Sequential Circuits

Digital System Design

Course Instructor: Ganaraj P S

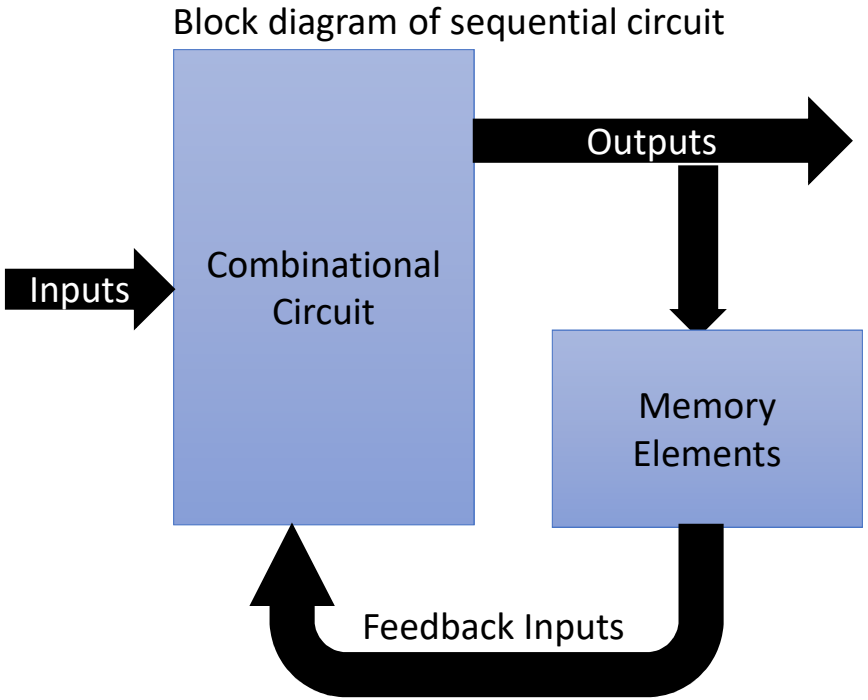
Assistant Professor (I), School of Electronics Engineering

Kalinga Institute of Industrial Engineering, Bhubaneswar

Fundamentals of sequential logic:

- In combinational circuits, output(s) depends only on the combination of present inputs, whereas in case of sequential circuits output(s) depends not only on the present input but also on the previous output(s).

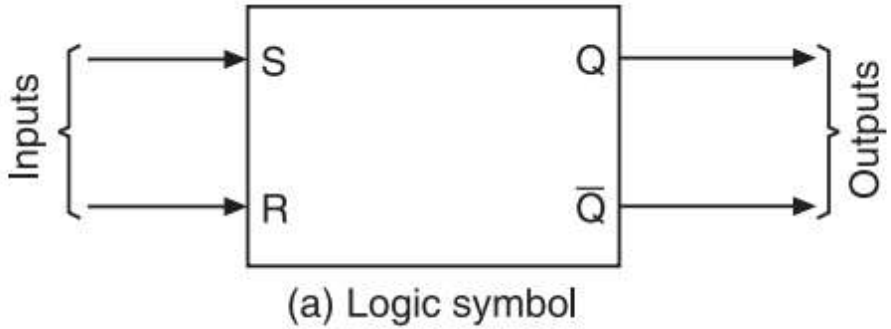
- Sequential circuits contain combinational circuits along with memory (storage) elements.
- The latches and flip-flops are the basic building block for sequential circuits, such as, memories.
- Sequential circuits includes counters, registers, and other sequential control logic



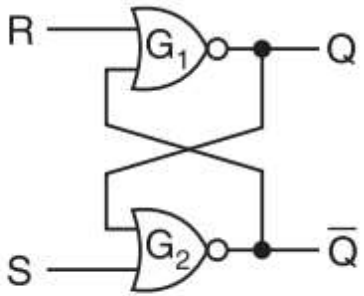
Combinational Circuits	Sequential Circuits
Outputs depend only on present inputs.	Outputs depend on both present inputs and present state.
Feedback path is not present.	Feedback path is present.
Memory elements are not required.	Memory elements are required.
Clock signal is not required.	Clock signal is required.
Easy to design.	Difficult to design.

- **Flip-flops** (also known as **Bistable Multivibrators)** which have two stable states, called SET and RESET; they can retain either of these states indefinitely, making them useful as storage devices.

S-R Latch (Active-High S-R latch):



(a) Logic symbol

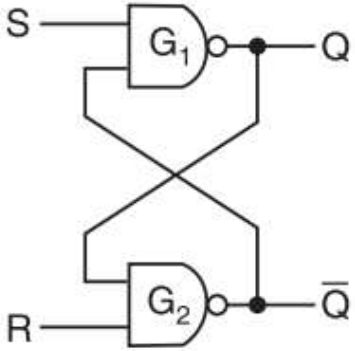


(b) Logic diagram

S	R	Q_n	Q_{n+1}	State
0	0	0	0	No Change (NC)
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	×	Indeterminate (invalid)
1	1	1	×	

(c) Truth table

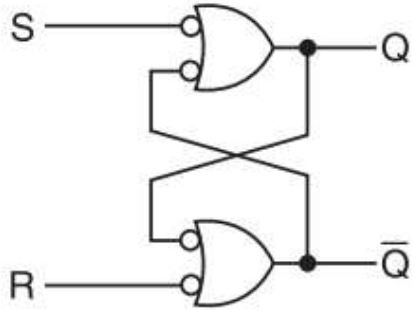
S-R Latch (Active-Low S-R latch):



(a) Using NAND gates

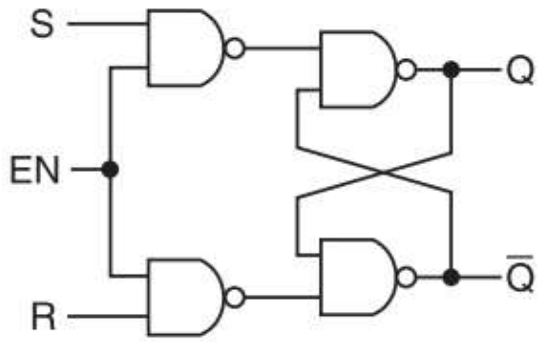
S	R	Q_n	Q_{n+1}	State
0	0	0	×	Indeterminate (invalid)
0	0	1	×	
0	1	0	1	Set
0	1	1	1	
1	0	0	0	Reset
1	0	1	0	
1	1	0	0	No Change (NC)
1	1	1	1	

(b) Truth table

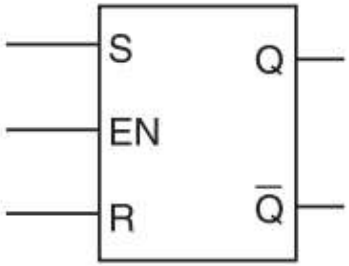


(c) Using OR gates

Gated S-R Latch (Clocked S-R Flip-Flop):



(a) Logic diagram

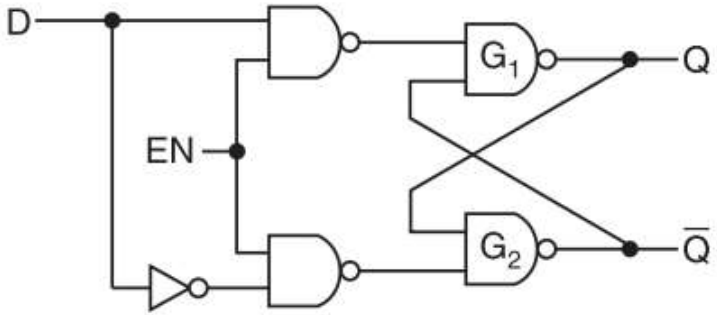


(b) Logic symbol

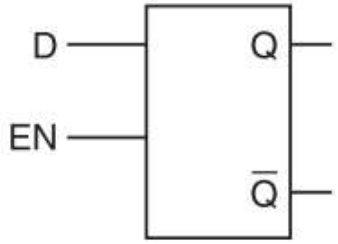
EN	S	R	Q_n	Q_{n+1}	State
1	0	0	0	0	No change (NC)
1	0	0	1	1	
1	0	1	0	0	Reset
1	0	1	1	0	
1	1	0	0	1	Set
1	1	0	1	1	
1	1	1	0	×	Indeterminate (invalid)
1	1	1	1	×	
0	×	×	0	0	No Change (NC)
0	×	×	1	1	

(c) Truth table

Gated D Latch (Clocked D Flip-Flop):



(a) Logic diagram

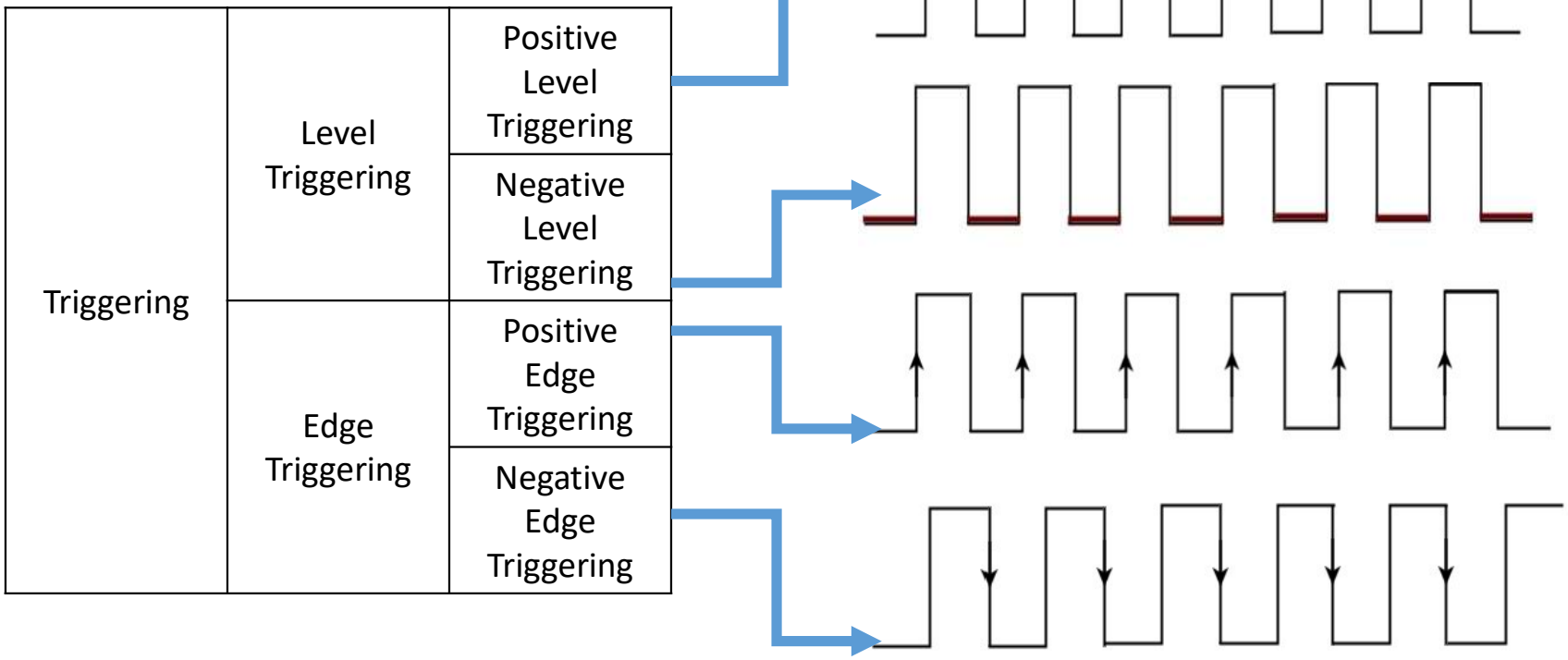


(b) Logic symbol

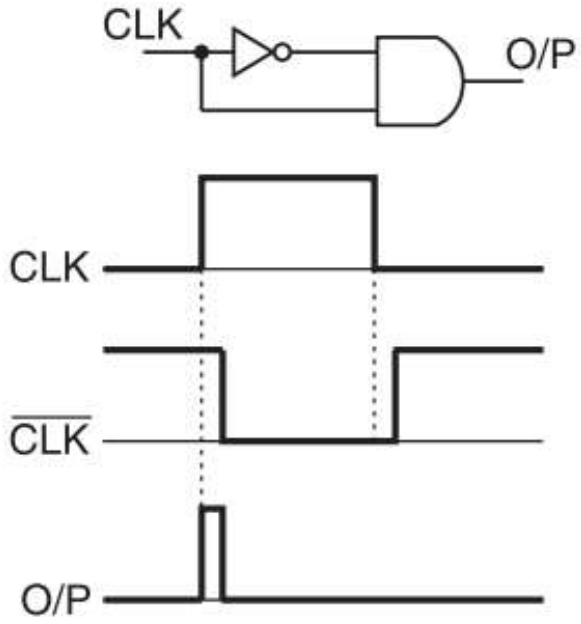
EN	D	Q_n	Q_{n+1}	State
1	0	0	0	Reset
1	0	1	0	
1	1	0	1	Set
1	1	1	1	
0	×	0	0	No Change (NC)
0	×	1	1	

(c) Truth table

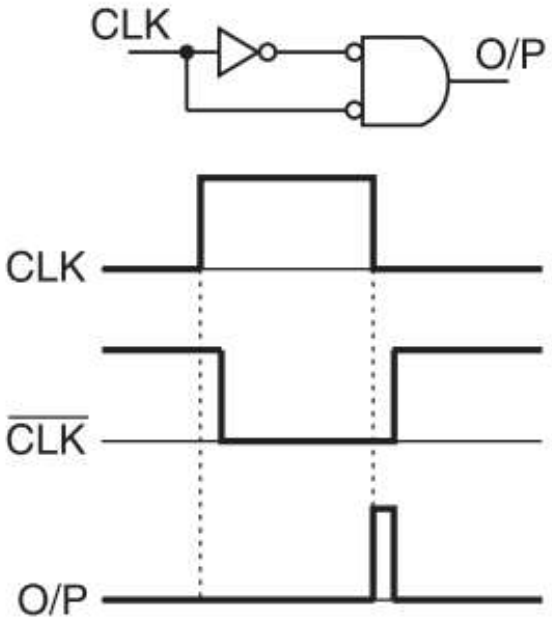
Clock Signal and Triggering



Generation of Narrow Spikes (PTD*):



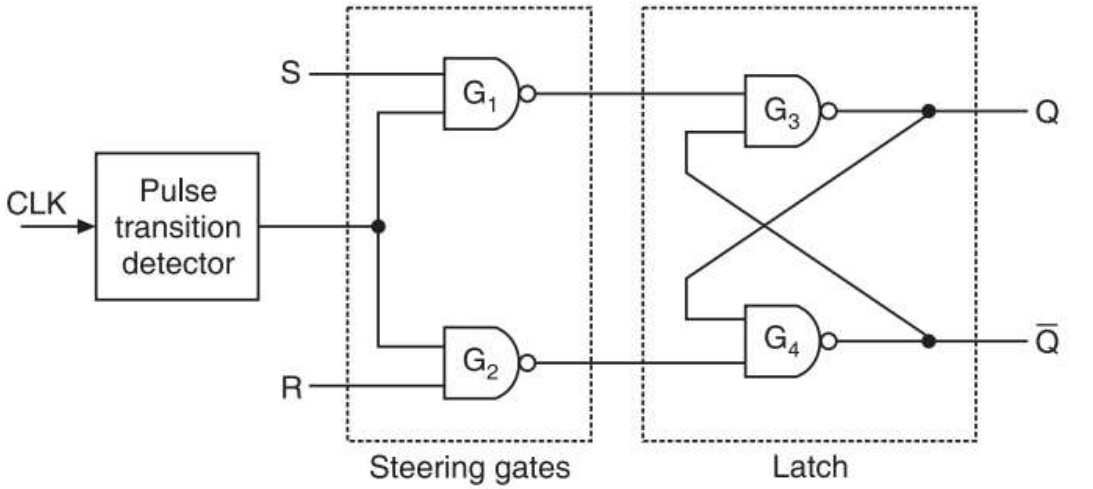
(a) Generation of a narrow spike at positive-going transition of the clock pulse

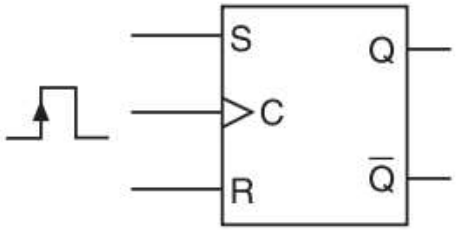


(b) Generation of a narrow spike at negative-going transition of the clock pulse

*** PTD: Pulse Transition Detector**

Edge-Triggered S-R Flip-Flop:



