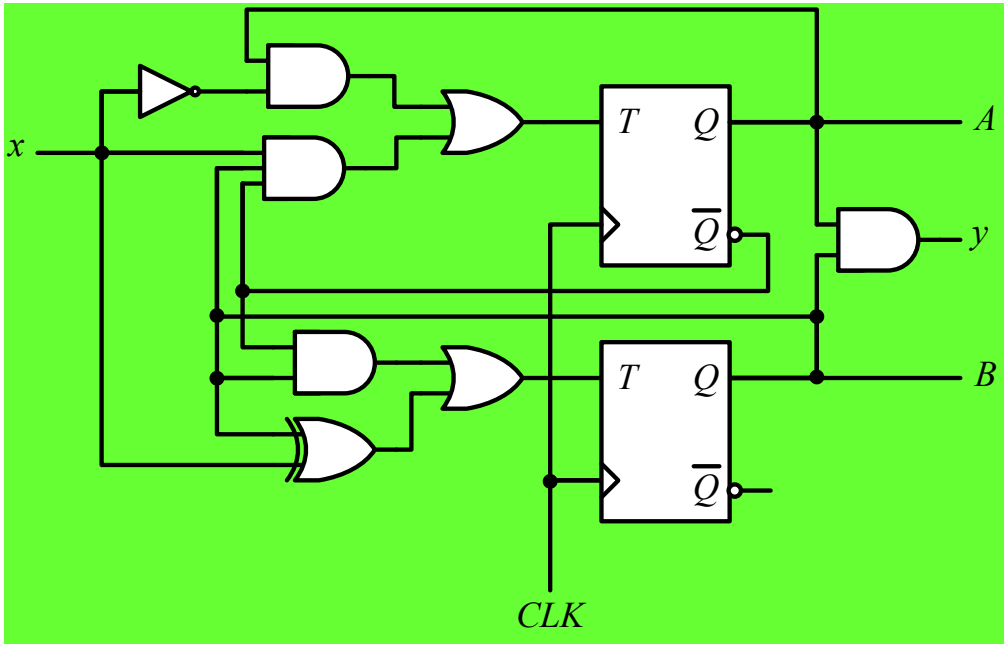
Synthesis using *T* Flip-Flops

$T_A = A\,x' + A'\,B\,x$

$T_B = A'\,B + B \oplus x$

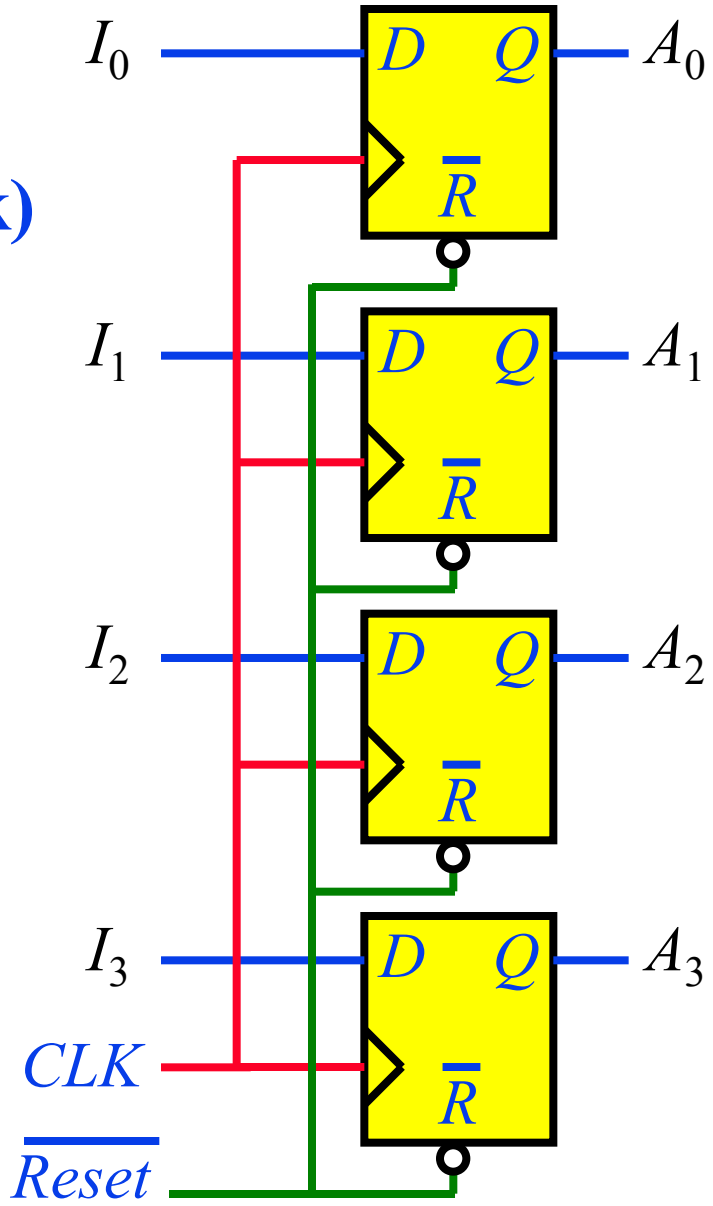




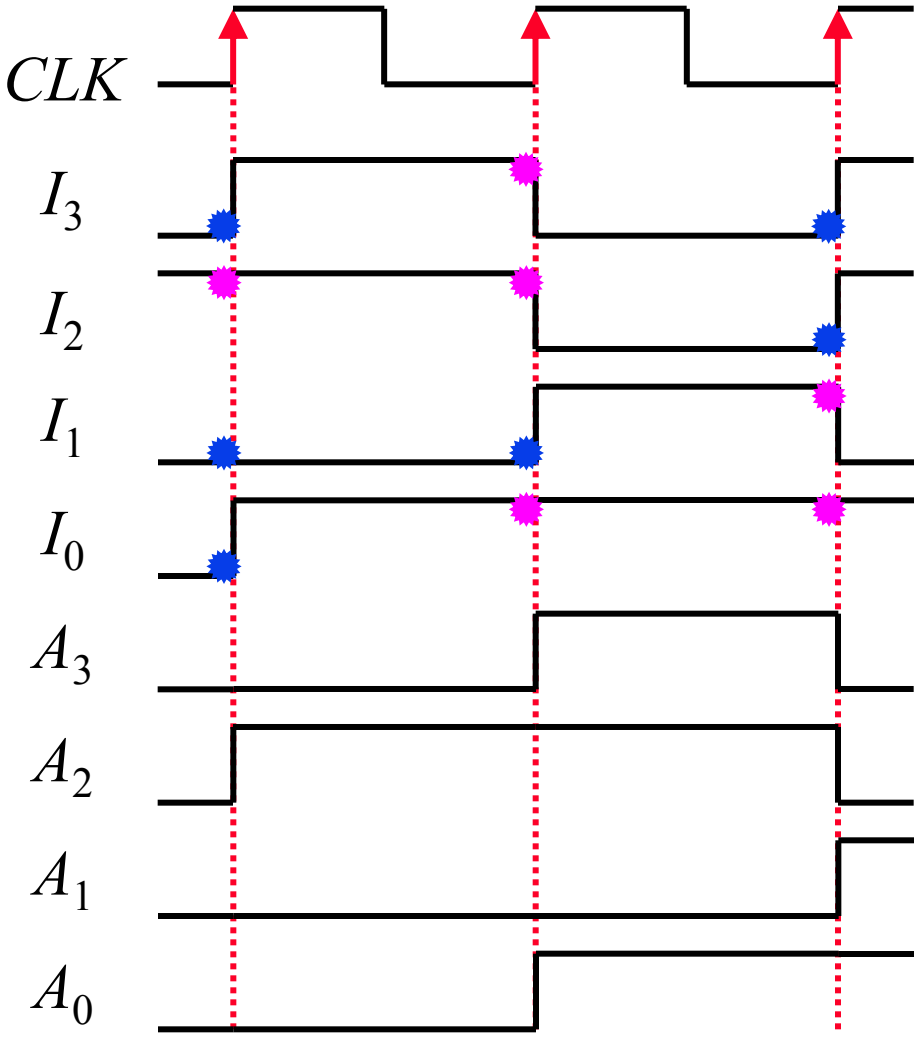


Registers

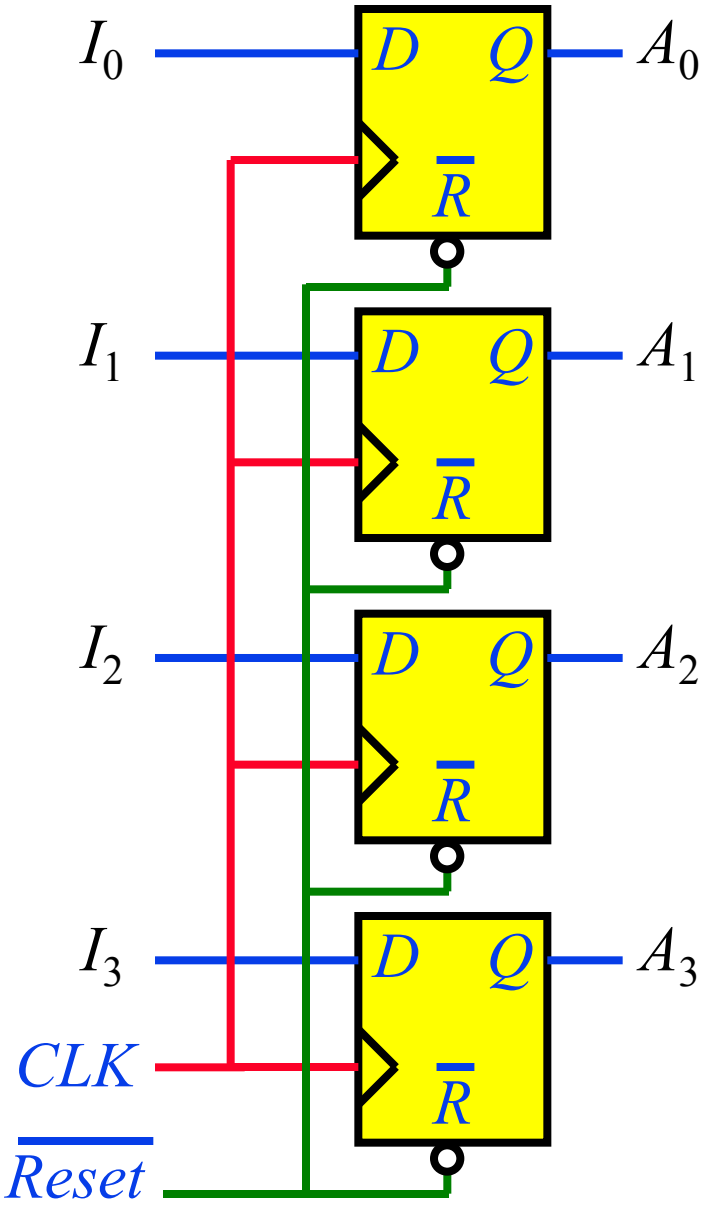
- ★ Group of *D* Flip-Flops
- ★ Synchronized (Single Clock)
- ★ Store Data



Registers

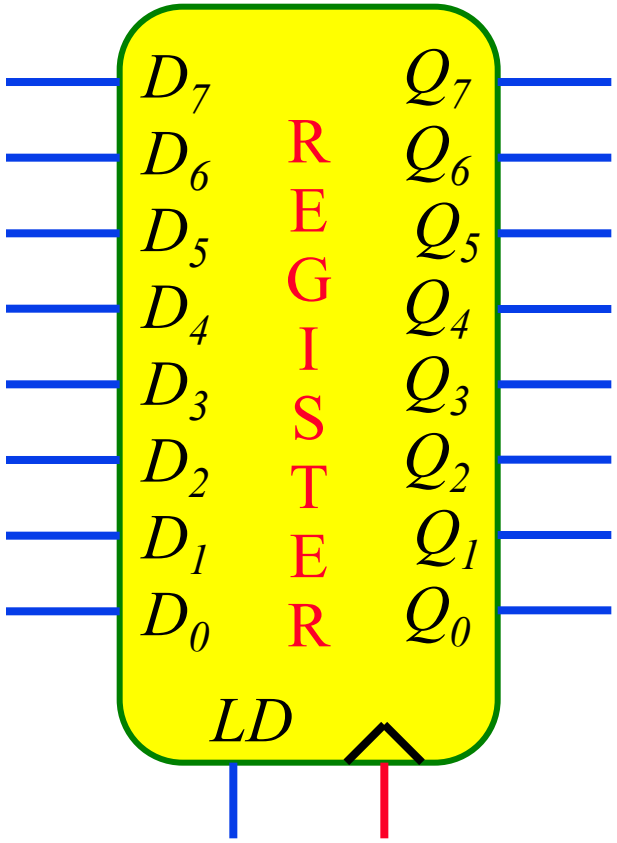


Note: New data has to go in with every clock



Registers with Parallel Load

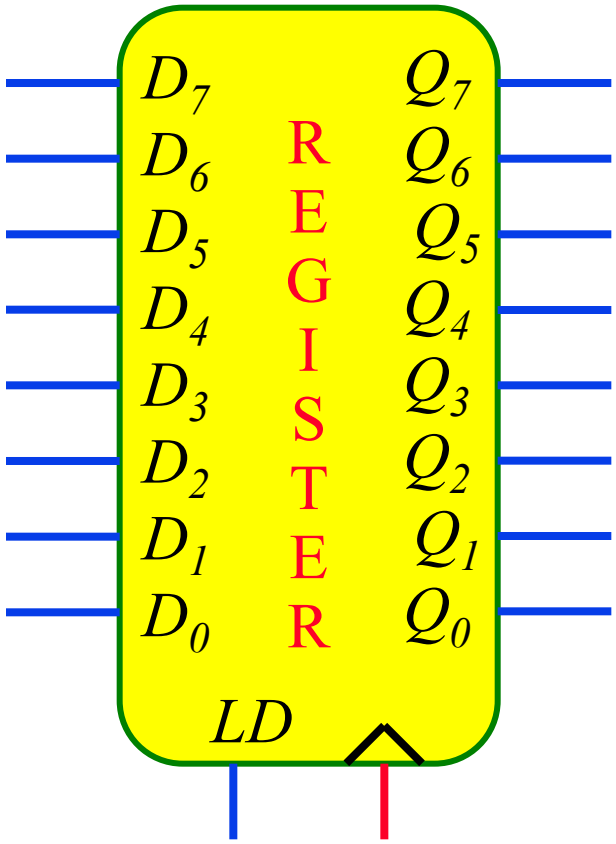
★ Control *Loading* the Register with New Data

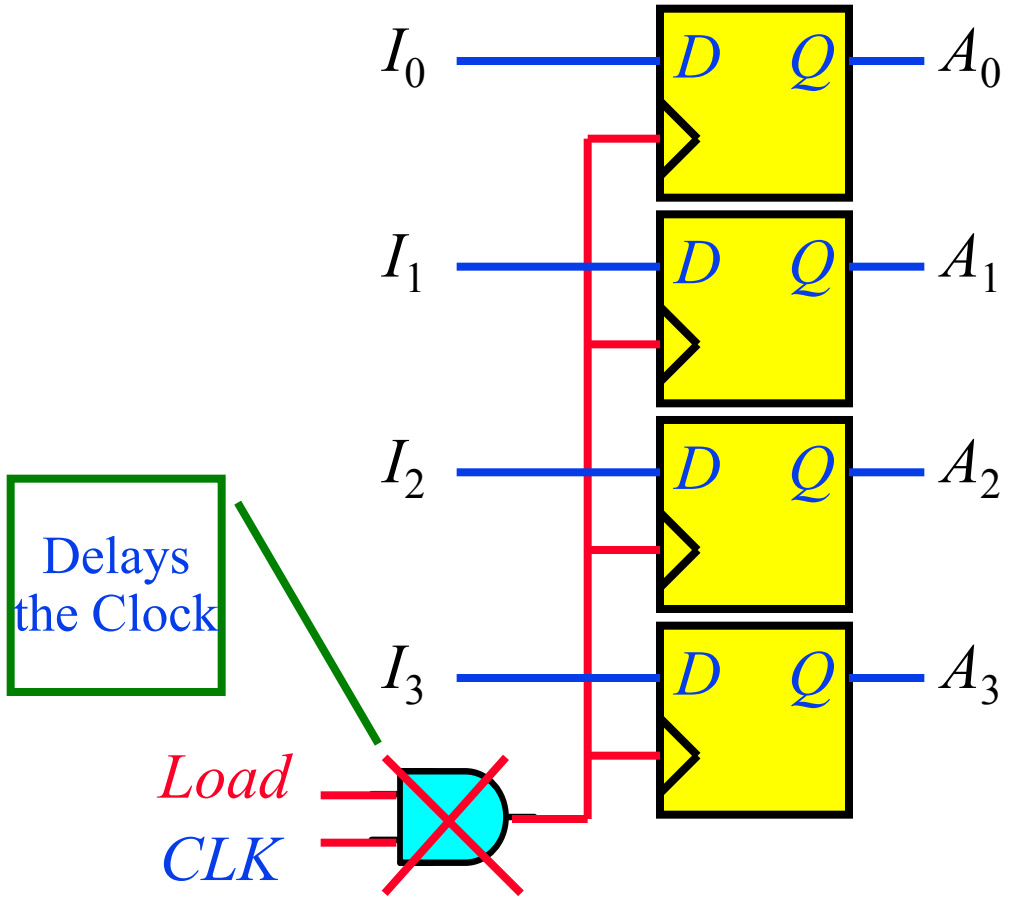


LD	*Q*(*t*+1)
0	*Q*(*t*)
1	*D*

Registers with Parallel Load

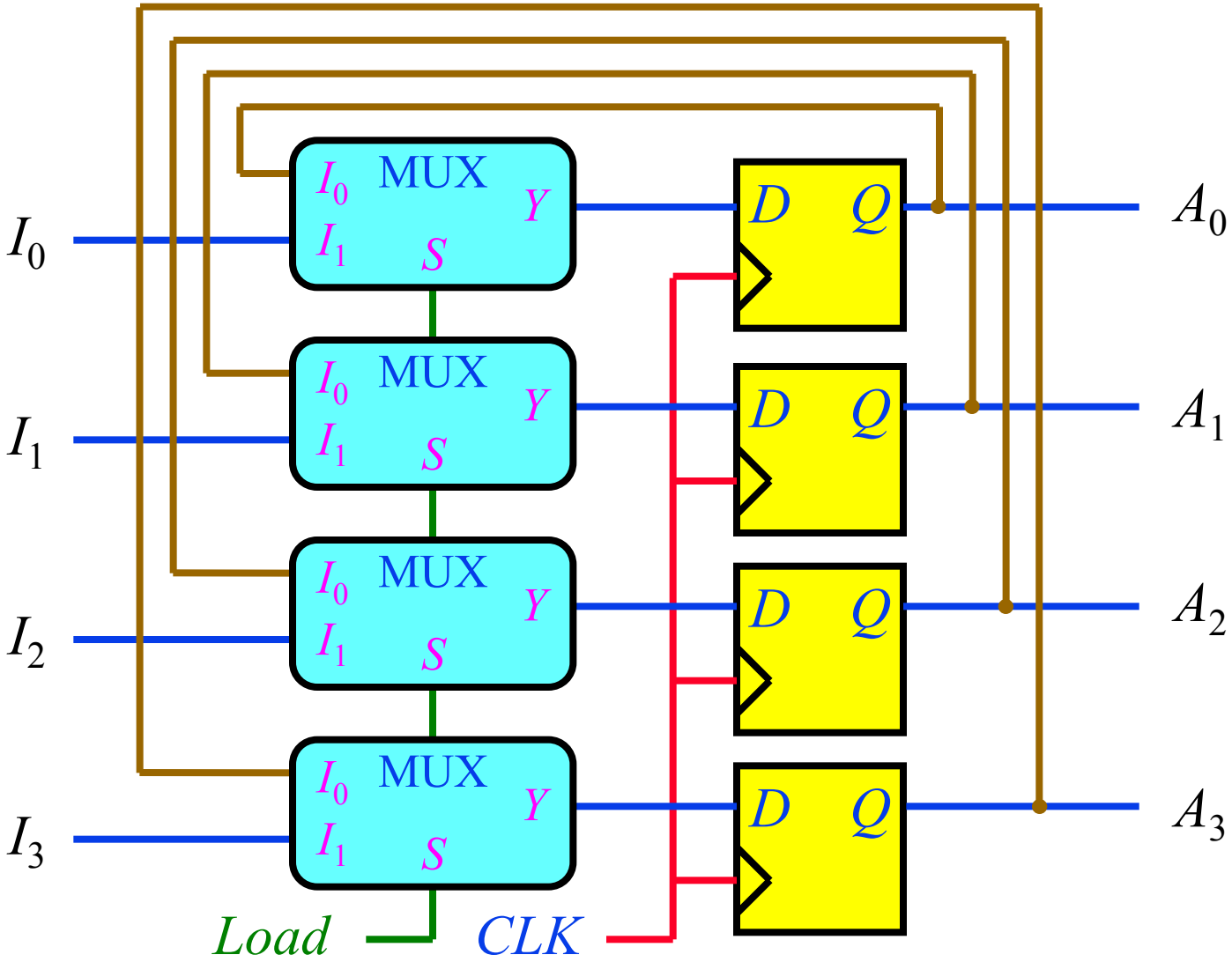
★ **Should we block the "Clock" to keep the "Data"?**