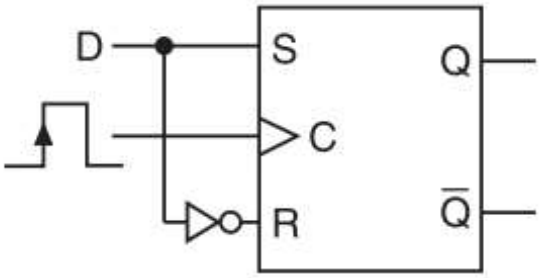


(a) Logic symbol

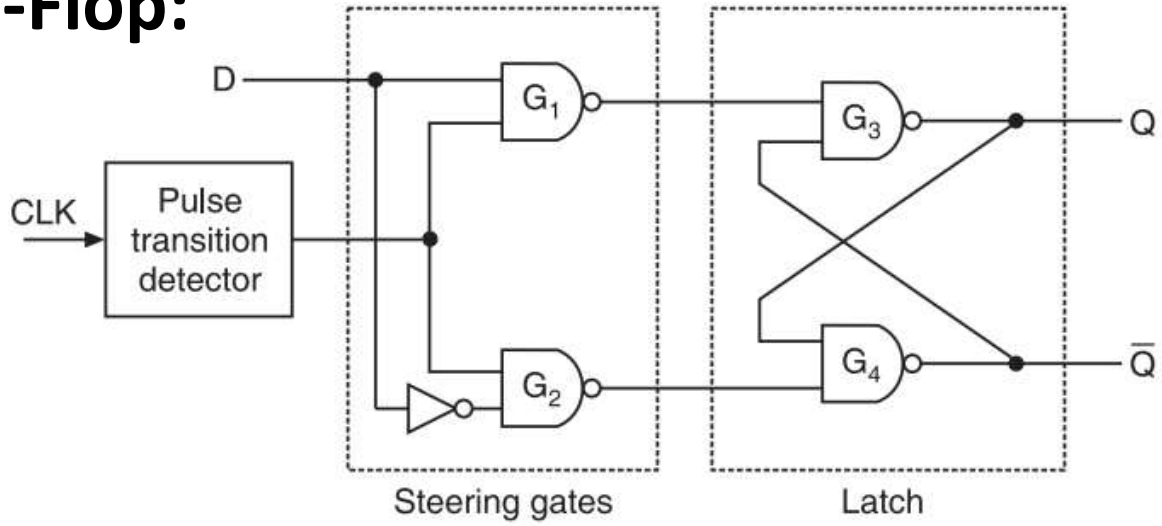
C	S	R	Q_n	Q_{n+1}	State
↑	0	0	0	0	No Change (NC)
↑	0	0	1	1	
↑	0	1	0	0	Reset
↑	0	1	1	0	
↑	1	0	0	1	Set
↑	1	0	1	1	
↑	1	1	0	×	Indeterminate
↑	1	1	1	×	
0	×	×	0	0	No Change (NC)
0	×	×	1	1	

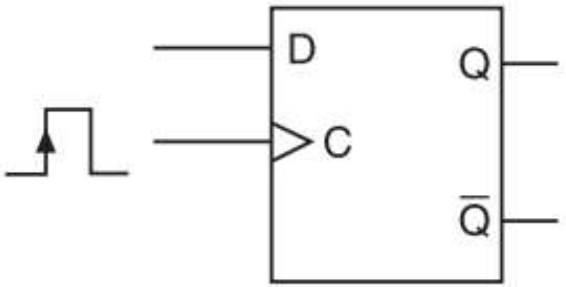
(b) Truth table

Edge-Triggered D Flip-Flop:



(a) D flip-flop from the S-R flip-flop



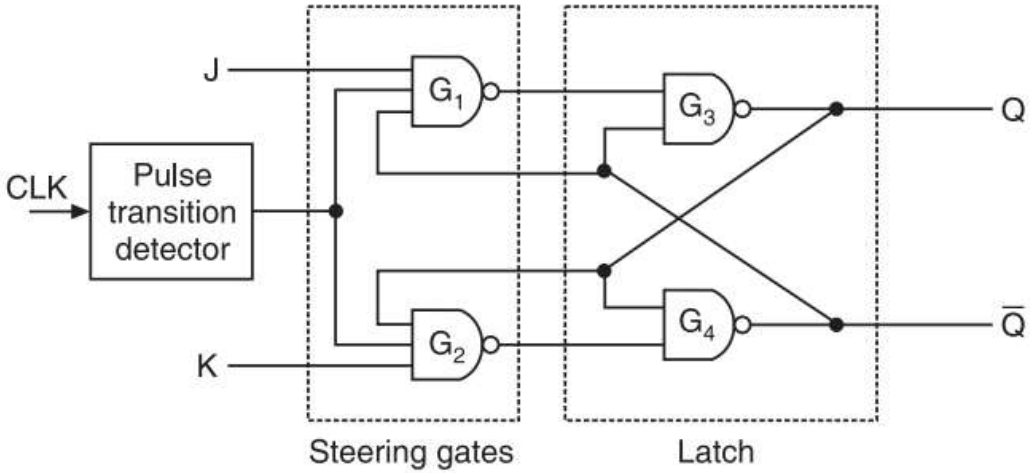


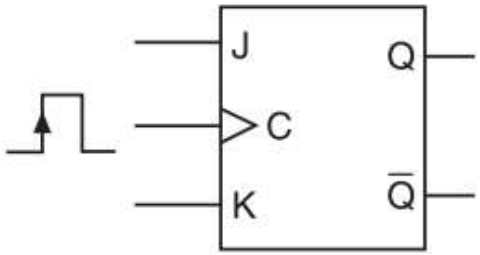
(b) Logic symbol

C	D	Q_n	Q_{n+1}	State
↑	0	0	0	Reset
↑	0	1	0	
↑	1	0	1	Set
↑	1	1	1	
0	×	0	0	No Change (NC)
0	×	1	1	

(c) Truth table

Edge-Triggered J-K Flip-Flop:



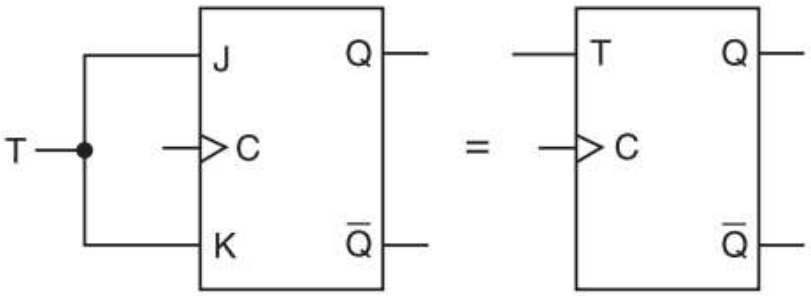


(a) Logic symbol

C	J	K	Q_n	Q_{n+1}	State
↑	0	0	0	0	No Change (NC)
↑	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	
0	×	×	0	0	No Change (NC)
0	×	×	1	1	

(b) Truth table

Edge-Triggered T Flip-Flop (Toggle Flip-Flop):

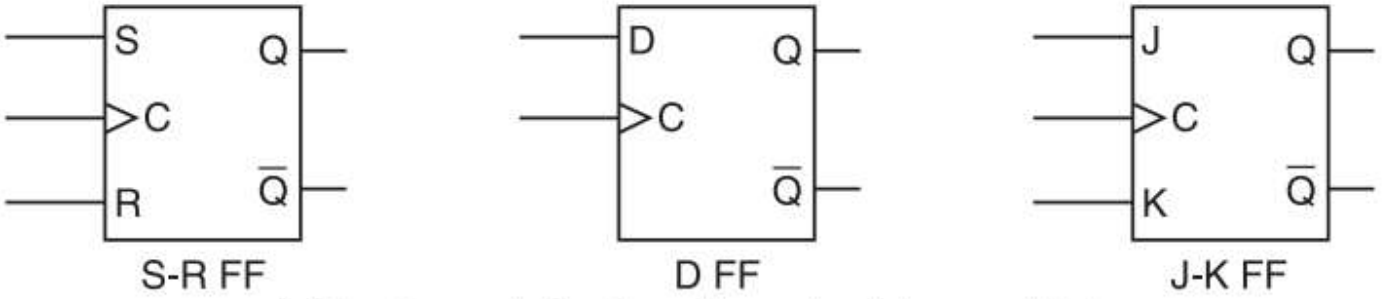


(a) T flip-flop from JK flip-flop (b) Logic symbol

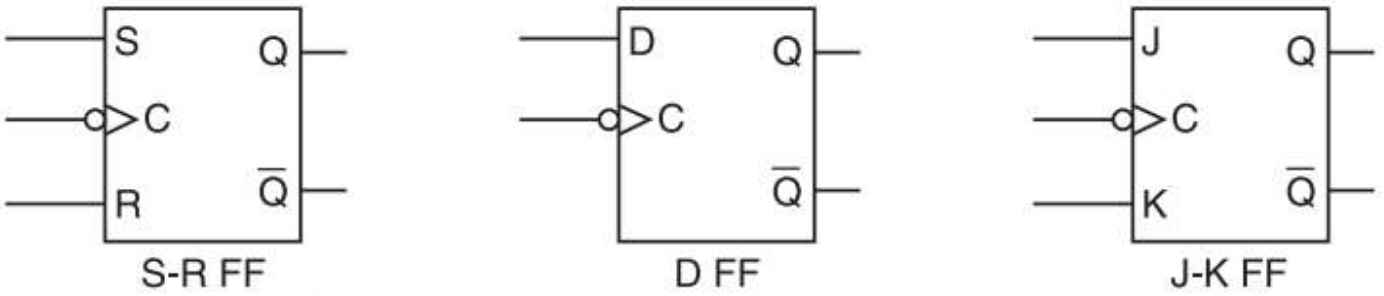
C	T	Q_n	Q_{n+1}	State
↑	0	0	0	No Change (NC)
↑	0	1	1	
↑	1	0	1	Toggle
↑	1	1	0	
0	×	0	0	No Change (NC)
0	×	1	1	

(c) Truth table

Symbols for +ve & -ve Edge-Triggered Flip-Flops:

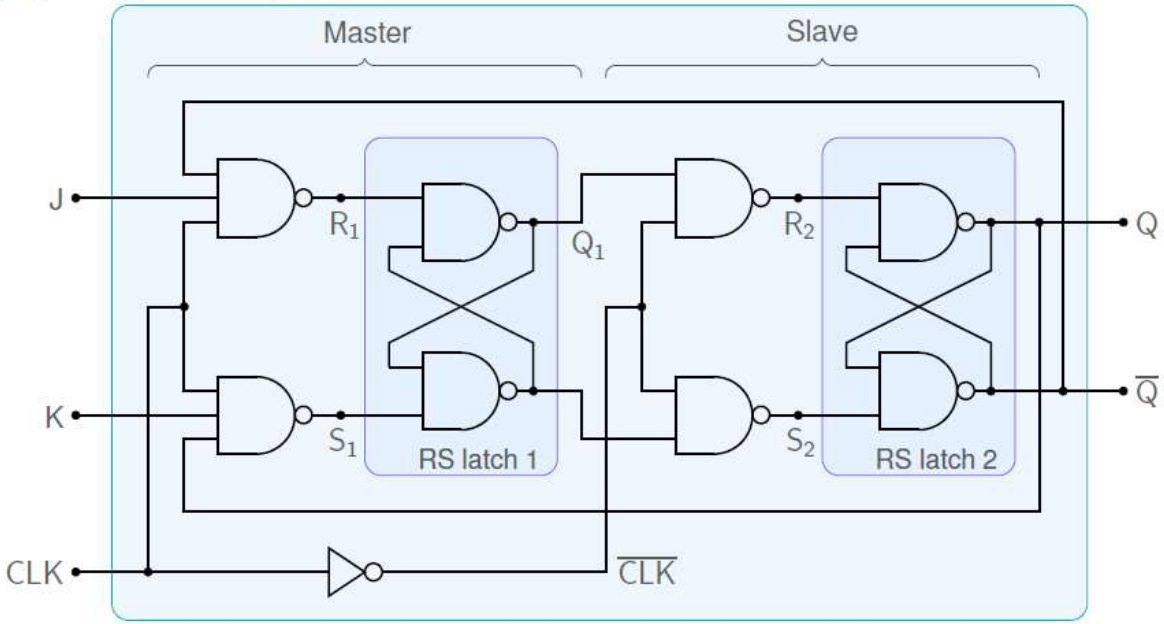


(a) Logic symbols of positive edge-triggered FFs



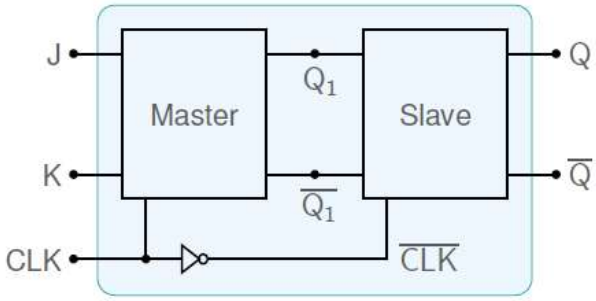
(b) Logic symbols of negative edge-triggered FFs

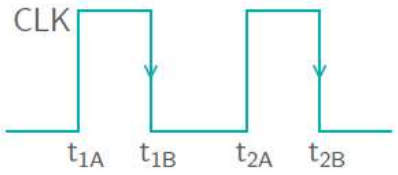
JK flip-flop (Master-Slave)



CLK	J	K	Q_{n+1}
↓	0	0	Q_n
↓	0	1	0
↓	1	0	1
↓	1	1	$\overline{Q_n}$

* When CLK goes high, only the first latch is affected; the second latch retains its previous value (because $\overline{CLK} = 0 \rightarrow R_2 = S_2 = 1$).
* When CLK goes low, the output of the first latch (Q_1) is retained (since $R_1 = S_1 = 1$), and Q_1 can now affect Q.
* In other words, the effect of any changes in J and K appears at the output Q only when CLK makes a transition from 1 to 0.
* Note that the JK flip-flop allows all four input combinations.





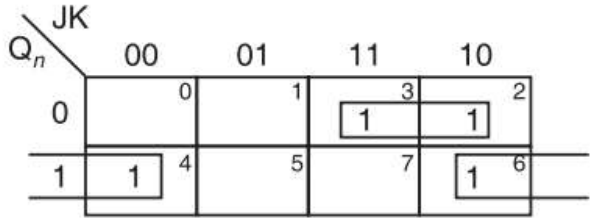
Consider a negative edge-triggered JK flip-flop.

* As seen earlier, when CLK is high (i.e., $t_{1A} < t < t_{1B}$, etc.), the input J and K determine the Master latch output Q_1.
  During this time, *no change* is visible at the flip-flop output Q.
* When the clock goes low, the Slave flip-flop becomes active, making it possible for Q to change.
* In short, although the flip-flop output Q can only change *after* the active edge, (t_{1B}, t_{2B}, etc.), the new Q value is determined by J and K values just *before* the active edge.

Characteristic Equations for Flip-Flops:

Present state	Inputs		Next state
Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Excitation requirements of JK flip-flop



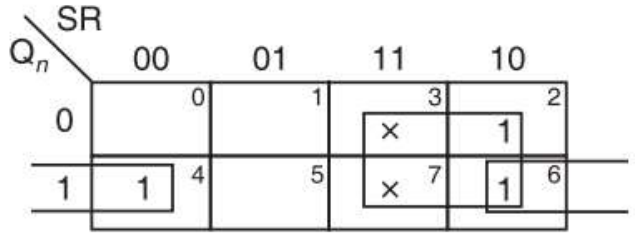
K-map for Q_{n+1}

The characteristic equation of a JK flip-flop is

$Q_{n+1} = \overline{Q}_n J + Q_n \overline{K}$

Present state	Inputs		Next state
Q_n	S	R	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
1	0	0	1
1	0	1	0
1	1	0	1

Excitation requirements of SR flip-flop



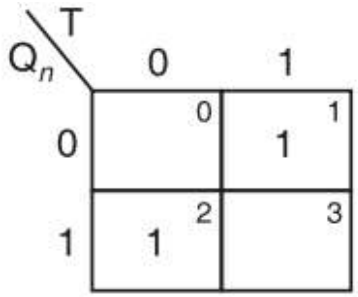
K-map for Q_{n+1} of SR flip-flop

The characteristic equation of SR flip-flop

$Q_{n+1} = S + Q_n \overline{R}$

Present state	Input	Next state
Q_n	T	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Excitation requirements of T flip-flop



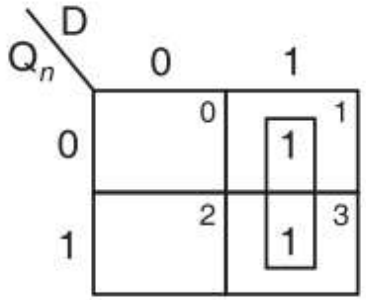
K-map for Q_{n+1} of T flip-flop

The characteristic equation of T flip-flop is

$$Q_{n+1} = \overline{Q}_n T + Q_n \overline{T}$$

Present state	Input	Next state
Q_n	D	Q_{n+1}
0	0	0
0	1	1
1	0	0
1	1	1

Excitation requirements of D flip-flop



K-map for Q_{n+1} of D flip-flop

The characteristic equation of D flip-flop is $Q_{n+1} = D$

Excitation Tables:

During the design process, we usually know the transition from present state to next state and wish to find the flip-flop input conditions that will cause the required transition. For this reason, we need **a table that lists the required inputs for a given change of state.** Such a list is called an **excitation table**.