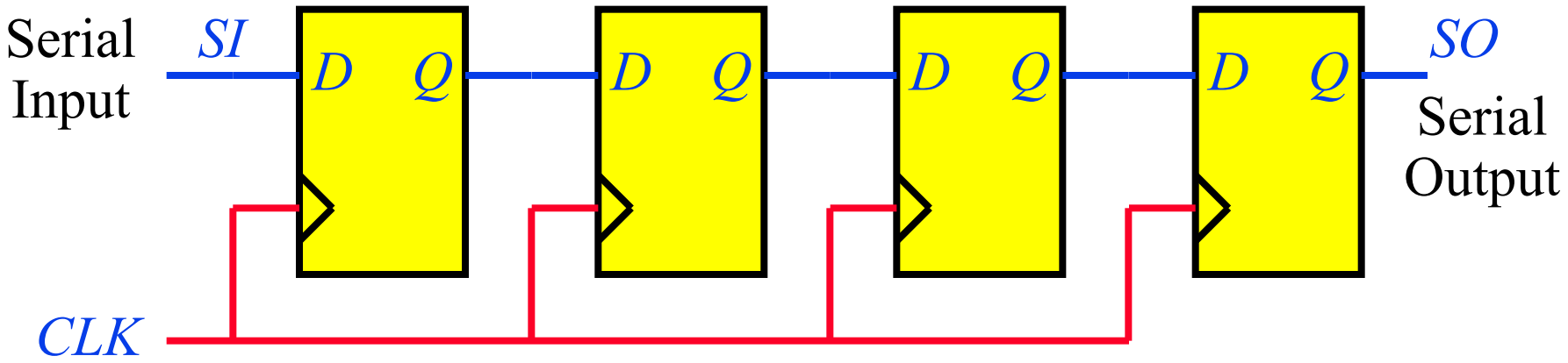
Registers with Parallel Load

★ Circulate the "old data"

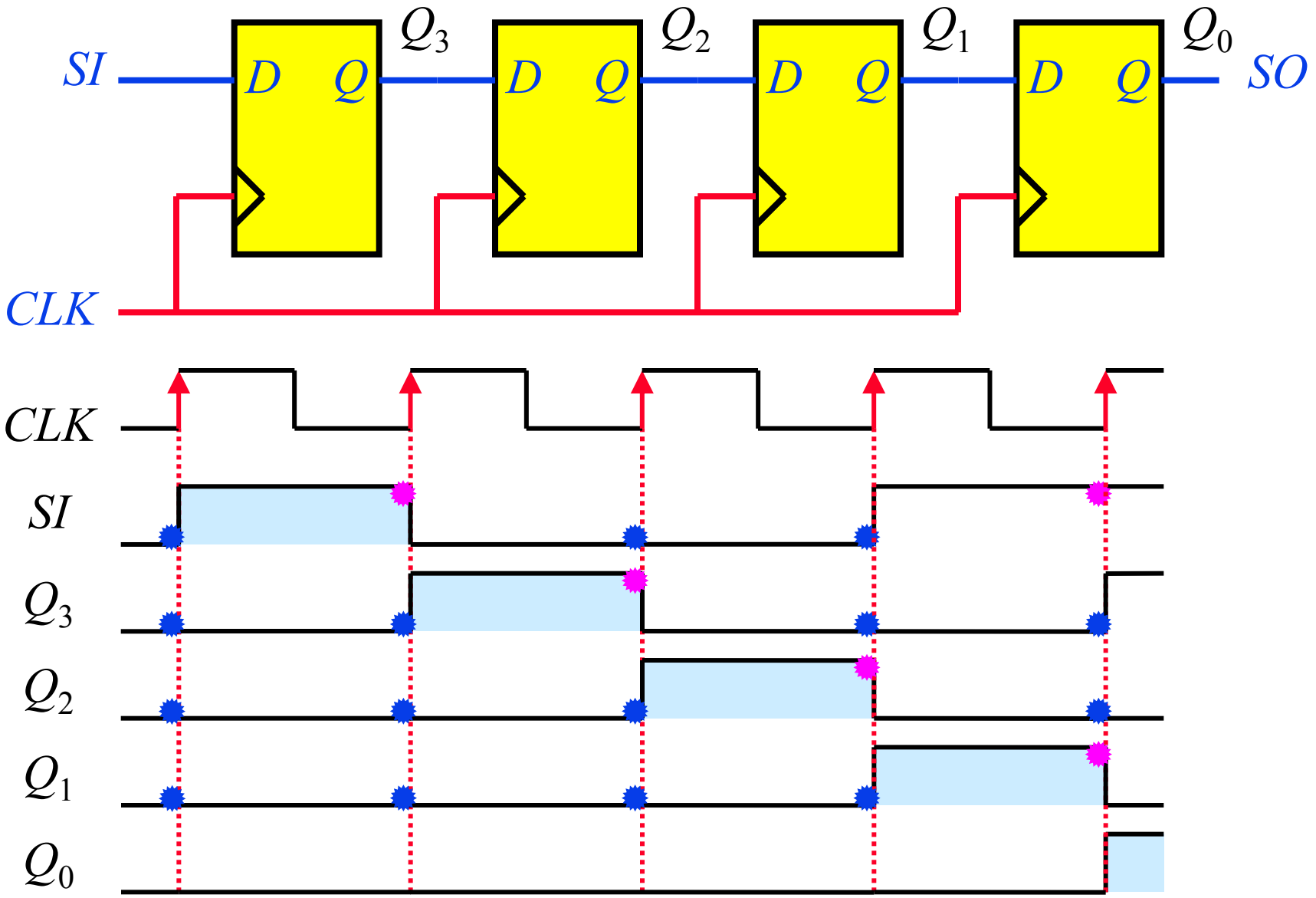


Shift Registers

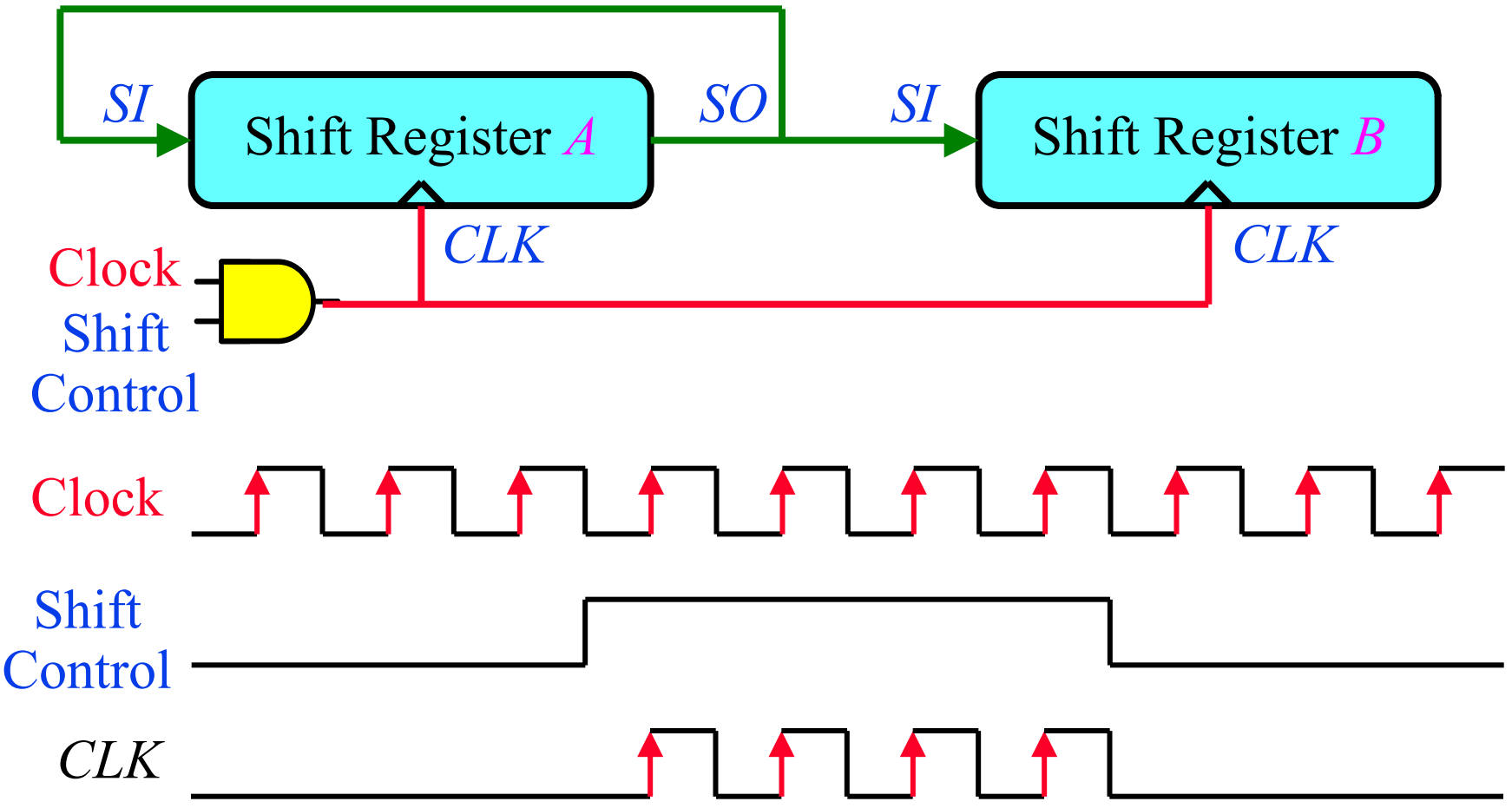
★ **4-Bit Shift Register**



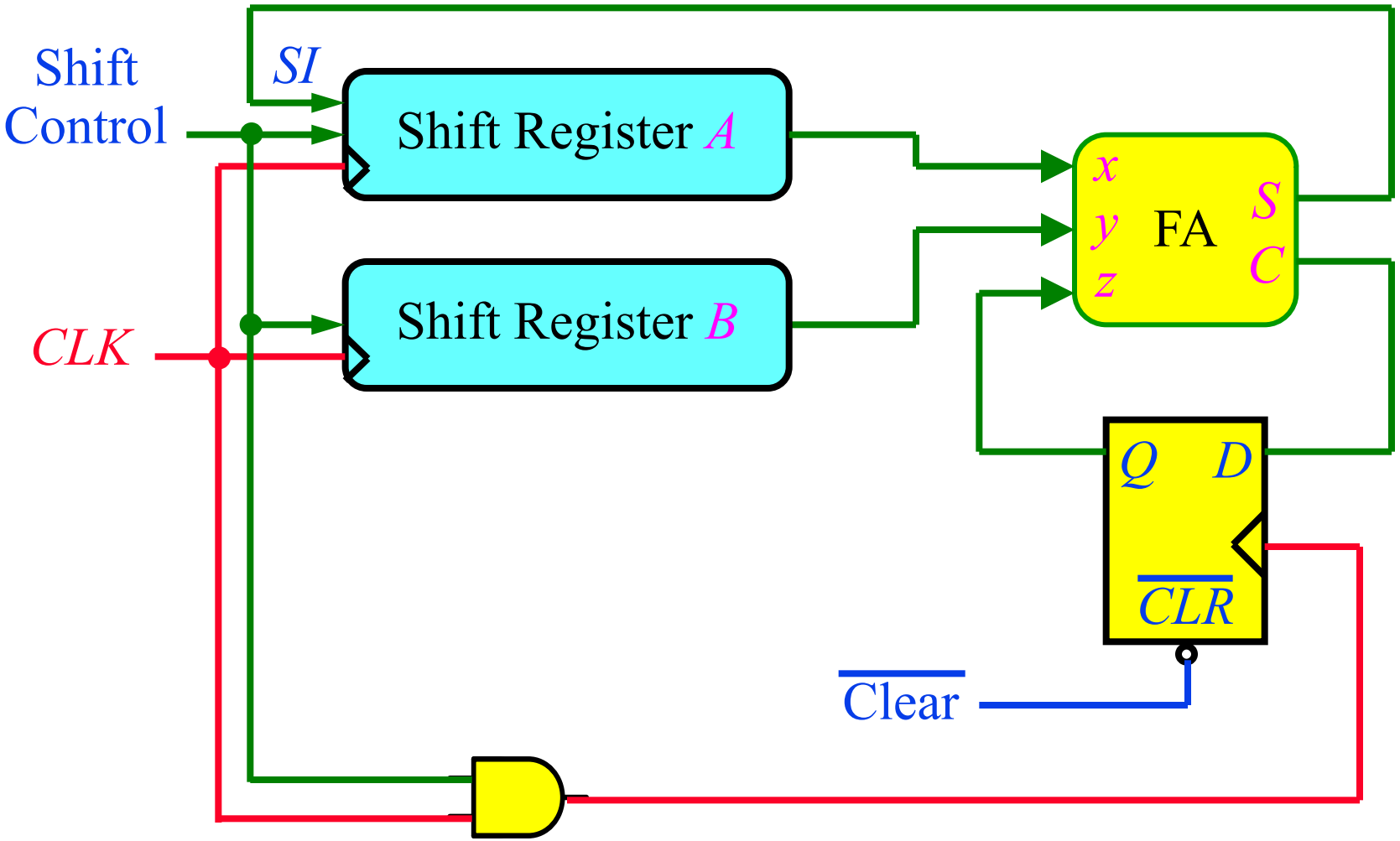
Shift Registers



Serial Transfer

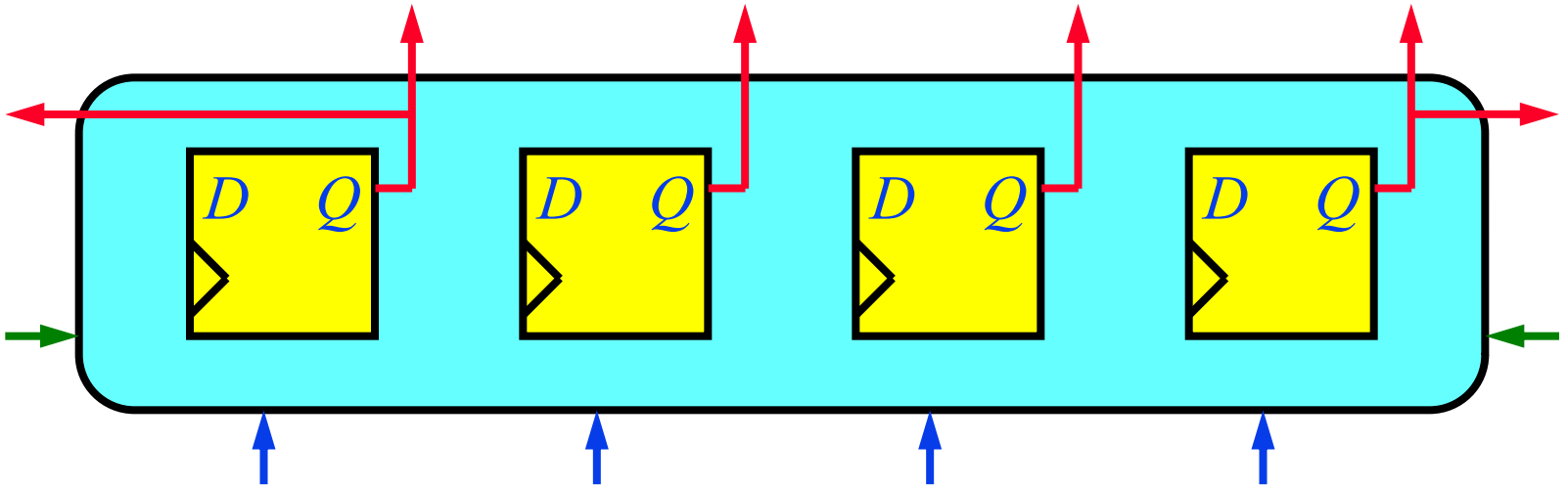


Serial Addition

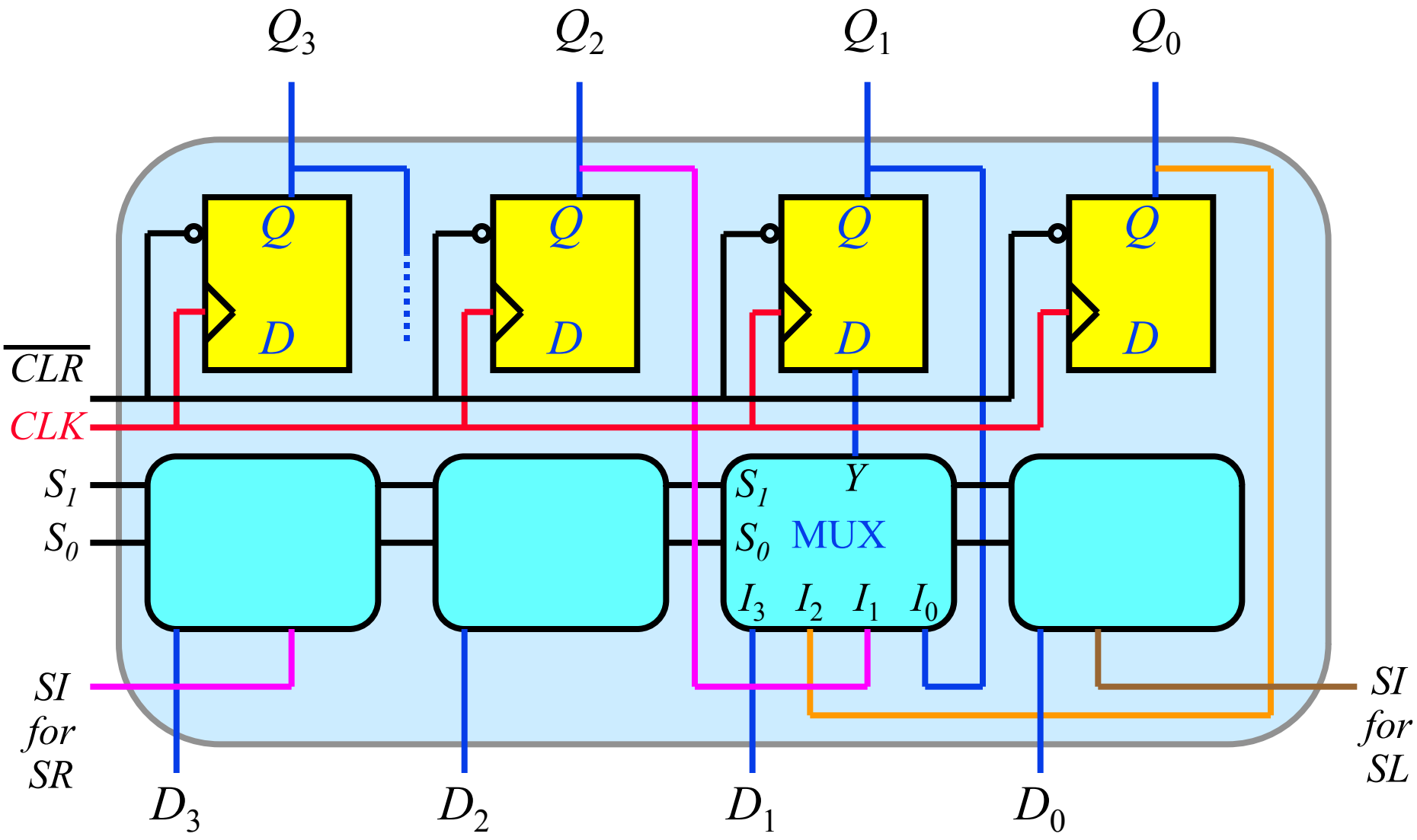


Universal Shift Register

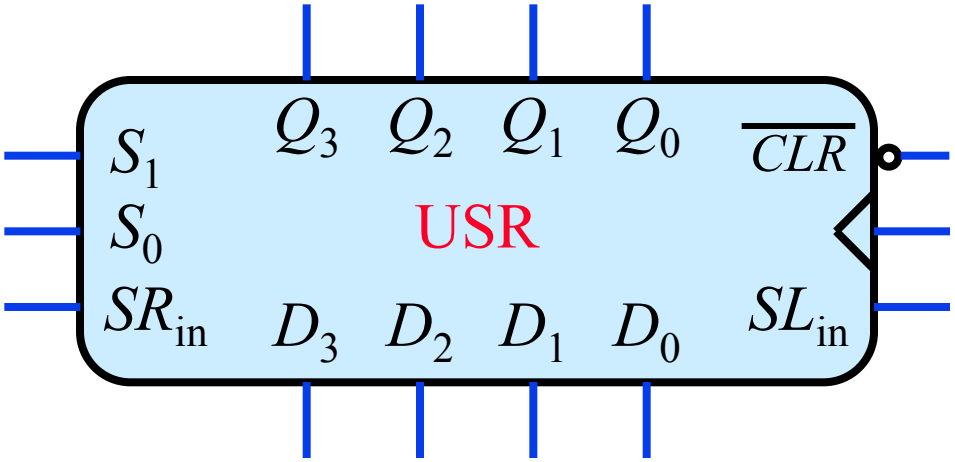