S-R Flip Flop			
Q(t)	Q(t+1)	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

J-K Flip Flop			
Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

D Flip Flop		
Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

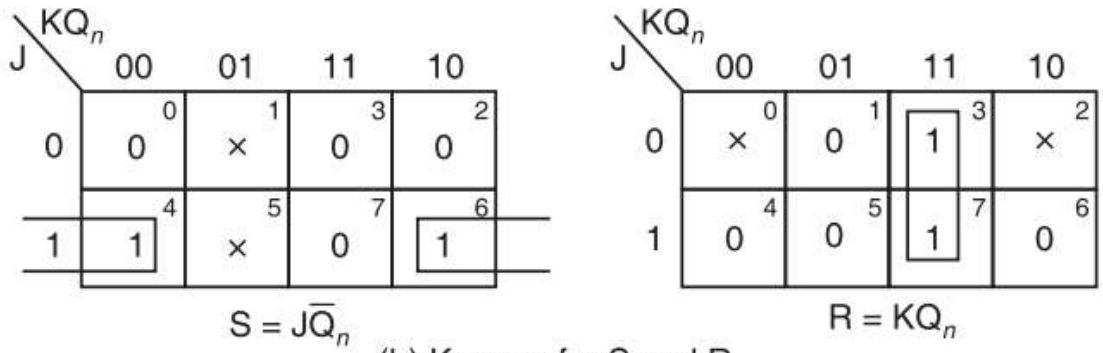
T Flip Flop		
Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

CONVERSION OF FLIP-FLOPS:

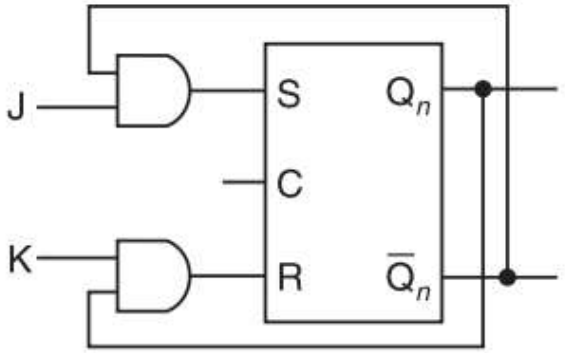
S-R flip-flop to J-K flip-flop:

External Inputs		Present State	Next State	Flip-flop Inputs	
J	K	Q_n	Q_{n+1}	S	R
0	0	0	0	0	×
0	0	1	1	×	0
0	1	0	0	0	×
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	×	0
1	1	0	1	1	0
1	1	1	0	0	1

(a) Conversion table



(b) K-maps for S and R

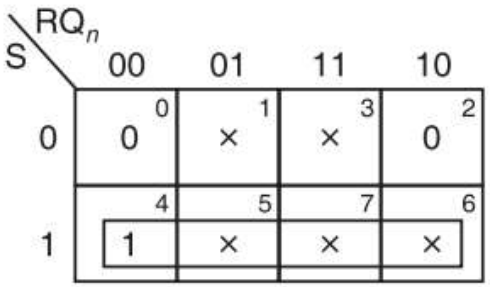


(c) Logic diagram

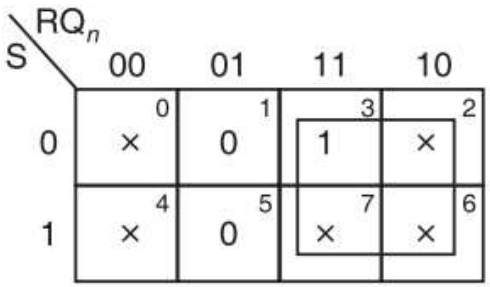
J-K flip-flop to S-R flip-flop:

External Inputs		Present State	Next State	Flip-flop Inputs	
S	R	Q_n	Q_{n+1}	J	K
0	0	0	0	0	×
0	0	1	1	×	0
0	1	0	0	0	×
0	1	1	0	×	1
1	0	0	1	1	×
1	0	1	1	×	0

(a) Conversion table

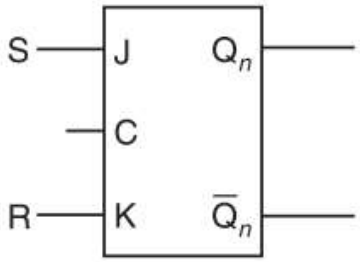


J = S



K = R

(b) K-maps for J and K

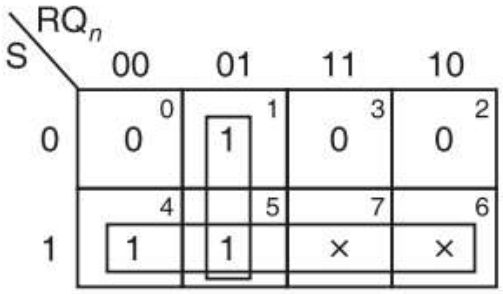


(c) Logic diagram

D flip-flop to S-R flip-flop:

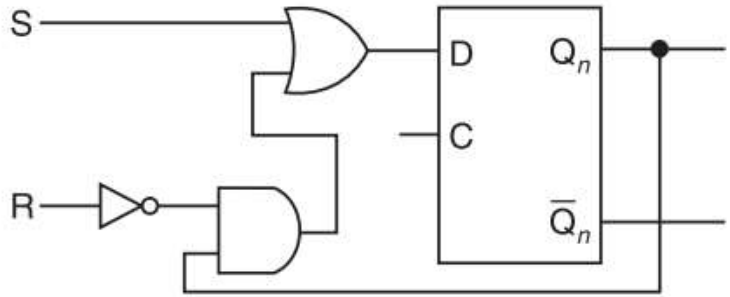
External Inputs		Present State	Next State	Flip-flop Input
S	R	Q_n	Q_{n+1}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1

(a) Conversion table



$D = S + \overline{R}Q_n$

(b) K-map for D



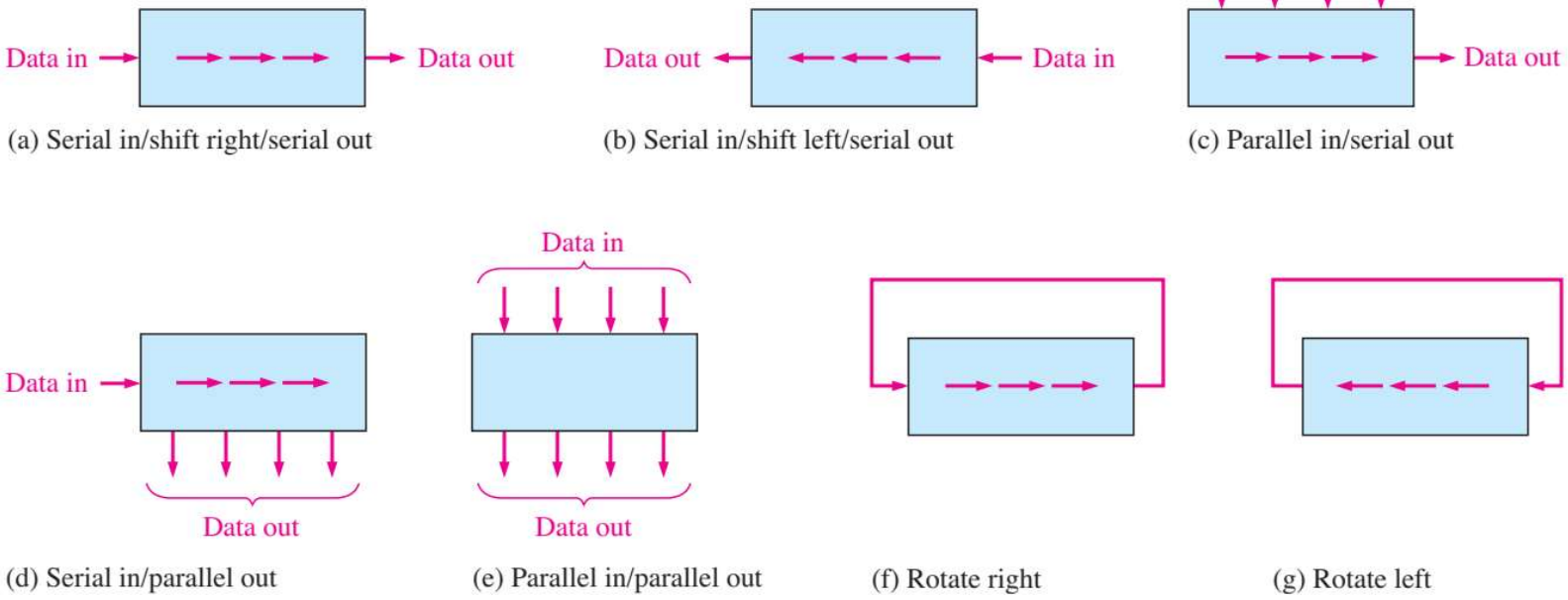
(c) Logic diagram

Shift Registers:

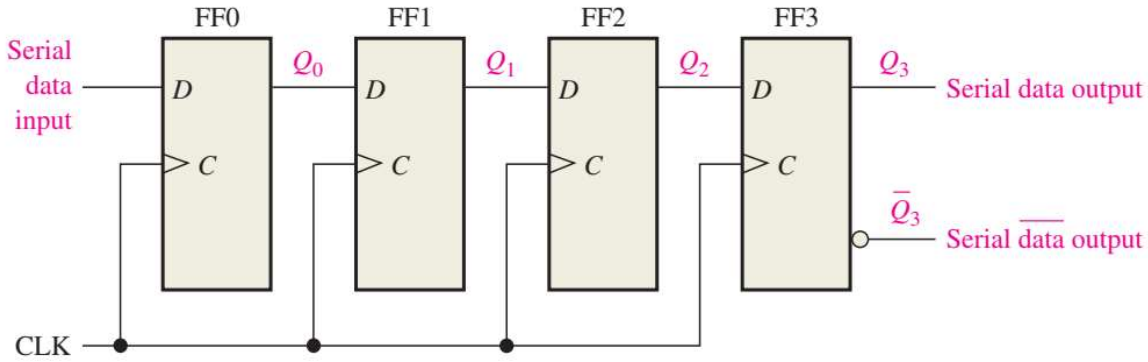
- A register is a group of binary cells (flip-flops) suitable for holding binary information, since each flip-flop is capable of storing 1 bit of information.
- An n-bit register has a group of n flip-flops and is capable of storing any binary information containing n bits.
- In addition to the flip-flops, a register may have combinational gates that perform certain data-processing tasks.
- The flip-flops hold binary information and the gates control when and how new information is transferred into the register.
- A register capable of shifting its binary information either to the right or to the left is called a ***shift register***.

- The logical configuration of a shift register consists of a chain of flip-flops connected in cascade, with the output of one flip-flop connected to the input of the next flip-flop. All flip-flops receive a common clock pulse that causes the shift from one stage to the next.

Basic Movement of Data in Shift Registers:



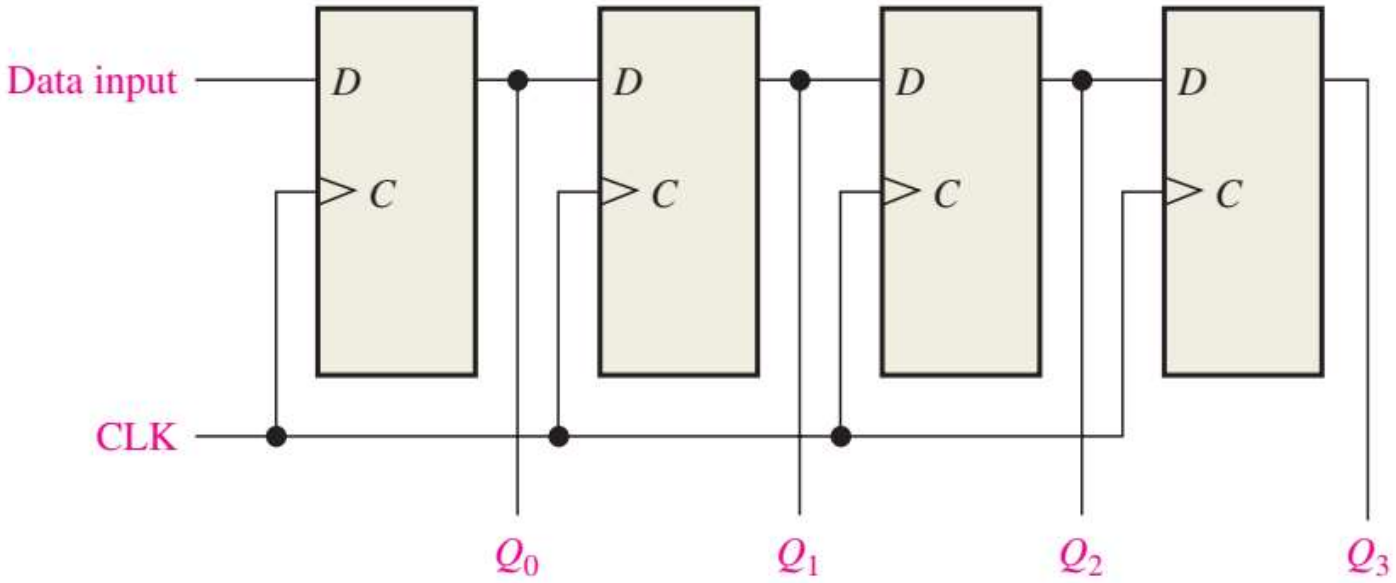
Serial Input – Serial Output (SISO) Shift Register



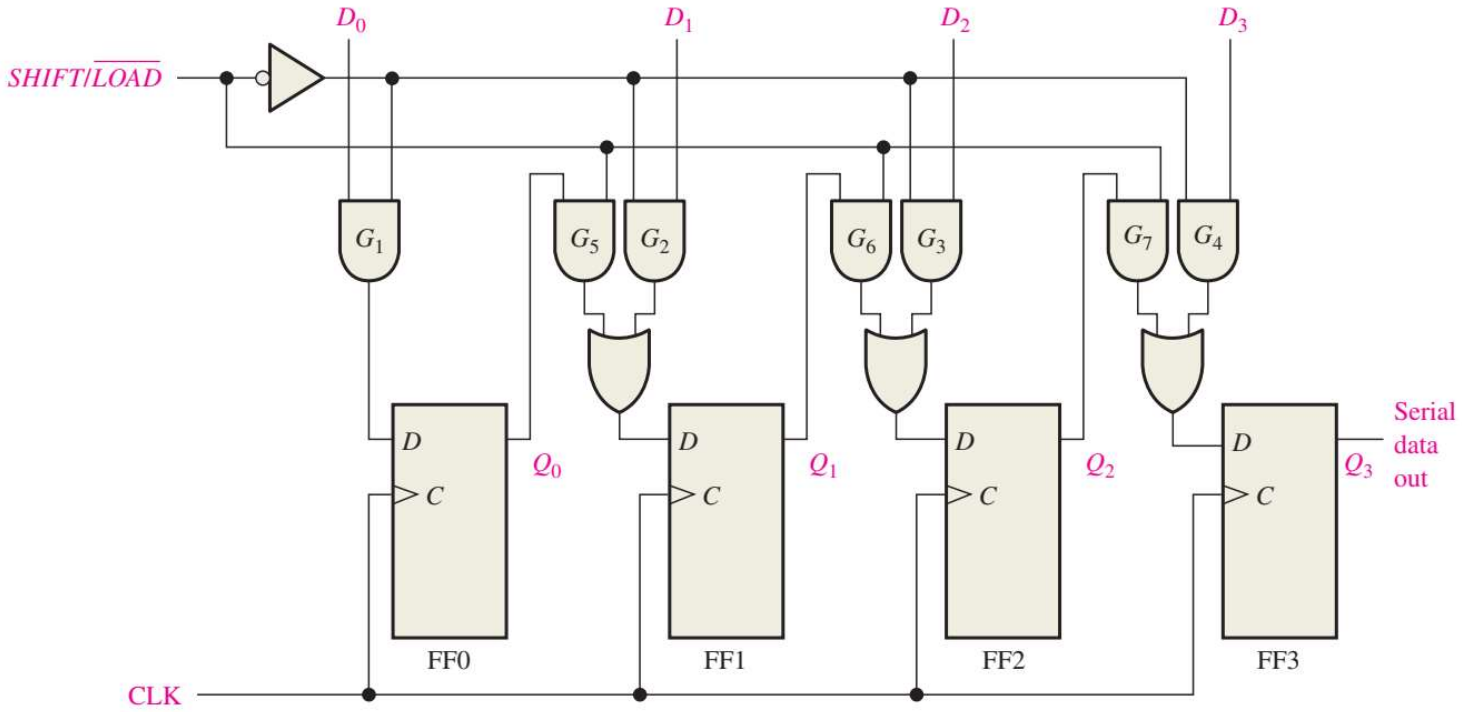
To store binary data "0101" the shifting proceeds as,

CLK	FF0 (Q_0)	FF1 (Q_1)	FF2 (Q_2)	FF3 (Q_3)
Initial	0	0	0	0
1	**0**	0	0	0
2	**1**	**0**	0	0
3	**0**	**1**	**0**	0
4	**1**	**0**	**1**	**0**

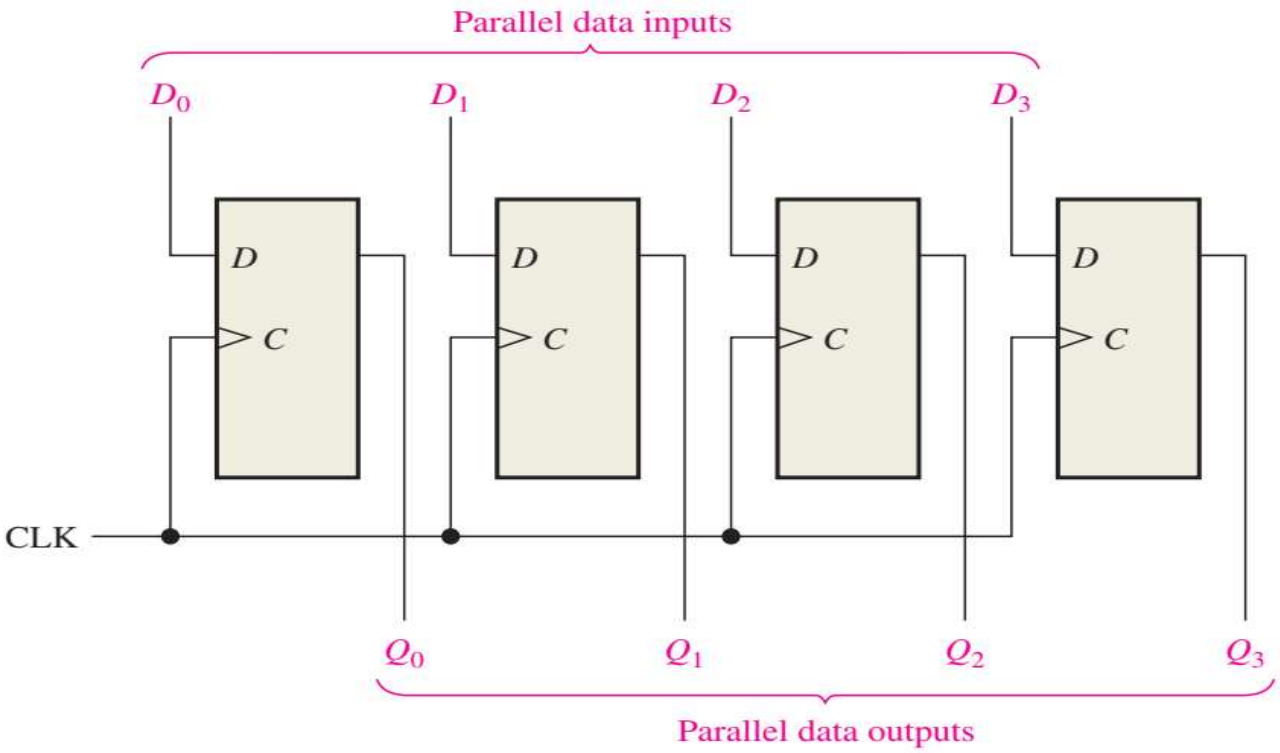
Serial In/Parallel Out Shift Registers



Parallel In/Serial Out Shift Registers

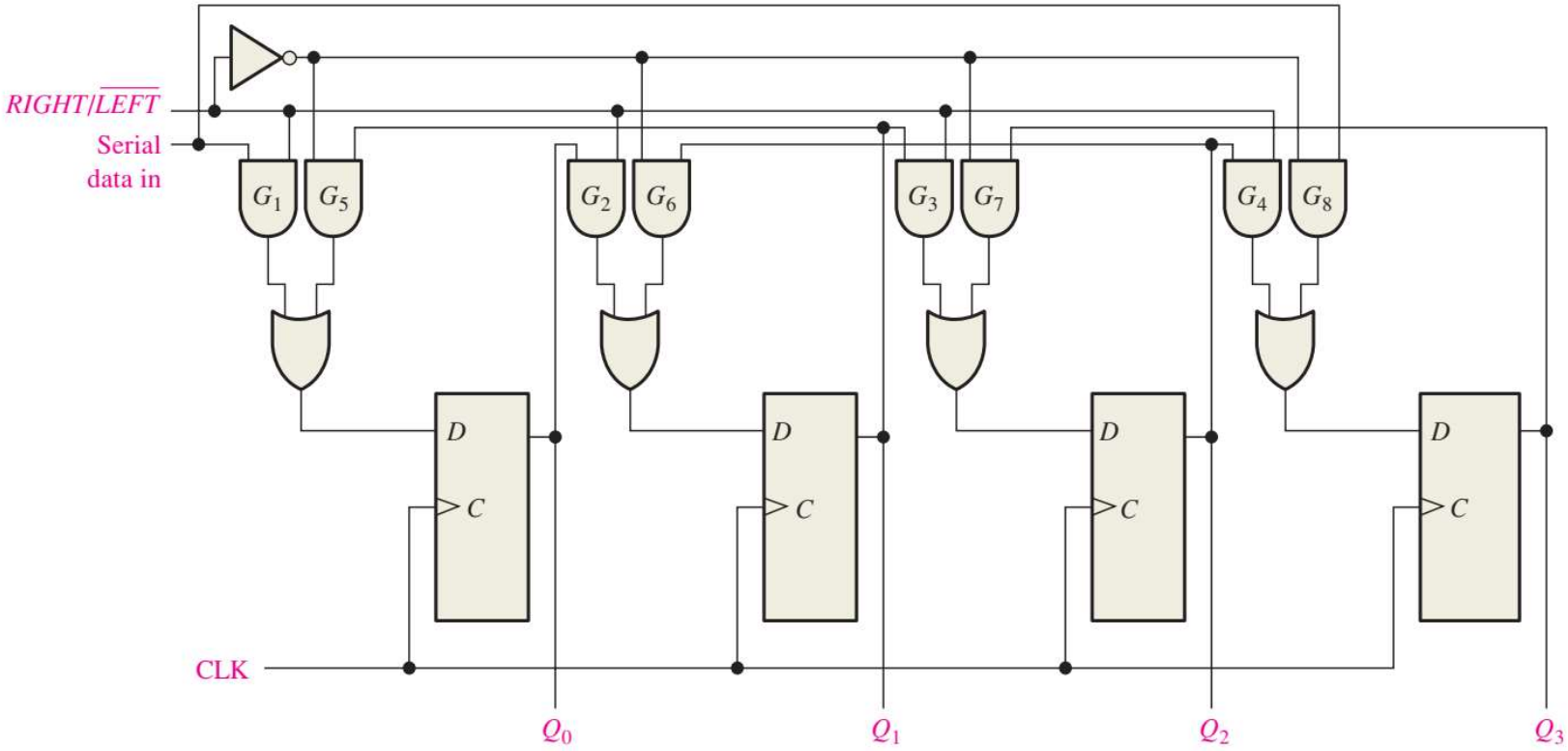


Parallel input – Parallel Output (PIPO) Shift Register



Four – Bit Bidirectional Shift Register

A **bidirectional** shift register is one in which the data can be shifted either left or right. It can be implemented by using gating logic that enables the transfer of a data bit from one stage to the next stage to the right or to the left, depending on the level of a control line.



Counters:

- A counter is a set of Flip-Flops whose states change in response to the pulse applied at the input to the counter.

- The flip-flops are interconnected in such a way that their combined state at any time is the binary equivalent of the total no. of pulses that have been occurred up to that time.
- Thus the counter is used to count the pulses.

Counters Classifications:

a) Asynchronous counters (also known as, Ripple counters/serial counters):

The flip-flop within the counters are not made to change the states at exactly the same time because the flip-flops are not triggered simultaneously. The clock doesn't directly control the time at which every stage changes its states. It uses T- flip-flops to perform the counting function. The actual hardware involved is th J-K flip-flop connected in toggle mode (J=K=1). It is to note that a D-flip-flop can also be used.

(b) Synchronous counters:

These are clocked in such a way that each flip-flop in the counter is triggered at the same time. This is accomplished by connecting the clock line to each stage of the counter they are faster than asynchronous counter as the propagation delay is less.

Points to be noted :

1. Each count of counter is called state of the counter.
2. The modulus of the counter is equal to the total no. of distinct states (counts) including the zero that the counter can store. In other words, the no. of input pulses that causes the counter to RESET to its initial count is called the modulus of the counter.

   e.g.: 2-bit counter ➔ $2^2 = 4$ ➔ (MOD-4) ➔ (Divide – by – 4)

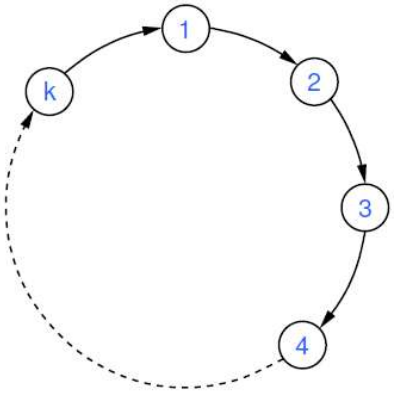
3. An n-bit counter will have n flip-flops and 2^n states and divides the input frequency by 2^n. Hence it is called as "divide-by-2^n" counter.
4. A counter may have shortened modulus. This type of counter does not utilize all the possible states. Some of the states are not utilized, i.e. remains invalid.

e.g.: MOD-6 counter ➔ Minimum no. of flip-flops required = 3 (as $2^3 = 8$)

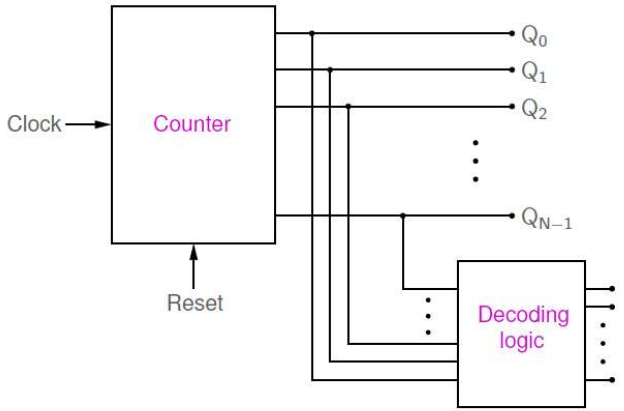
MOD-10 counter ➔ Minimum no. of flip-flops required = 4 (as $2^4 = 10$)

5. The no. of flip-flops required to construct the “MOD-N” counter equals to the smallest integer “n” for which $N \leq 2^n$
6. In an asynchronous counter invalid states are bypassed by providing suitable feedback, whereas, in synchronous counter the invalid states are taken care by treating the excitation as don’t care conditions.
7. The least significant bit (LSB) is the bit that changes most often. In ripple counters the LSB is the Q output of the flip-flop to which external CLOCK Pulse is applied.
8. The final state of the counter sequence is called the terminal count.

Counters



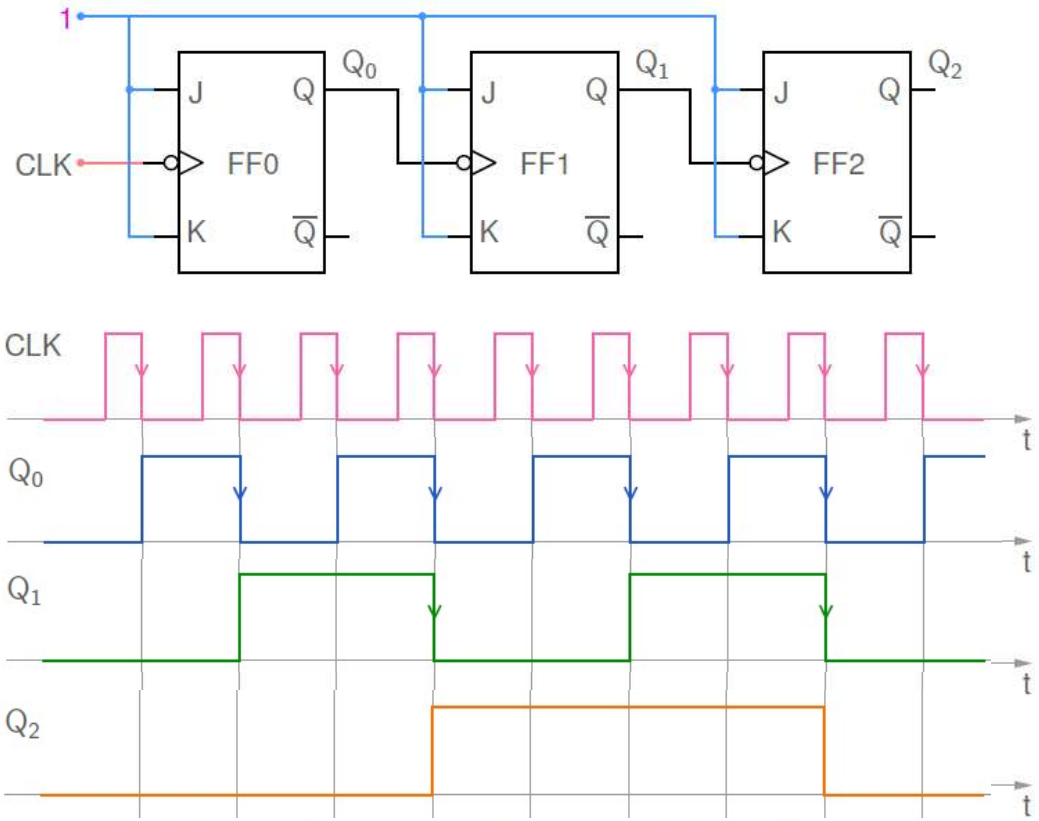
State transition diagram



General configuration

* A counter with k states is called a modulo-k (mod-k) counter.
* A counter can be made with flip-flops, each flip-flop serving as a memory element with two states (0 or 1).
* If there are N flip-flops in a counter, there are 2^N possible states (since each flip-flop can have $Q = 0$ or $Q = 1$). It is possible to exclude some of these states.
  $\rightarrow$ N flip-flops can be used to make a mod-k counter with $k \leq 2^N$.
* Typically, a reset facility is also provided, which can be used to force a certain state to initialize the counter.

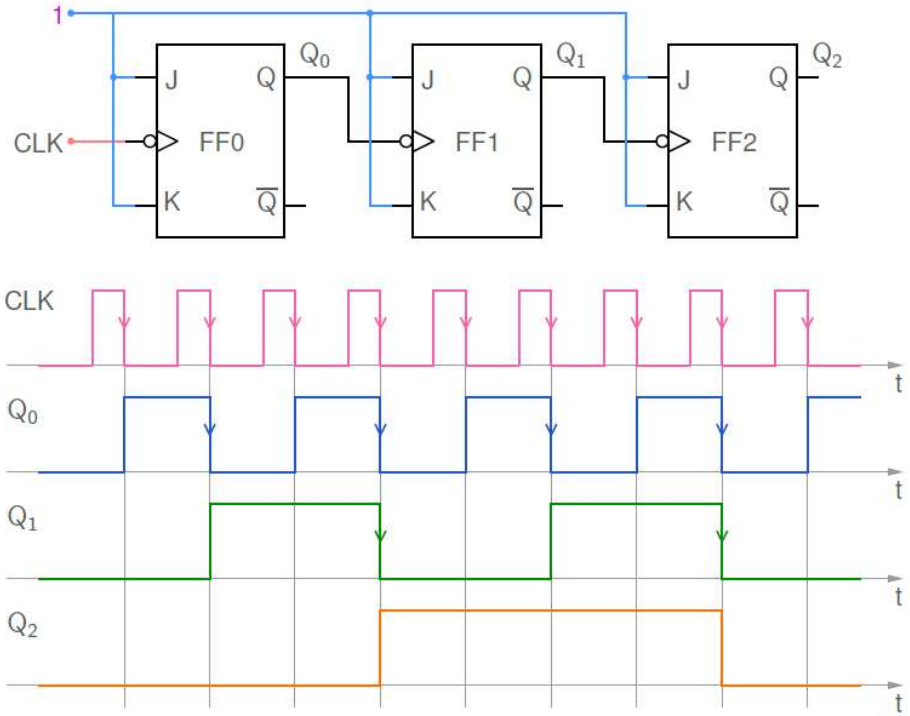
Binary Ripple counter(Asynchronous Counter)



Q_2	Q_1	Q_0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1
0	0	0

repeats

* $J = K = 1$ for all flip-flops. Let $Q_0 = Q_1 = Q_2 = 0$ initially.
* Since $J = K = 1$, each flip-flop will toggle when an active (in this case, negative) clock edge arrives.
* For FF1 and FF2, Q_0 and Q_1, respectively, provide the clock.



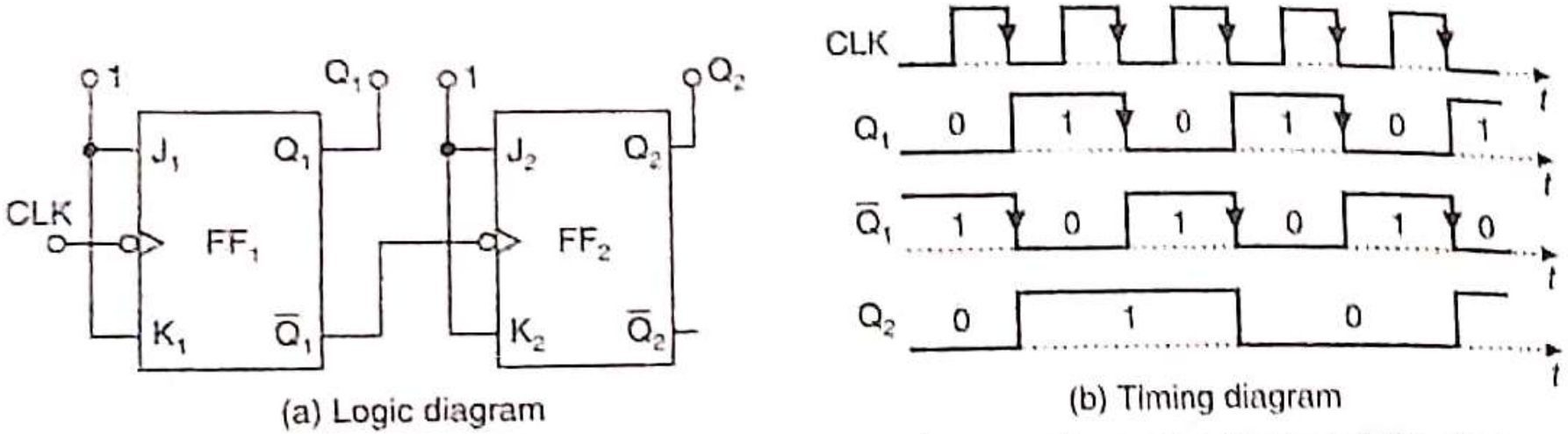
Q_2	Q_1	Q_0	
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	
0	0	0	repeats

* The counter has 8 states, $Q_2 Q_1 Q_0 = 000$, 001, 010, 011, 100, 101, 110, 111.
  $\rightarrow$ it is a mod-8 counter. In particular, it is a *binary, mod-8, up* counter (since it counts *up* from 000 to 111).