- ★ Parallel-in Parallel-out
- ★ Serial-in Serial-out
- ★ Serial-in Parallel-out
- ★ Parallel-in Serial-out



Universal Shift Register



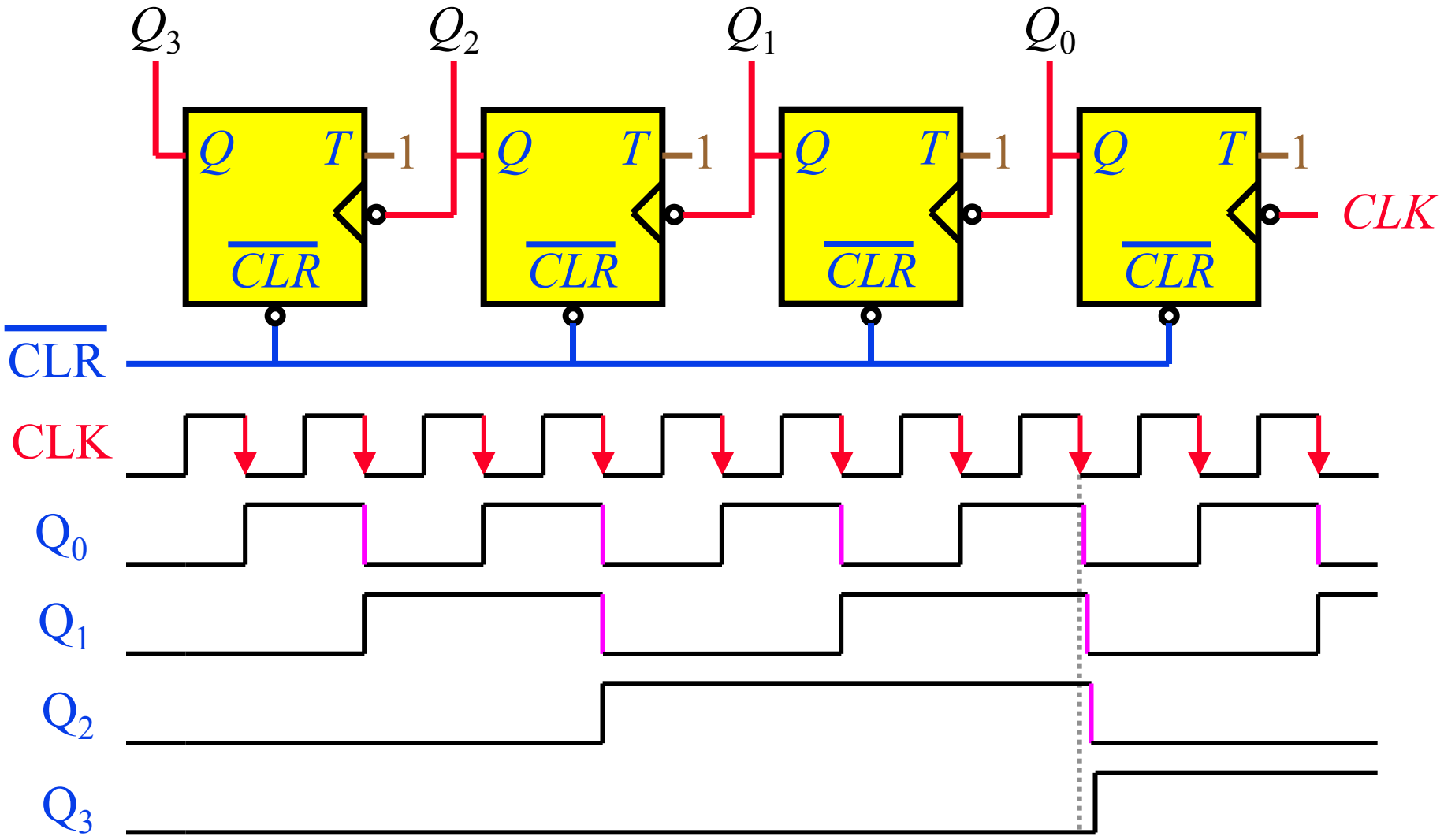
Universal Shift Register



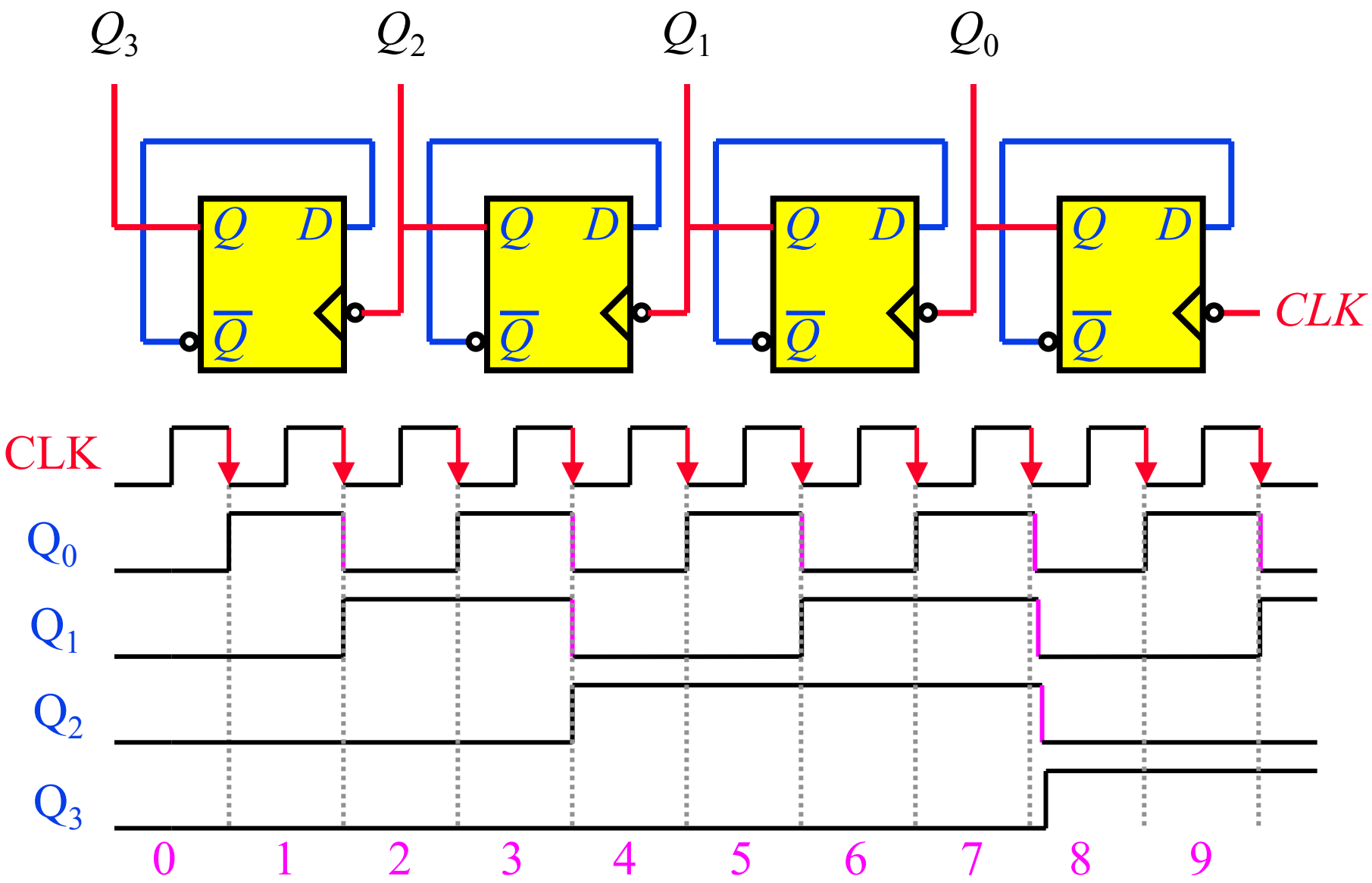
Mode Control		Register Operation
S_1	S_0	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