* If the clock frequency is f_c, the frequency at the Q_0, Q_1, Q_2 outputs is $f_c/2$, $f_c/4$, $f_c/8$, respectively. For this counter, therefore, div-by-2, div-by-4, div-by-8 outputs are already available, without requiring decoding logic.

* This type of counter is called a "ripple" counter since the clock transitions *ripple* through the flip-flops.

Asynchronous 2-bit down counter using Negative edge triggered Flip-Flops

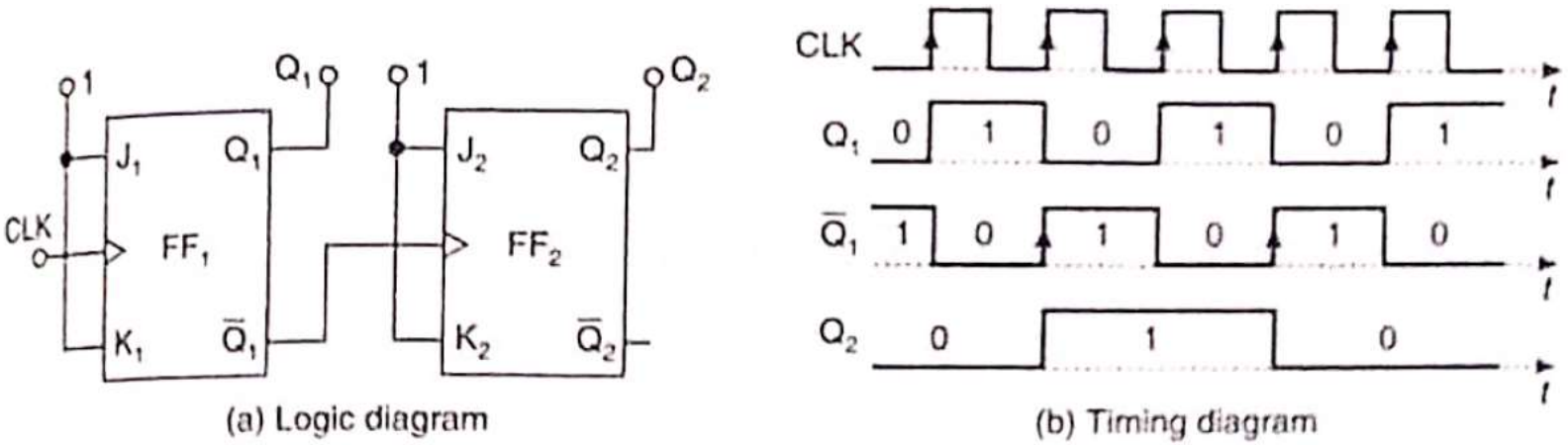


(a) Logic diagram

(b) Timing diagram

Asynchronous 2-bit down-counter using negative edge-triggered flip-flops.

2-bit Ripple Up-Counter Using Positive Edge triggered Flip-Flops

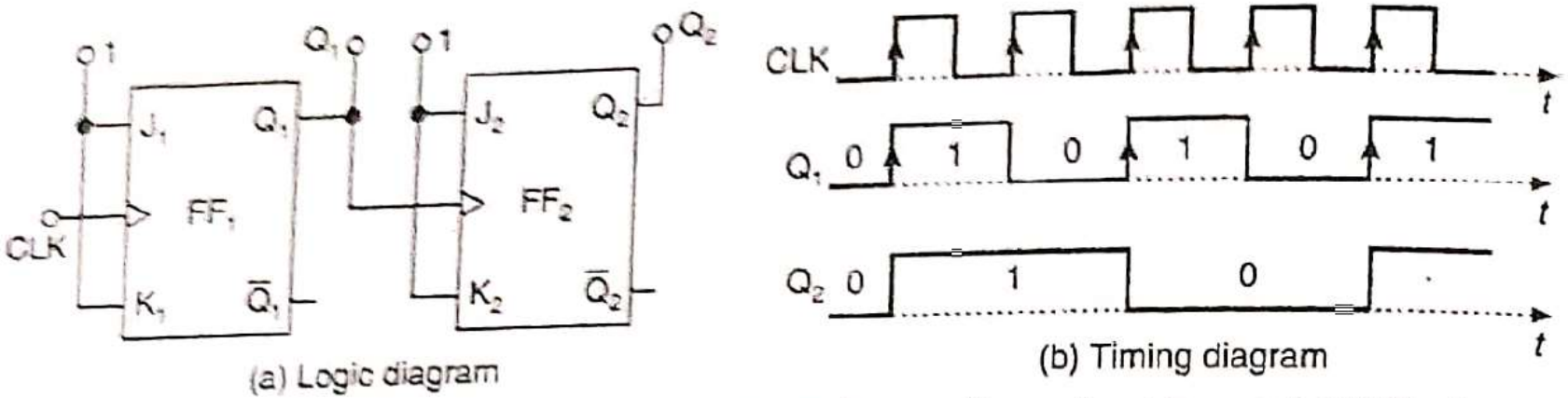


(a) Logic diagram

(b) Timing diagram

Asynchronous 2-bit up-counter using positive-edge triggered J-K flip-flops.

2-bit Ripple Down-Counter Using Positive Edge triggered Flip-Flops

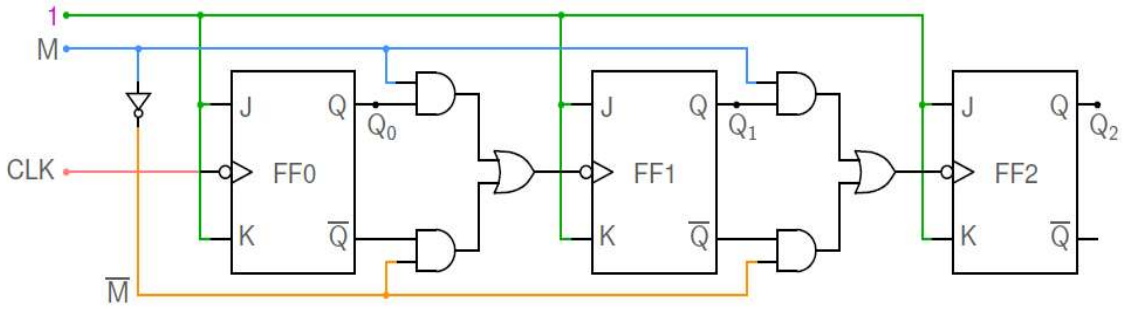


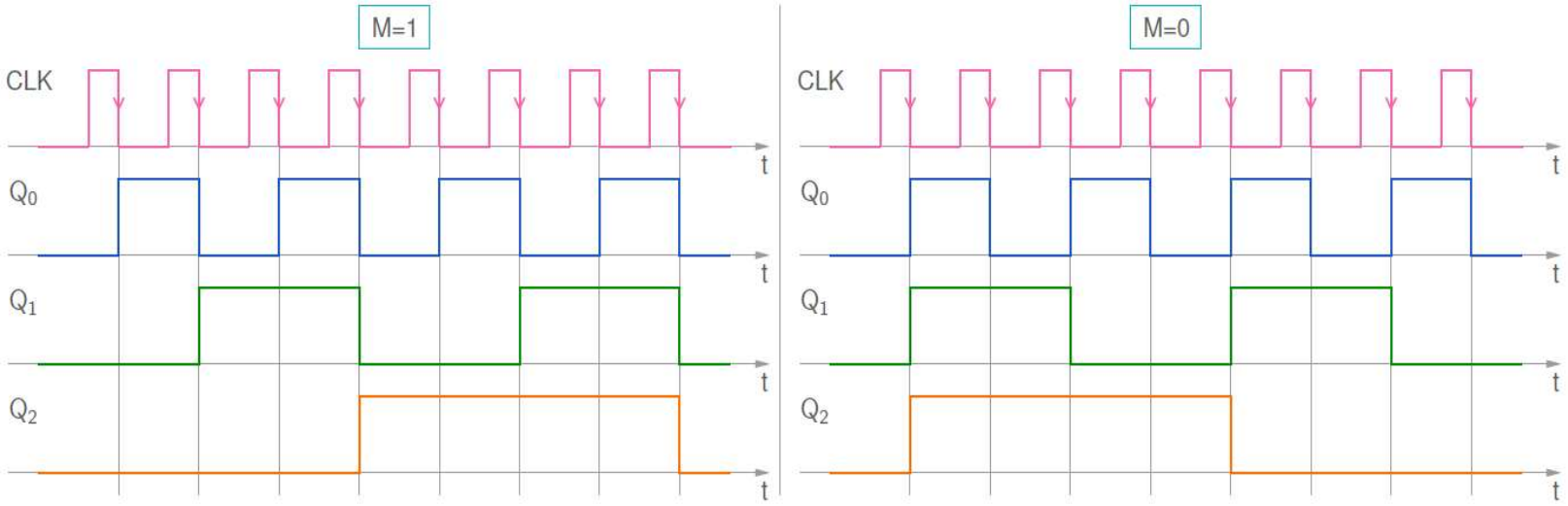
(a) Logic diagram

(b) Timing diagram

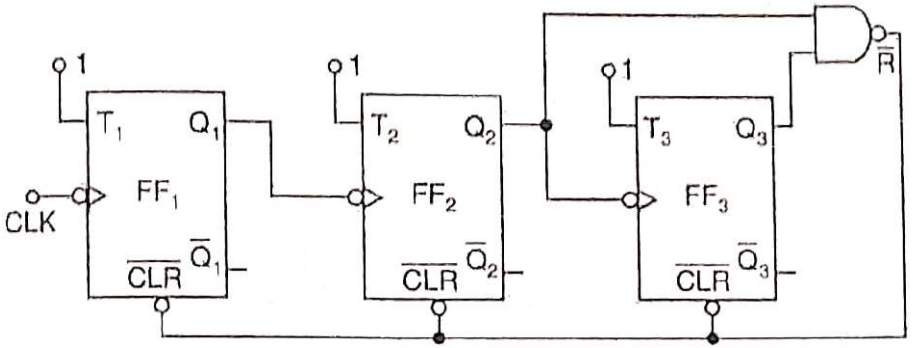
Asynchronous 2-bit down-counter using positive edge-triggered J-K flip-flops.

3-bit Up-Down binary ripple counter

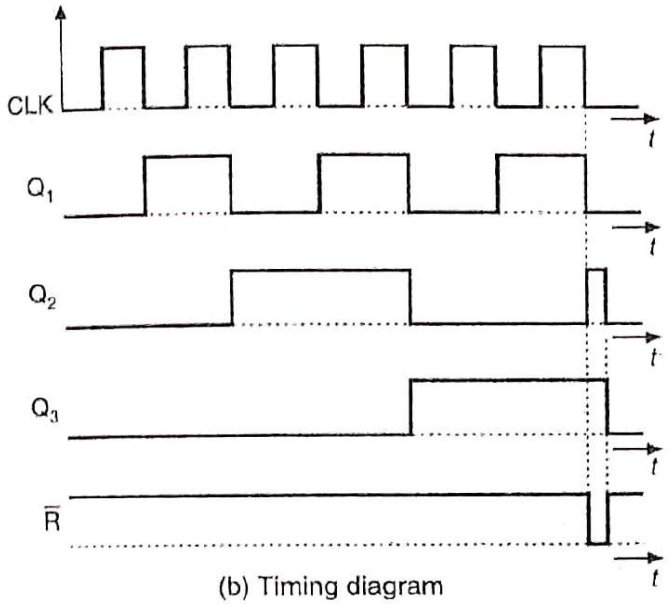




Design of a Mod-6 Asynchronous Counter using T- FFs



(a) Logic diagram

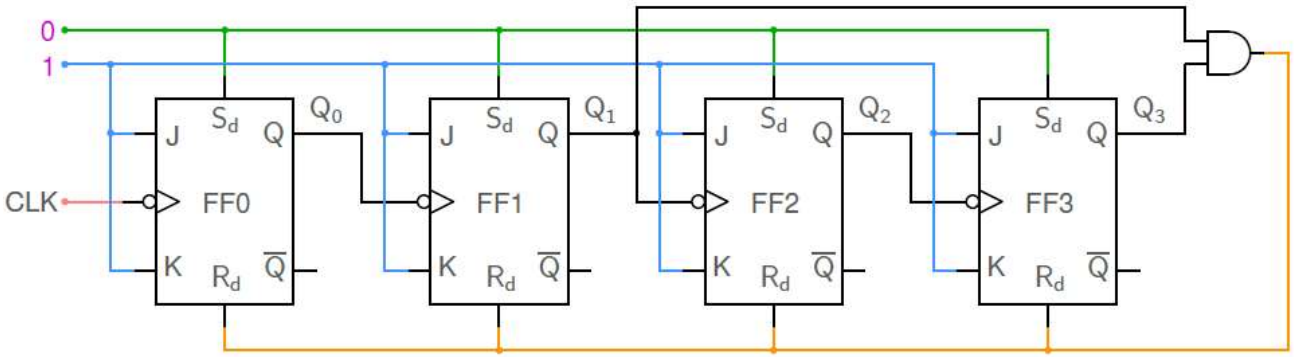


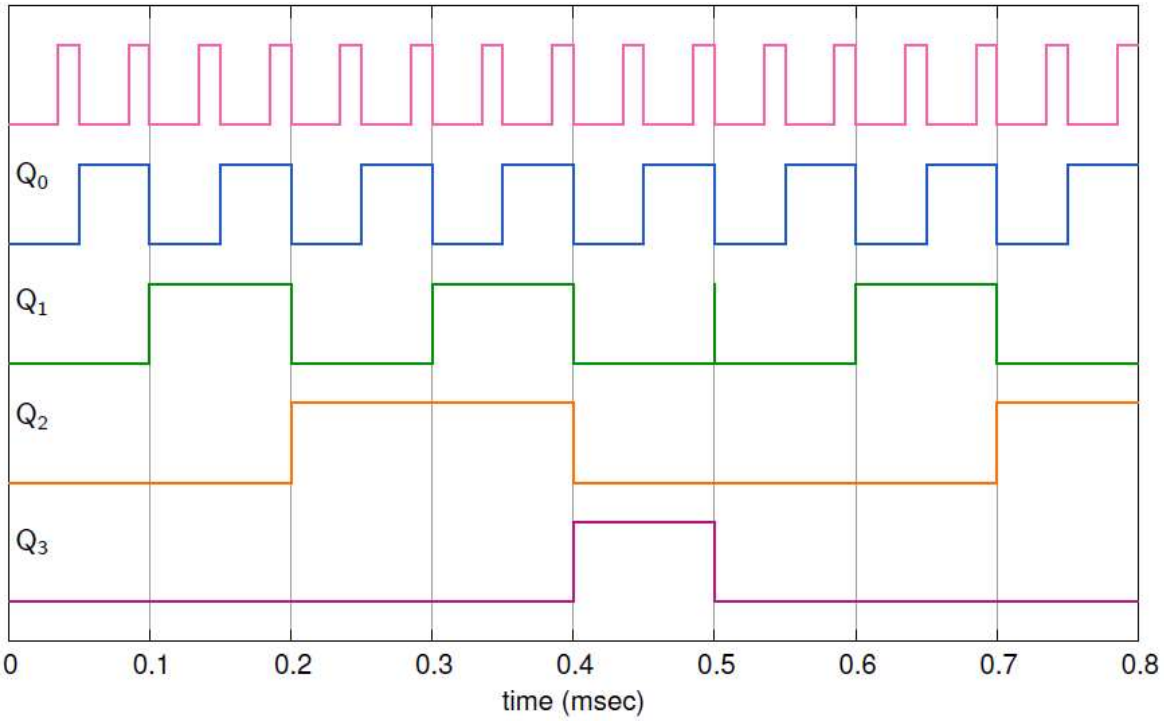
(b) Timing diagram

After pulses	State Q_3	Q_2	Q_1	R
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
	↓	↓	↓	
	0	0	0	0
7	0	0	1	0

(c) Table for R

Decade counter using direct inputs

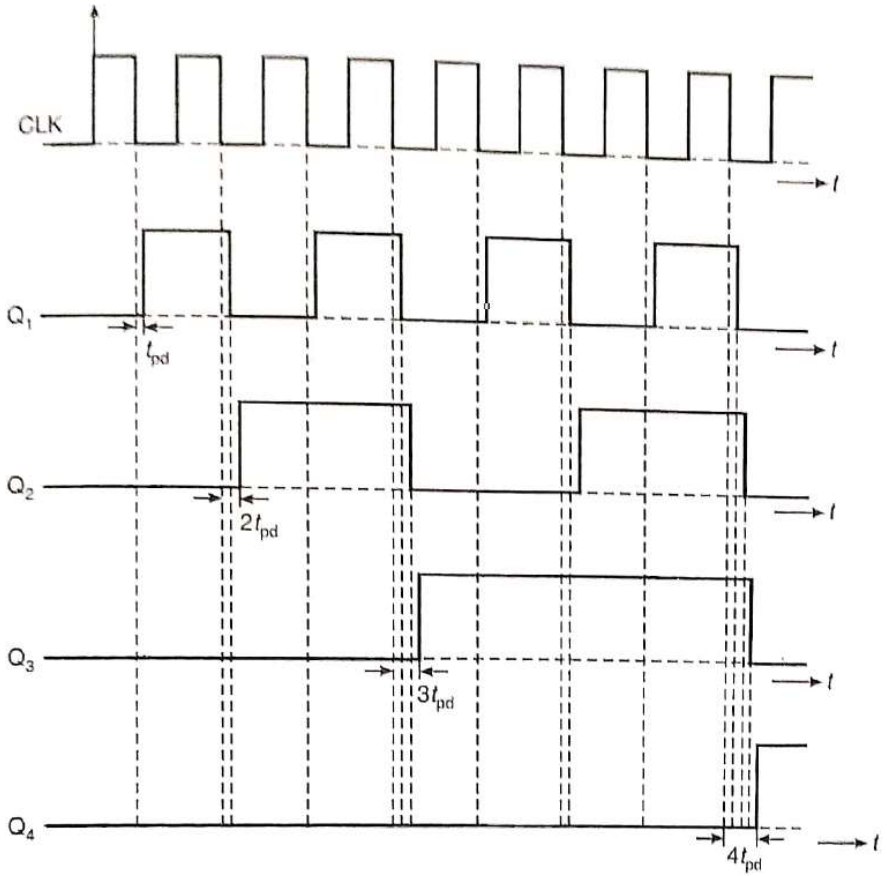




Q_3	Q_2	Q_1	Q_0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
0	0	0	0

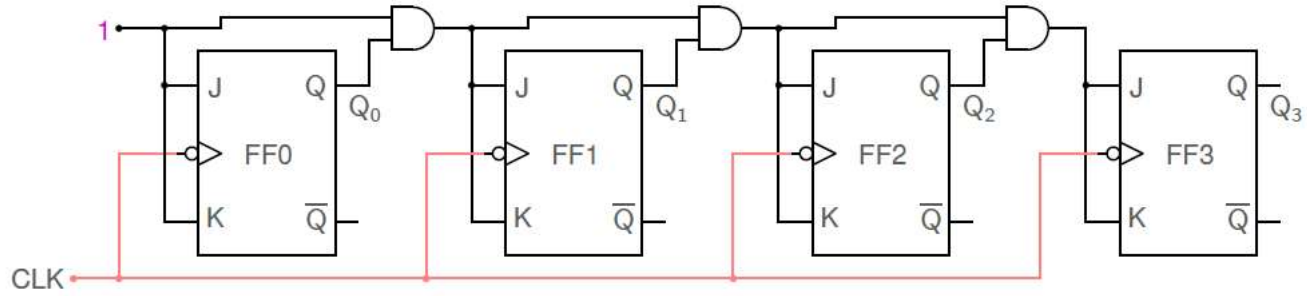
repeats

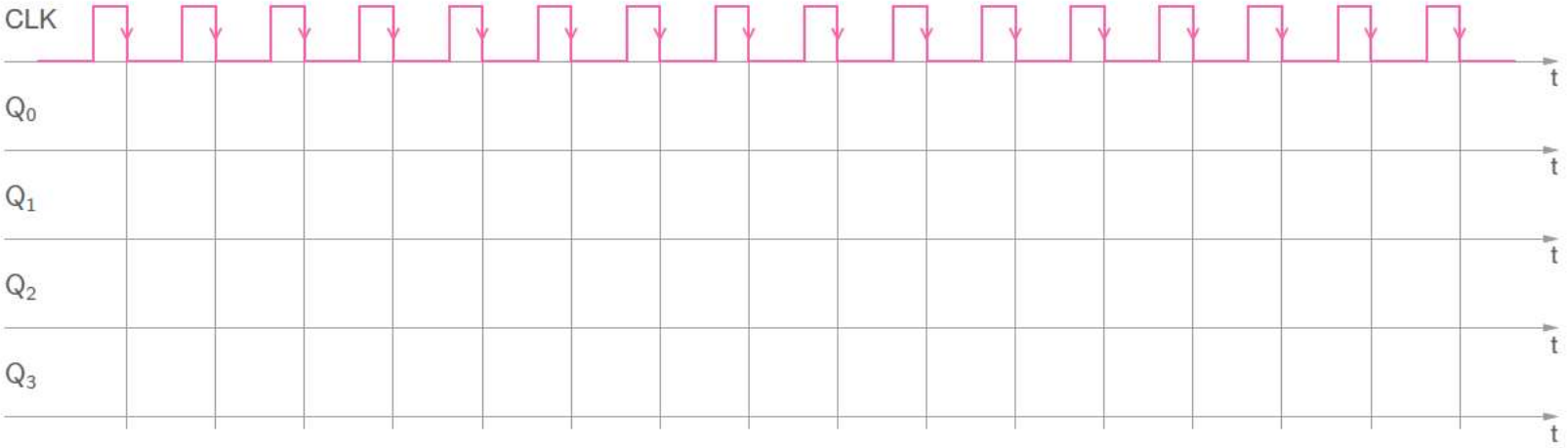
Effects of Propagation Delay in Ripple counters



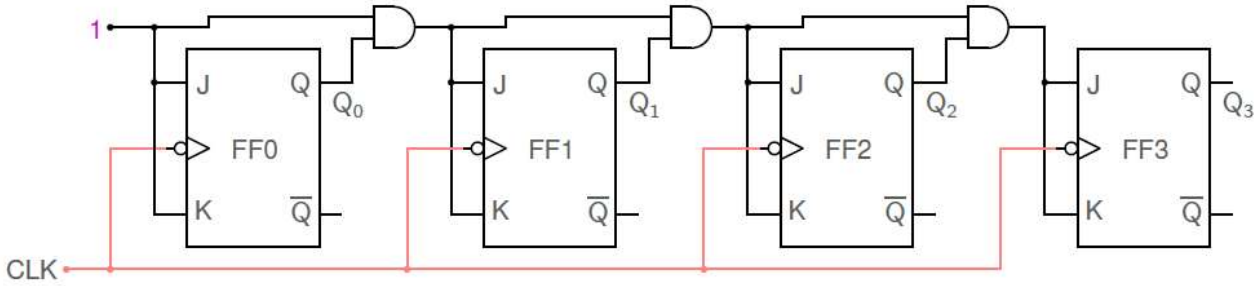
Timing diagram considering propagation delay (no skipping of states).

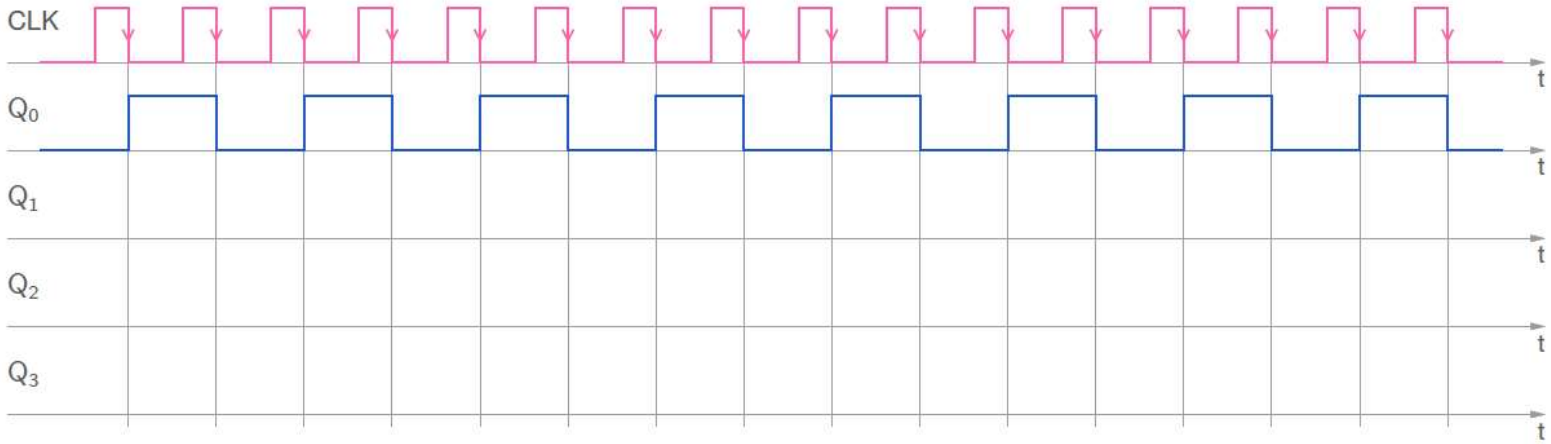
Synchronous Counters



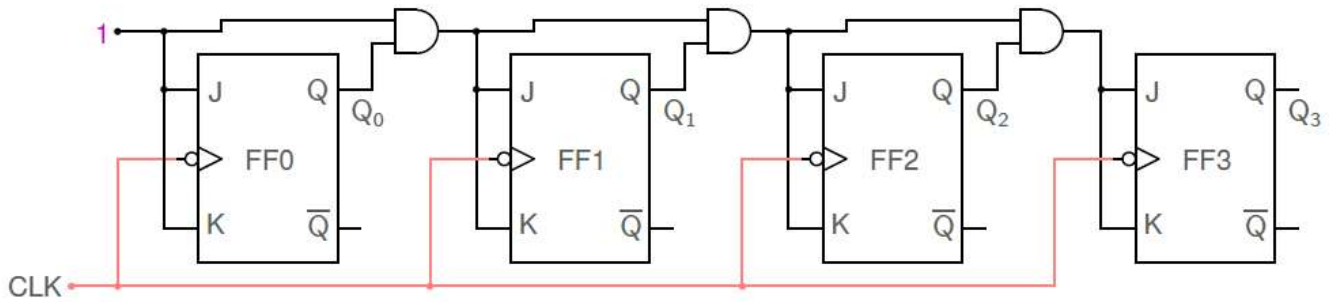


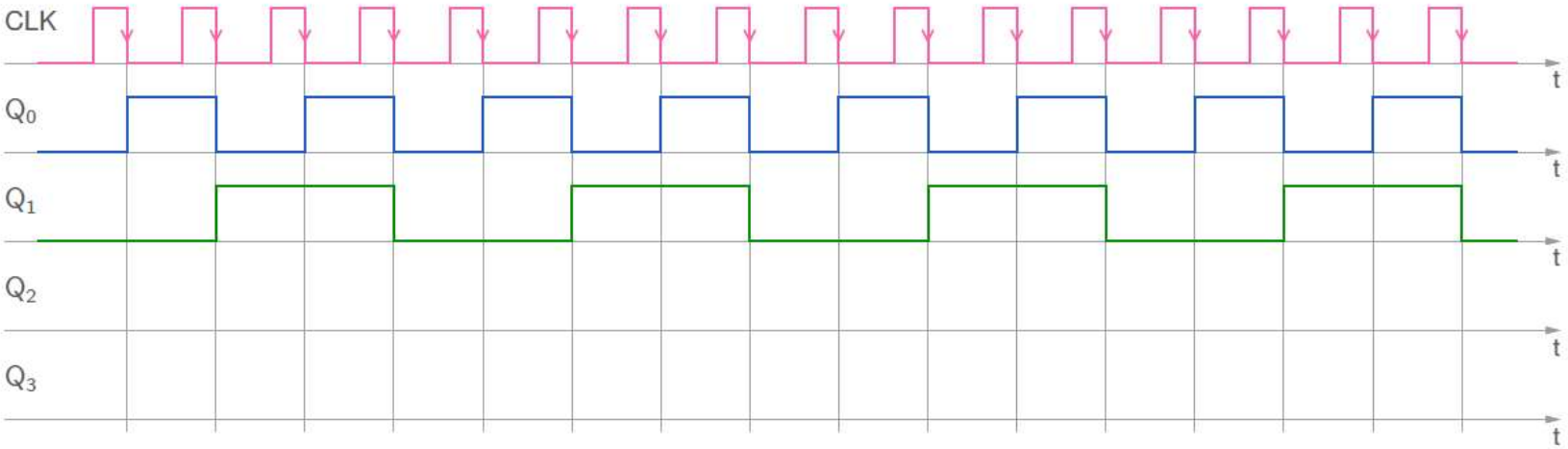
* Since all flip-flops are driven by the same clock, the counter is called a "synchronous" counter.
* $J_0 = K_0 = 1$, $J_1 = K_1 = Q_0$, $J_2 = K_2 = Q_1 Q_0$, $J_3 = K_3 = Q_2 Q_1 Q_0$.
* FF0 toggles after every active edge.
  FF1 toggles if $Q_0 = 1$ (just before the active clock edge); else, it retains its previous state. (Similarly, for FF2 and FF3)



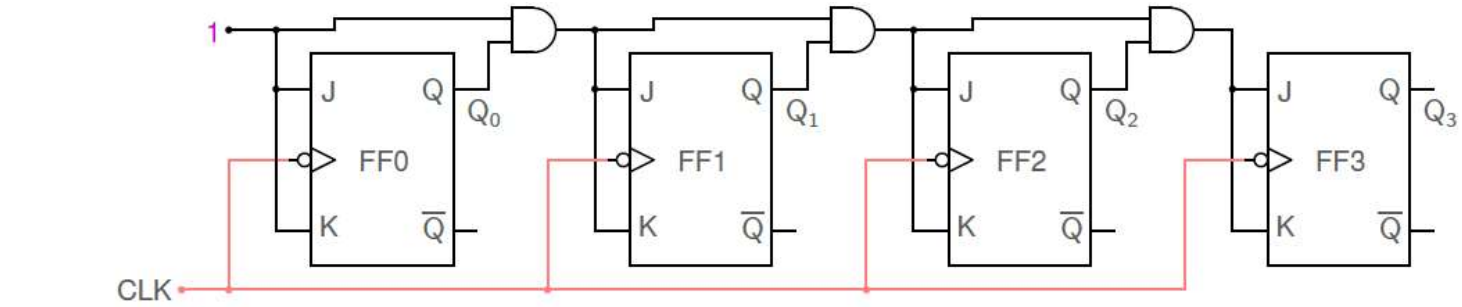


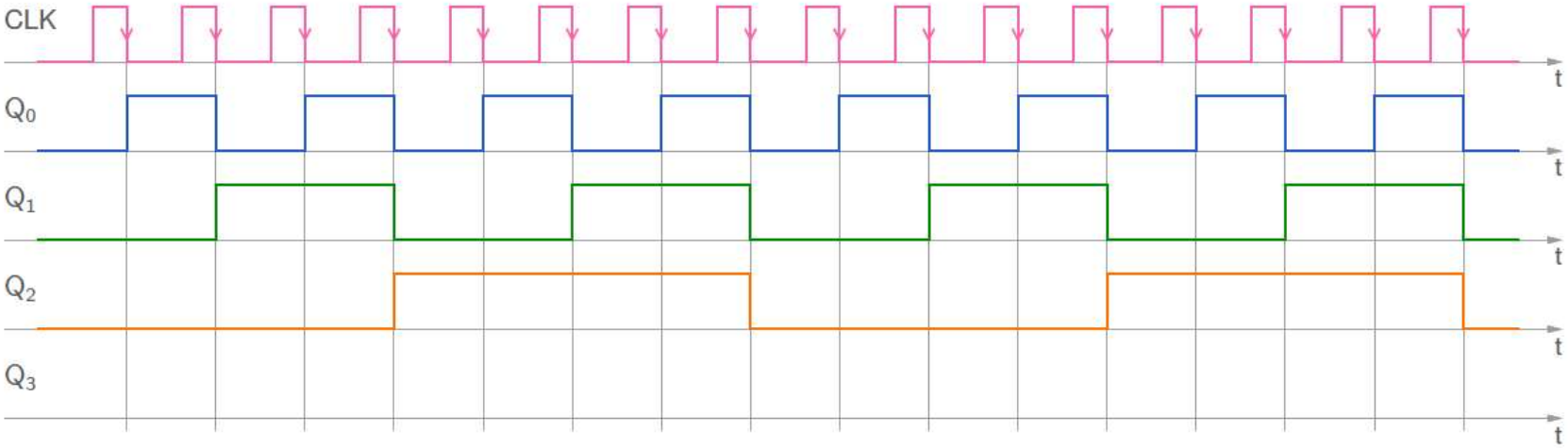
* Since all flip-flops are driven by the same clock, the counter is called a "synchronous" counter.
* $J_0 = K_0 = 1$, $J_1 = K_1 = Q_0$, $J_2 = K_2 = Q_1 Q_0$, $J_3 = K_3 = Q_2 Q_1 Q_0$.
* FF0 toggles after every active edge.
  FF1 toggles if $Q_0 = 1$ (just before the active clock edge); else, it retains its previous state. (Similarly, for FF2 and FF3)