Ripple Counters

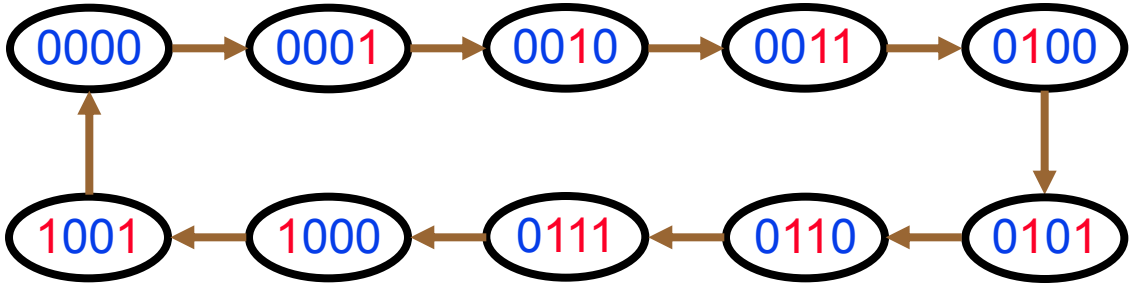
★ **Ripple ↔ Asynchronous**

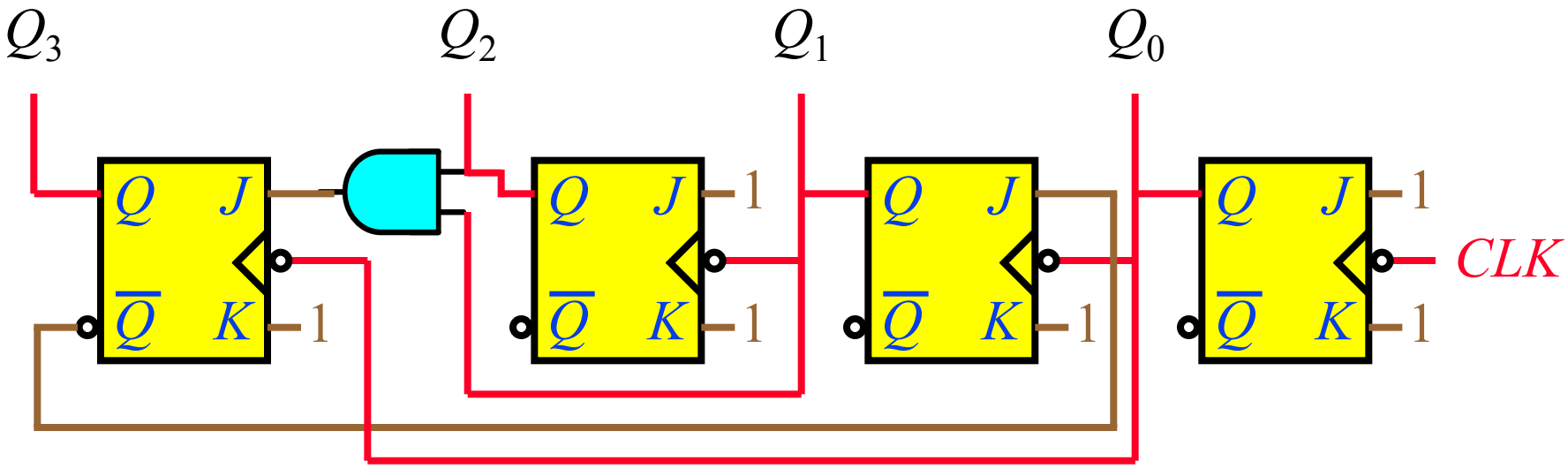


Ripple Counters

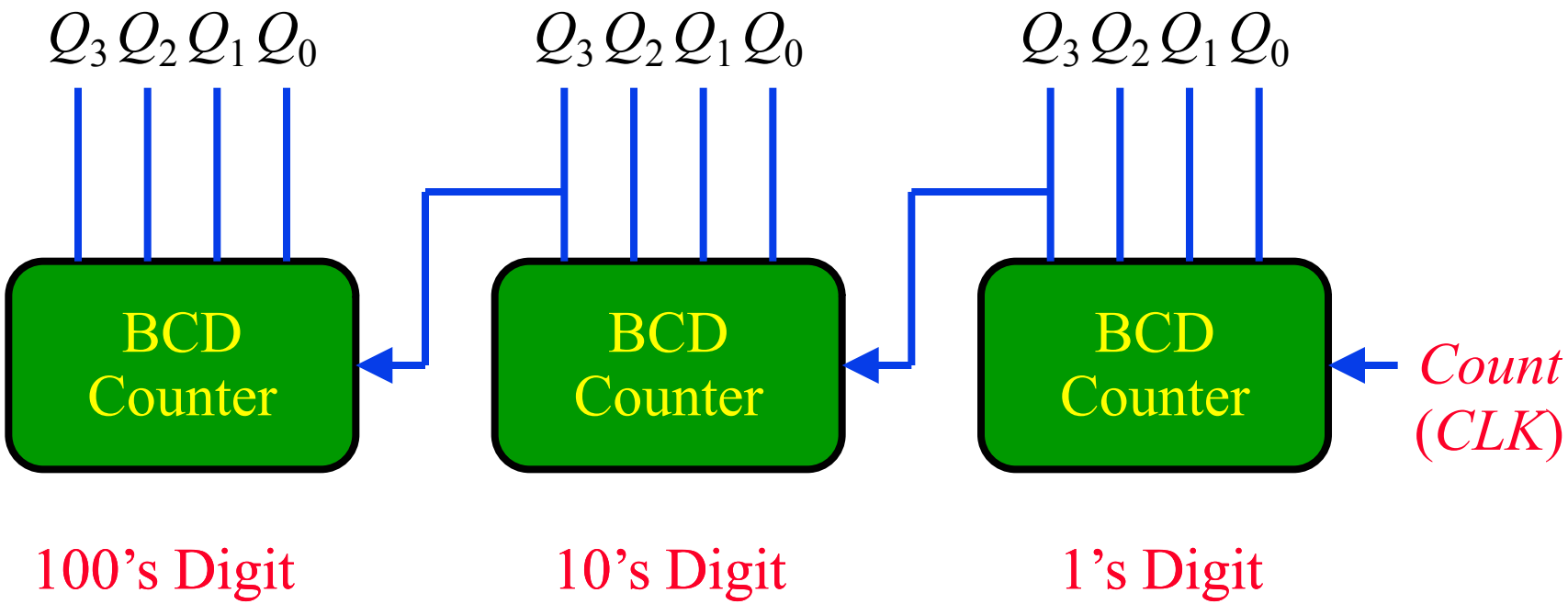


BCD Ripple Counter

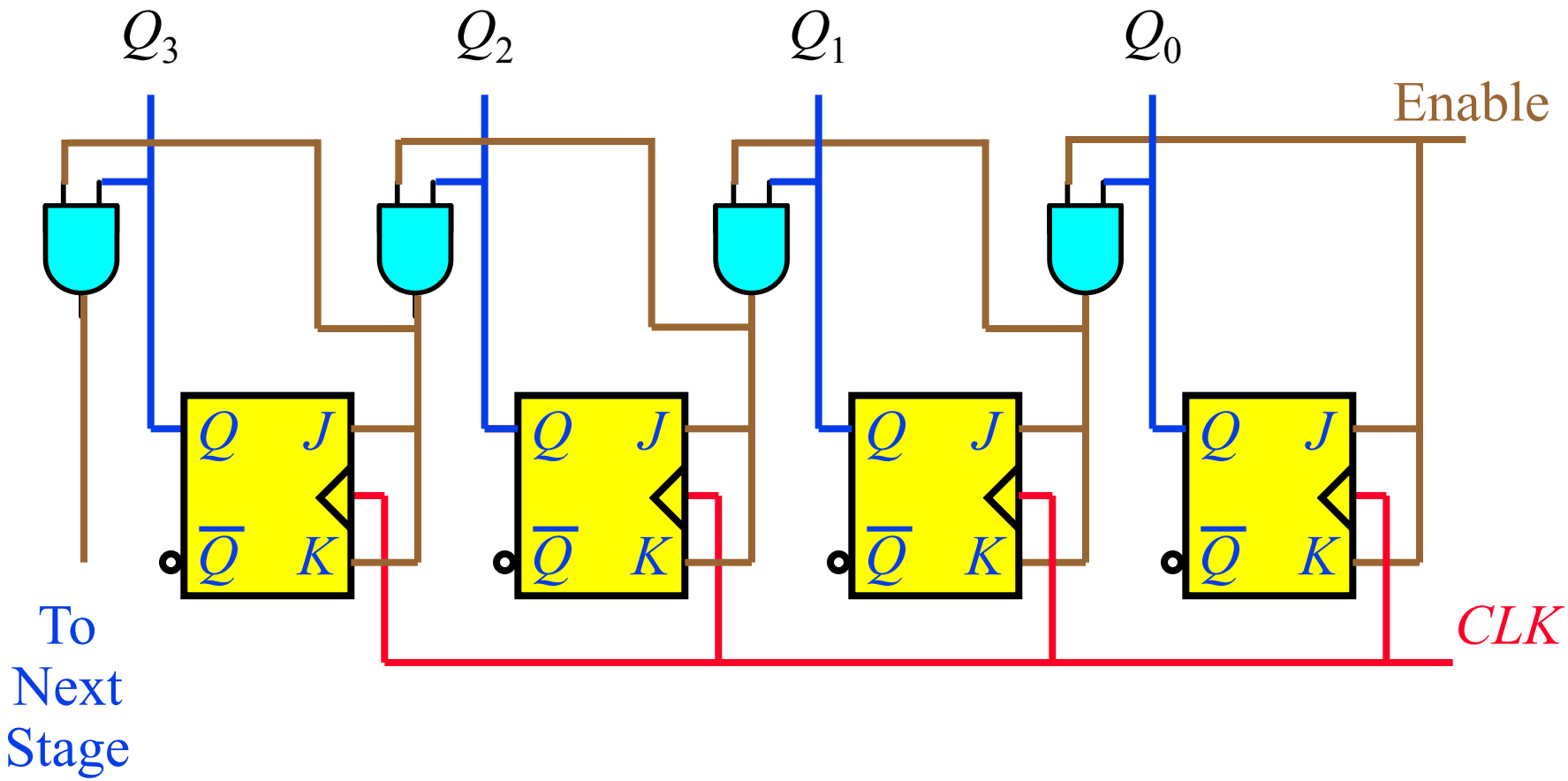




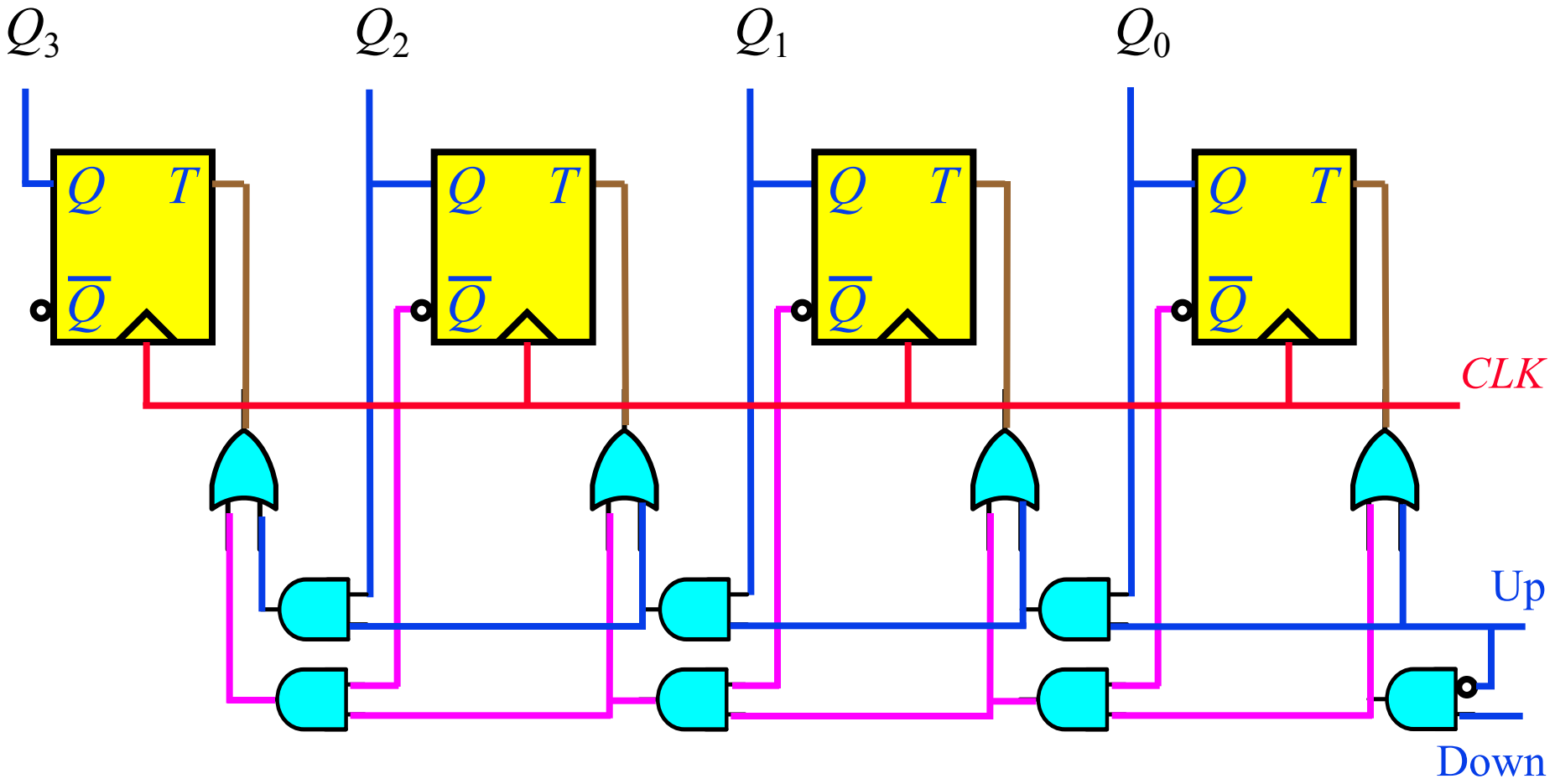
Decades Counter