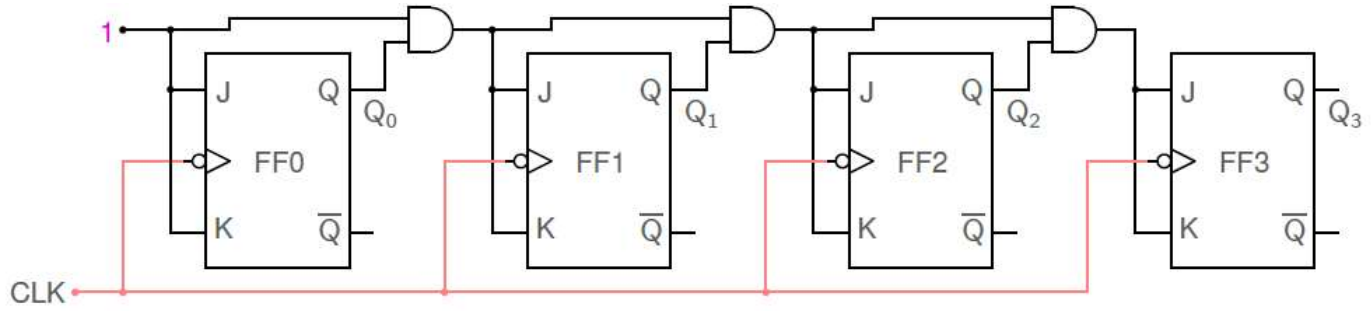


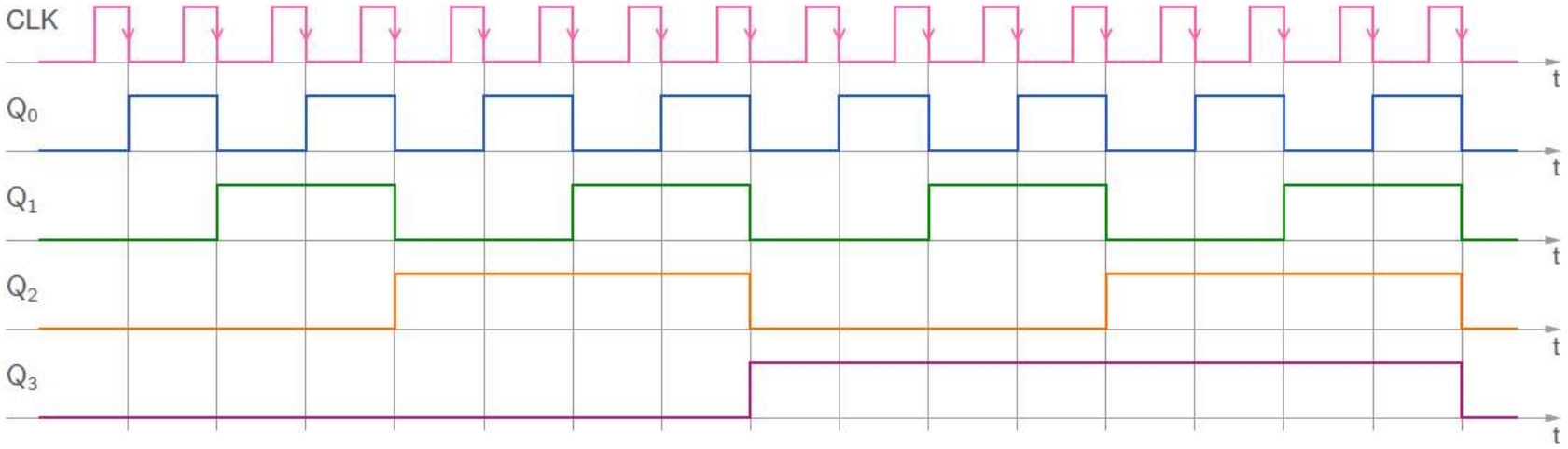
* Since all flip-flops are driven by the same clock, the counter is called a "synchronous" counter.
* $J_0 = K_0 = 1$, $J_1 = K_1 = Q_0$, $J_2 = K_2 = Q_1 Q_0$, $J_3 = K_3 = Q_2 Q_1 Q_0$.
* FF0 toggles after every active edge.
  FF1 toggles if $Q_0 = 1$ (just before the active clock edge); else, it retains its previous state. (Similarly, for FF2 and FF3)





* Since all flip-flops are driven by the same clock, the counter is called a "synchronous" counter.
* $J_0 = K_0 = 1$, $J_1 = K_1 = Q_0$, $J_2 = K_2 = Q_1 Q_0$, $J_3 = K_3 = Q_2 Q_1 Q_0$.
* FF0 toggles after every active edge.
  FF1 toggles if $Q_0 = 1$ (just before the active clock edge); else, it retains its previous state. (Similarly, for FF2 and FF3)





* Since all flip-flops are driven by the same clock, the counter is called a "synchronous" counter.
* $J_0 = K_0 = 1$, $J_1 = K_1 = Q_0$, $J_2 = K_2 = Q_1 Q_0$, $J_3 = K_3 = Q_2 Q_1 Q_0$.
* FF0 toggles after every active edge.
  FF1 toggles if $Q_0 = 1$ (just before the active clock edge); else, it retains its previous state. (Similarly, for FF2 and FF3)
* From the waveforms, we see that it is a binary up counter.

Design of Synchronous Counters

- Determine the number of Flip-Flops
- Draw the State diagram
- Select the type of Flip-Flops and draw the Excitation table
- Obtain the Minimal expressions for excitations
- Draw the Logic Diagram.

Excitation Table of Flip-Flops

- It indicates the input required to be applied to the flip flop to take it from present state to the next state.

S-R Flip-Flop:

S-R truth table

S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	?

S-R excitation table

PS	NS	Required inputs	
Q_n	Q_{n+1}	S	R
0	0	0	×
0	1	1	0
1	0	0	1
1	1	×	0

J-K truth table

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q}_n$

J-K excitation table

PS	NS	Required inputs	
Q_n	Q_{n+1}	J	K
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0

D truth table

D	Q_{n+1}
0	0
1	1

D excitation table

PS	NS	Required input
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

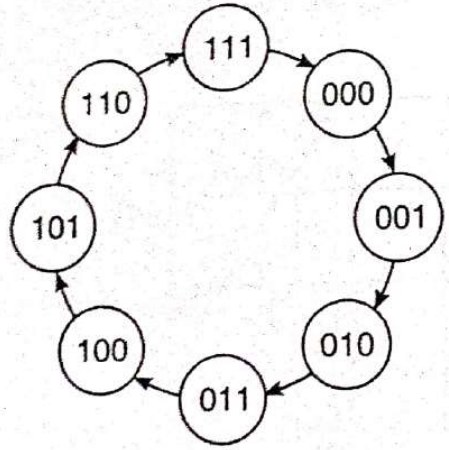
T truth table

T	Q_{n+1}
0	Q_n
1	$\overline{Q}_n$

T excitation table

PS	NS	Required input
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Design of Synchronous 3-bit counter



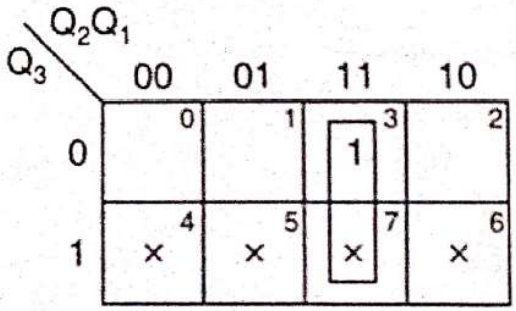
(a) State diagram

PS			NS			Required excitations					
Q_3	Q_2	Q_1	Q_3	Q_2	Q_1	J_3	K_3	J_2	K_2	J_1	K_1
0	0	0	0	0	1	0	×	0	×	1	×
0	0	1	0	1	0	0	×	1	×	×	1
0	1	0	0	1	1	0	×	×	0	1	×
0	1	1	1	0	0	1	×	×	1	×	1
1	0	0	1	0	1	×	0	0	×	1	×
1	0	1	1	1	0	×	0	1	×	×	1
1	1	0	1	1	1	×	0	×	0	1	×
1	1	1	0	0	0	×	1	×	1	×	1

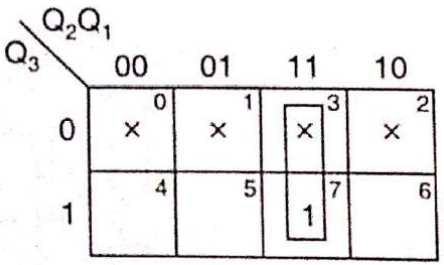
(b) Excitation table

J-K excitation table

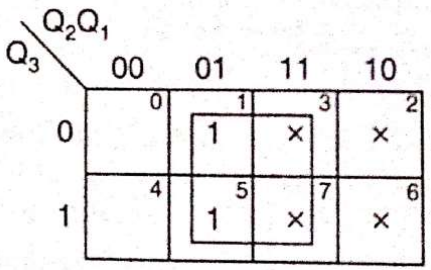
PS	NS	Required inputs	
Q_n	Q_{n+1}	J	K
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0



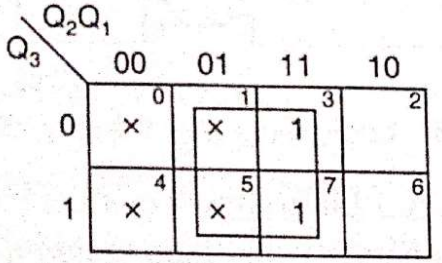
$J_3 = Q_2Q_1$



$K_3 = Q_2Q_1$

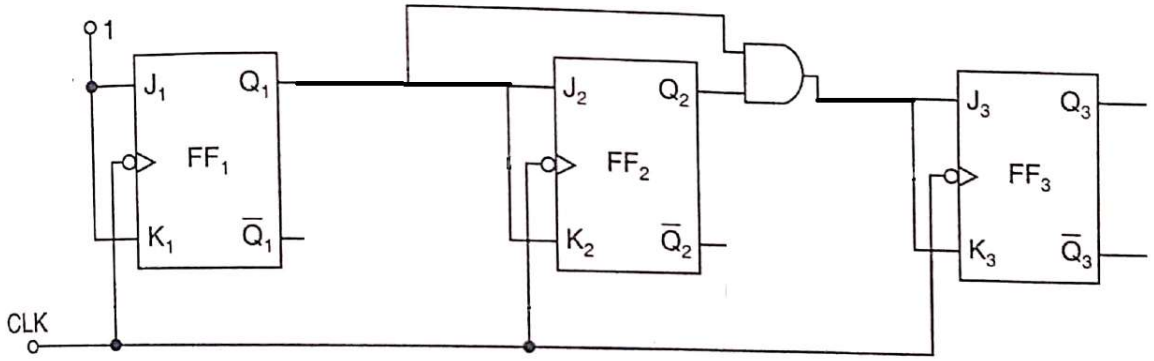


$J_2 = Q_1$



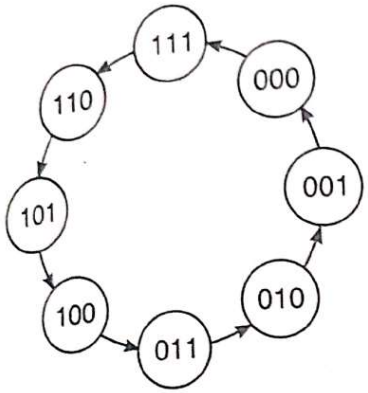
$K_2 = Q_1$

- Logic Diagram of 3-bit Synchronous Counter



Logic diagram of the synchronous 3-bit up-down counter using J-K FFs.

Design of 3-bit Synchronous down counter

- 

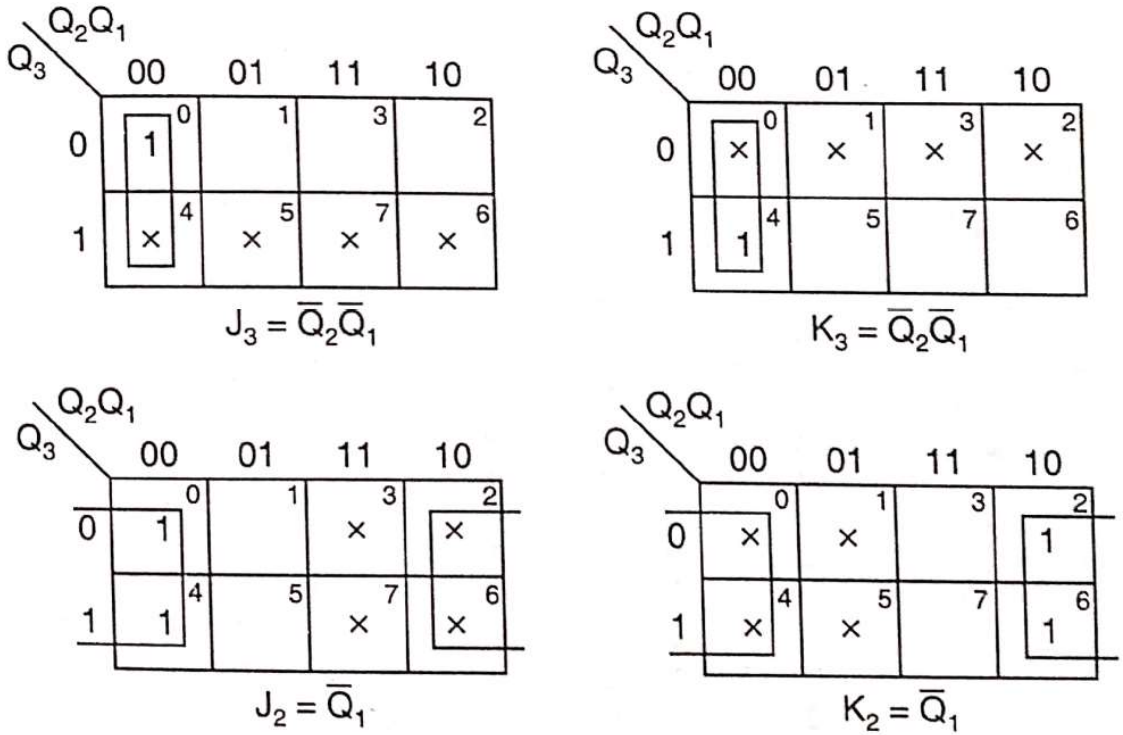
(a) State diagram

PS			NS			Required excitations					
Q_3	Q_2	Q_1	Q_3	Q_2	Q_1	J_3	K_3	J_2	K_2	J_1	K_1
0	0	0	1	1	1	1	×	1	×	1	×
1	1	1	1	1	0	×	0	×	0	×	1
1	1	0	1	0	1	×	0	×	1	1	×
1	0	1	1	0	0	×	0	0	×	×	1
1	0	0	0	1	1	×	1	1	×	1	×
0	1	1	0	1	0	0	×	×	0	×	1
0	1	0	0	0	1	0	×	×	1	1	×
0	0	1	0	0	0	0	×	0	×	×	1

(b) Excitation table

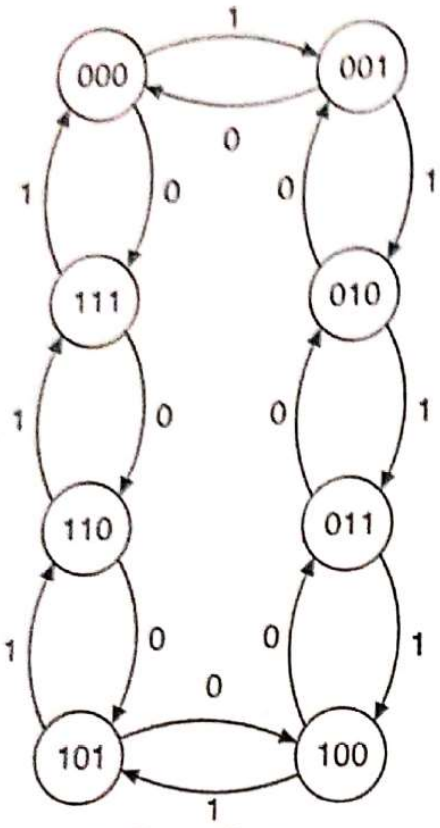
J-K excitation table

PS	NS	Required inputs	
Q_n	Q_{n+1}	J	K
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0



K-maps for a three-bit down counter using J-K FFs.