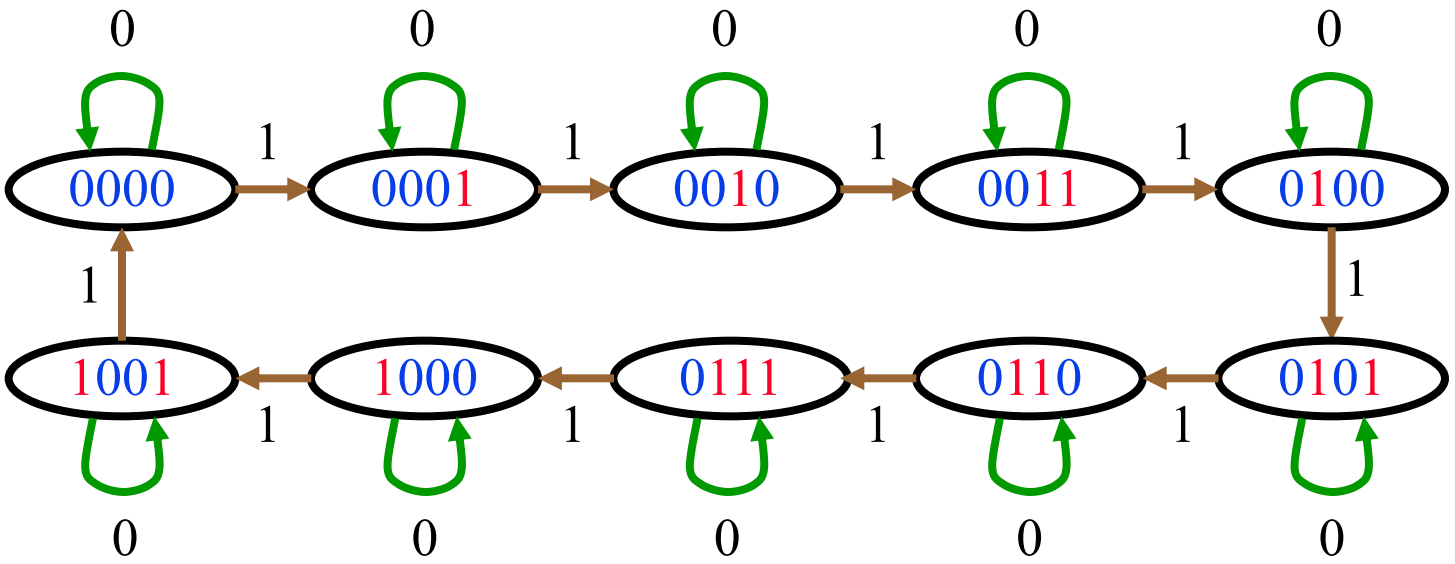
Synchronous Binary Counter

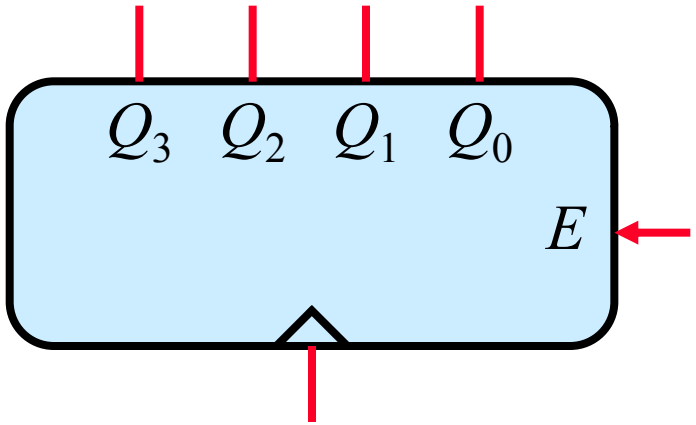


Up-Down Binary Counter

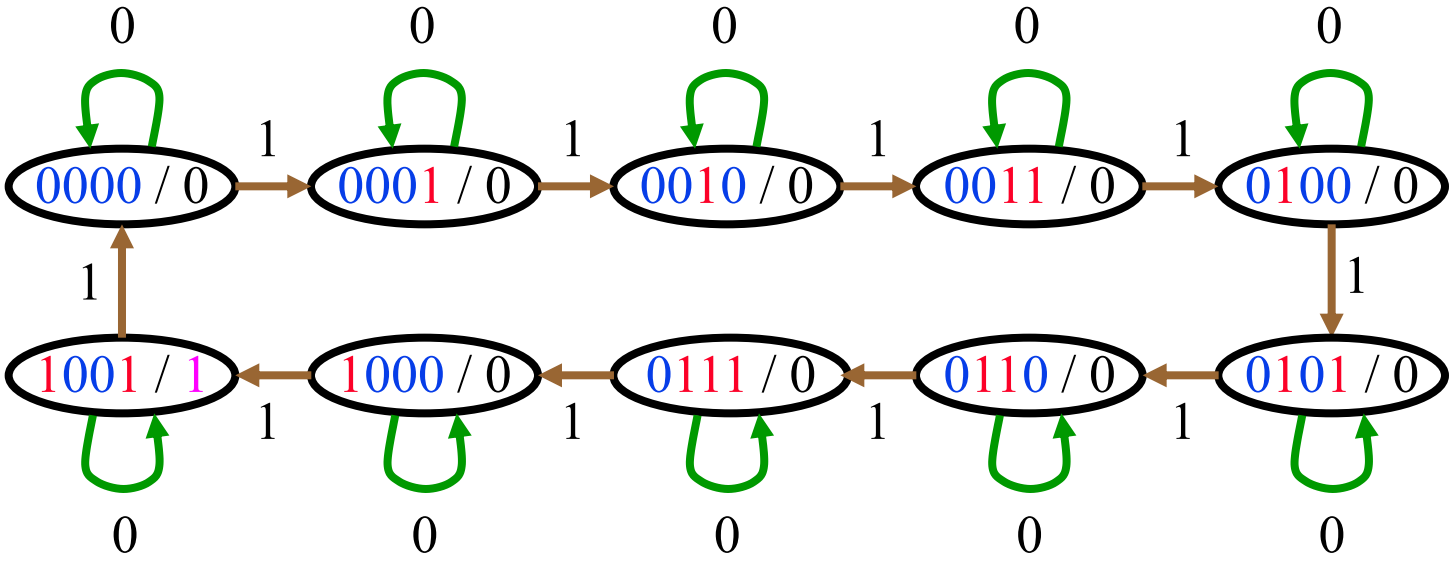


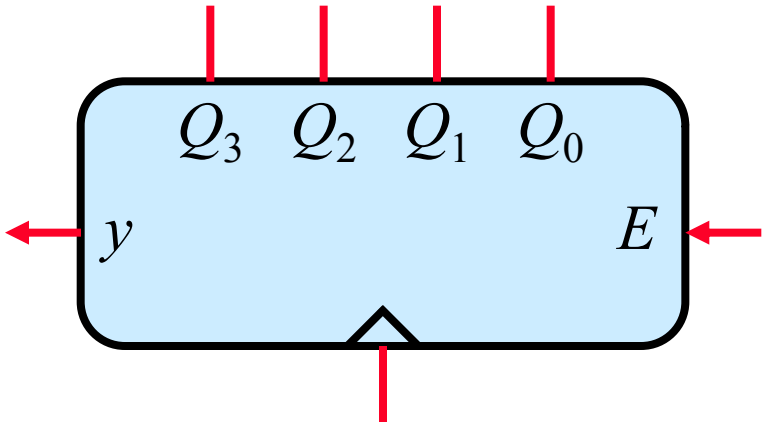
BCD Counter


0
0
0
0
0
1
1
1
1
0000
0001
0010
0011
0100
1
1
1001
1000
0111
0110
0101
1
1
1
1
0
0
0
0
0


Q_3 Q_2 Q_1 Q_0
E

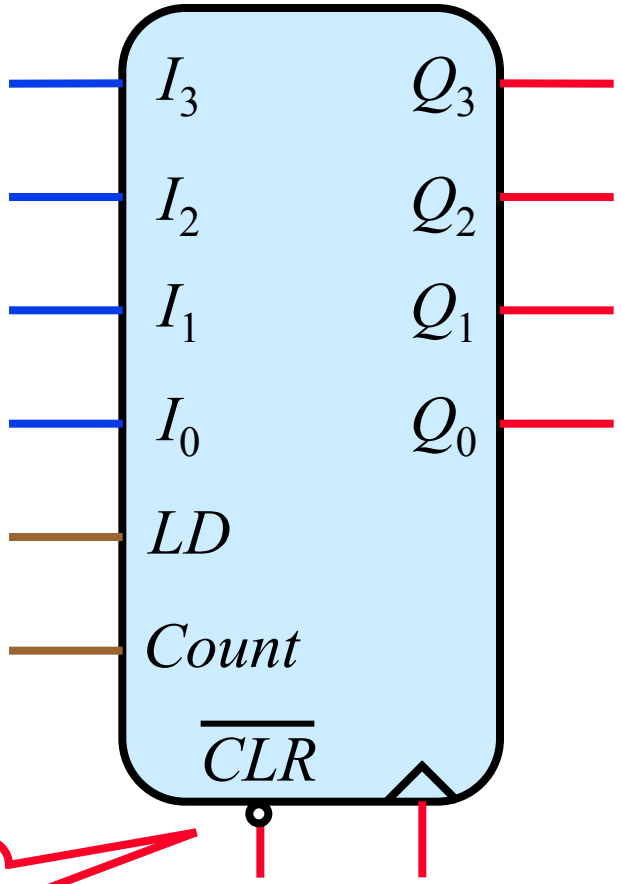
BCD Counter


0
0
0
0
0
1
1
1
1
0000 / 0
0001 / 0
0010 / 0
0011 / 0
0100 / 0
1
1
1001 / 1
1000 / 0
0111 / 0
0110 / 0
0101 / 0
1
1
1
1
0
0
0
0
0


Q_3 Q_2 Q_1 Q_0
y
E

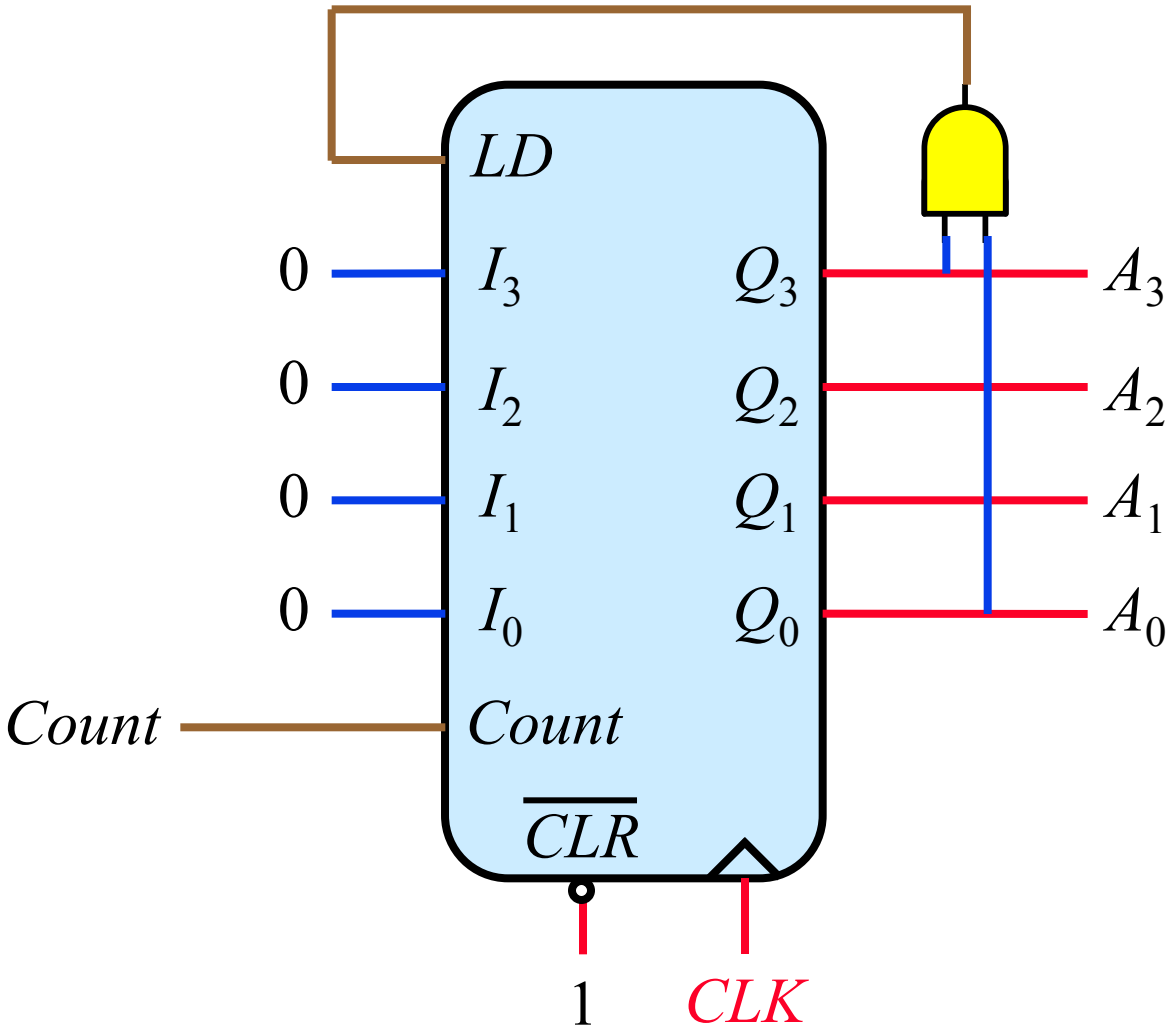
Binary Counter with Parallel Load

$\overline{CLR}$	LD	Count	Q(t+1)
0	x	x	0
1	0	0	Q(t)
1	0	1	Q(t)+1
1	1	x	I

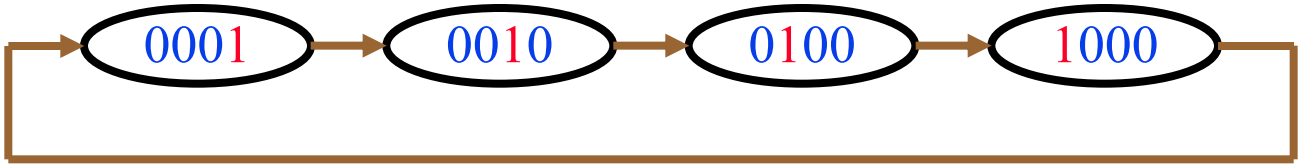


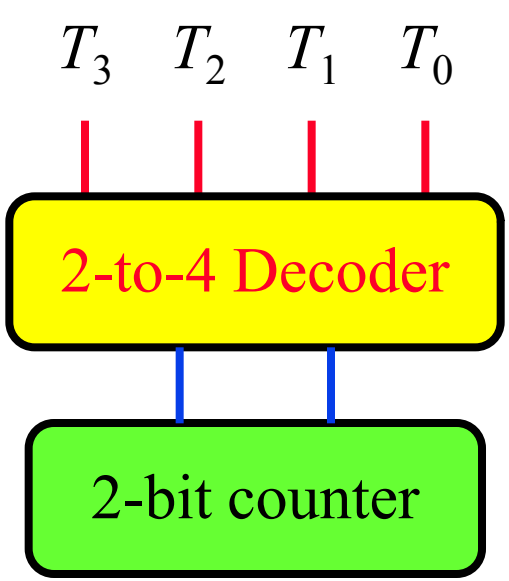
Usually Asynchronous Clear

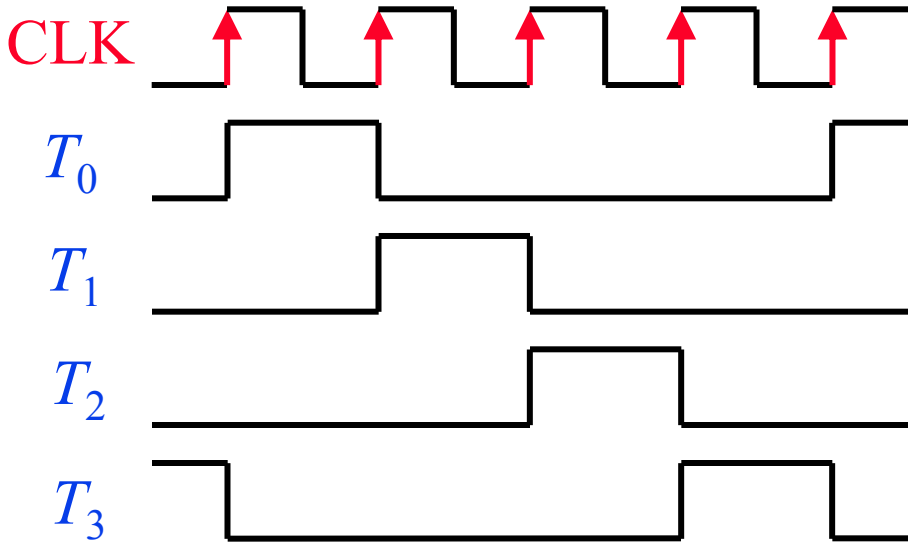
BCD Counter Example



Ring Counter







Johnson Counter