Design of 3-bit Synchronous Up/Down Counter



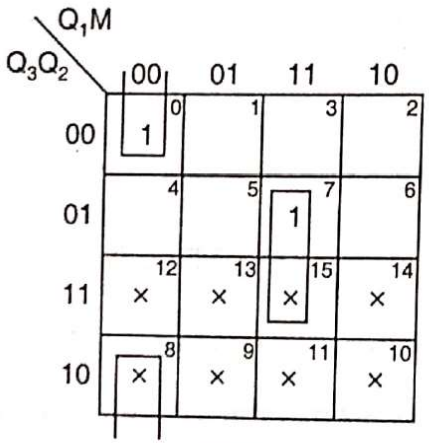
(a) State diagram

PS			Mode	NS			Required excitations					
Q_3	Q_2	Q_1	M	Q_3	Q_2	Q_1	J_3	K_3	J_2	K_2	J_1	K_1
0	0	0	0	1	1	1	1	×	1	×	1	×
0	0	0	1	0	0	1	0	×	0	×	1	×
0	0	1	0	0	0	0	0	×	0	×	×	1
0	0	1	1	0	1	0	0	×	1	×	×	1
0	1	0	0	0	0	1	0	×	×	1	1	×
0	1	0	1	0	1	1	0	×	×	0	1	×
0	1	1	0	0	1	0	0	×	×	0	×	1
0	1	1	1	1	0	0	1	×	×	1	×	1
1	0	0	0	0	1	1	×	1	1	×	1	×
1	0	0	1	1	0	1	×	0	0	×	1	×
1	0	1	0	1	0	0	×	0	0	×	×	1
1	0	1	1	1	1	0	×	0	1	×	×	1
1	1	0	0	1	0	1	×	0	×	1	1	×
1	1	0	1	1	1	1	×	0	×	0	1	×
1	1	1	0	1	1	0	×	0	×	0	×	1
1	1	1	1	0	0	0	×	1	×	1	×	1

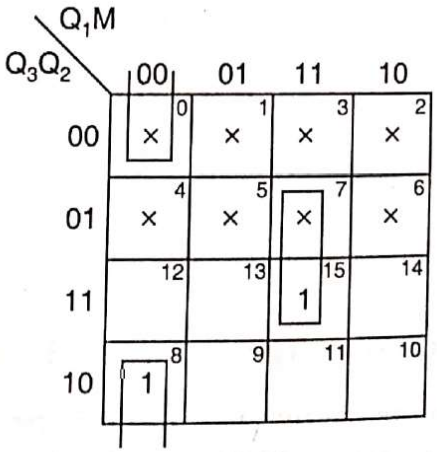
(b) Excitation table

J-K excitation table

PS	NS	Required inputs	
Q_n	Q_{n+1}	J	K
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0

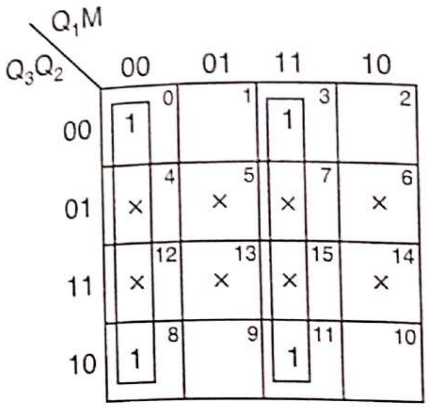


$J_3 = \overline{Q}_2\overline{Q}_1\overline{M} + Q_2Q_1M$

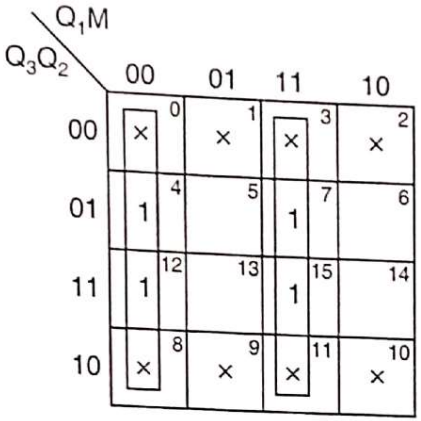


$K_3 = \overline{Q}_2\overline{Q}_1\overline{M} + Q_2Q_1M$

K-maps for excitations of synchronous 3-bit up-down counter

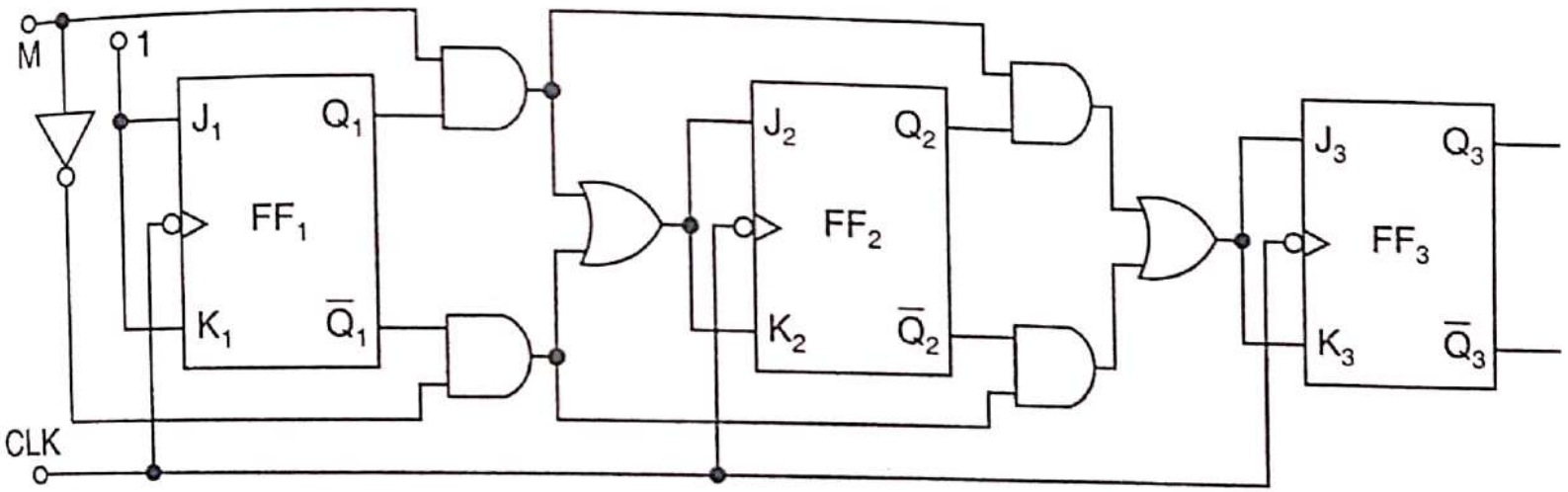


$J_2 = \overline{Q}_1\overline{M} + Q_1M$



$K_2 = \overline{Q}_1\overline{M} + Q_1M$

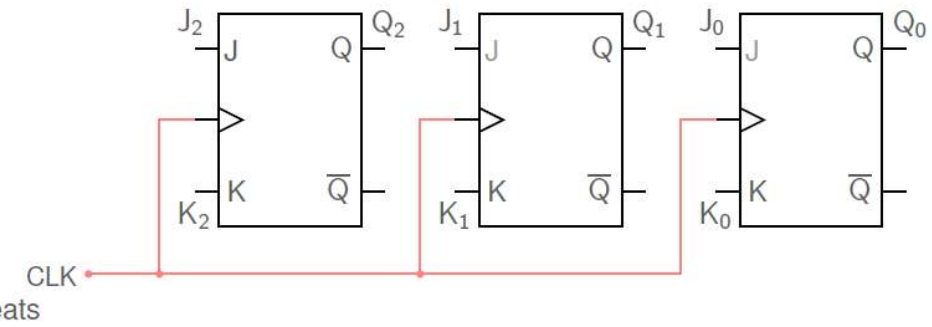
K-maps for excitations of synchronous 3-bit up-down counter.



Logic diagram of the synchronous 3-bit up-down counter using J-K FFs.

state	Q_2	Q_1	Q_0
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
1	0	0	0

repeats



CLK	Q_n	Q_{n+1}	J	K
↑	0	0	0	X
↑	0	1	1	X
↑	1	0	X	1
↑	1	1	X	0

Design a synchronous mod-5 counter with the given state transition table.

Outline of method:

* State 1 $\rightarrow$ State 2 means
  Q_2: $0 \rightarrow 0$,
  Q_1: $0 \rightarrow 0$,
  Q_0: $0 \rightarrow 1$.
* Refer to the right table. For Q_2: $0 \rightarrow 0$, we must have $J_2 = 0$, $K_2 = X$, and so on.
* When we cover all transitions in the left table, we have the truth tables for J_0, K_0, J_1, K_1, J_2, K_2 in terms of Q_0, Q_1, Q_2.
* The last step is to come up with suitable functions for J_0, K_0, J_1, K_1, J_2, K_2 in terms of Q_0, Q_1, Q_2. This can be done with K-maps. (If the number of flip-flops is more than 4, other techniques can be employed.)

state	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
1	0	0	0	0	X	0	X	1	X
2	0	0	1	0	X	1	X	X	1
3	0	1	0	0	X	X	0	1	X
4	0	1	1	1	X	X	1	X	1
5	1	0	0	X	1	0	X	0	X
1	0	0	0						

CLK	Q_n	Q_{n+1}	J	K
↑	0	0	0	X
↑	0	1	1	X
↑	1	0	X	1
↑	1	1	X	0

state	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
1	0	0	0	0	X	0	X	1	X
2	0	0	1	0	X	1	X	X	1
3	0	1	0	0	X	X	0	1	X
4	0	1	1	1	X	X	1	X	1
5	1	0	0	X	1	0	X	0	X
1	0	0	0						

J_2

Q_0 \ Q_2Q_1	00	01	11	10
0	0	0	X	X
1	0	1	X	X

K_2

Q_0 \ Q_2Q_1	00	01	11	10
0	X	X	X	1
1	X	X	X	X

J_1

Q_0 \ Q_2Q_1	00	01	11	10
0	0	X	X	0
1	1	X	X	X

K_1

Q_0 \ Q_2Q_1	00	01	11	10
0	X	0	X	X
1	X	1	X	X

J_0

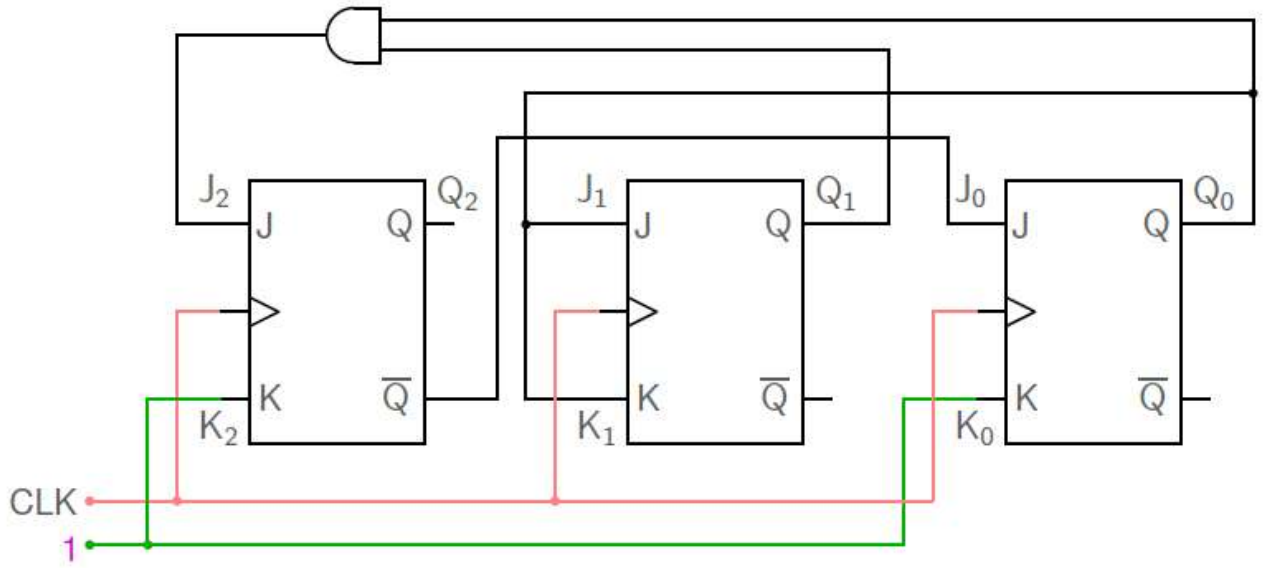
Q_0 \ Q_2Q_1	00	01	11	10
0	1	1	X	0
1	X	X	X	X

K_0

Q_0 \ Q_2Q_1	00	01	11	10
0	X	X	X	X
1	1	1	X	X

* $J_2 = Q_1 Q_0,$
  $K_2 = 1,$
  $J_1 = Q_0,$
  $K_1 = Q_0,$
  $J_0 = \overline{Q_2},$
  $K_0 = 1.$

* We treat the unused states ($Q_2 Q_1 Q_0 = 101$, 110, 111) as (additional) don't care conditions. Since these are different from the don't care conditions arising from the state transition table, we mark them with a different colour.
* We will assume that a suitable initialization facility is provided to ensure that the counter starts up in one of the five allowed states (say, $Q_2 Q_1 Q_0 = 000$).
* From the K-maps, $J_2 = Q_1 Q_0$, $K_2 = 1$, $J_1 = Q_0$, $K_1 = Q_0$, $J_0 = \overline{Q_2}$, $K_0 = 1$.



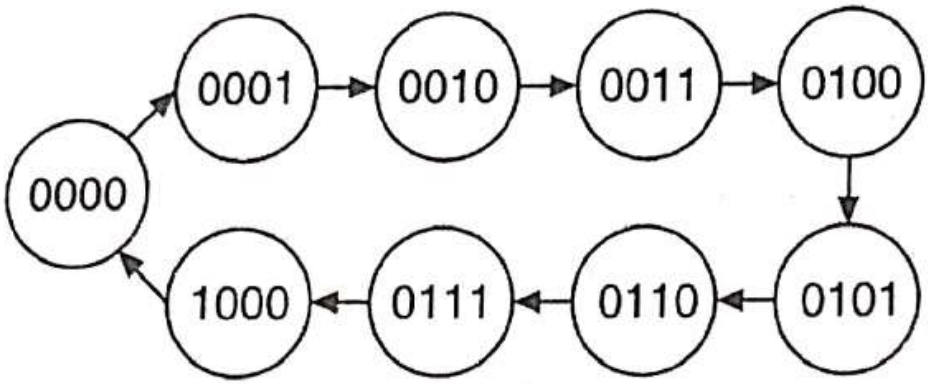
* $J_2 = Q_1 Q_0$,
  $K_2 = 1$,
  $J_1 = Q_0$,
  $K_1 = Q_0$,
  $J_0 = \overline{Q_2}$,
  $K_0 = 1$.

Design a Modulo -9 Synchronous counter

- Step 1: The Counter has 9 states. Number of Flip-Flops: 4.

  Remaining 7 states are invalid and are treated as don't cares.

- Step 2: State Diagram



(a) State diagram

- Step 3: Type of Flip-Flops and Excitation Table

  T Flip-Flops are used and writing of excitation table for the counter using the Excitation table of T Flip-Flops.

PS				NS				Required Excitations			
Q_4	Q_3	Q_2	Q_1	Q_4	Q_3	Q_2	Q_1	T_4	T_3	T_2	T_1
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	0	0	1	0	0	0

(b) Excitation table

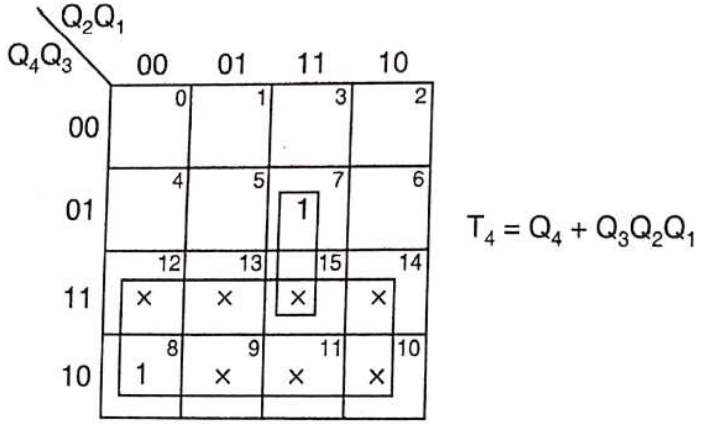
T truth table

T	Q_{n+1}
0	Q_n
1	$\bar{Q}_n$

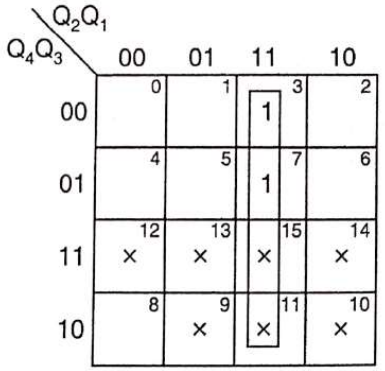
T excitation table

PS	NS	Required input
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

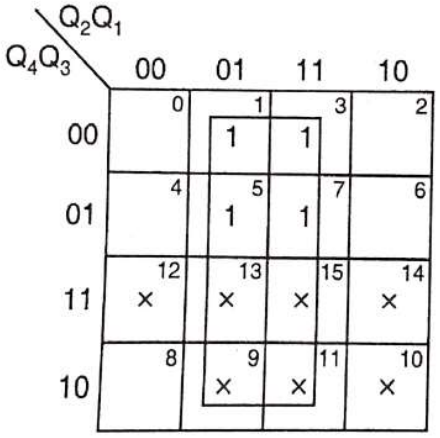
- Step 4 : Minimal Expressions:



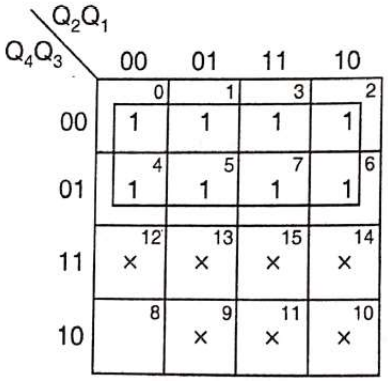
$T_4 = Q_4 + Q_3Q_2Q_1$



$T_3 = Q_2Q_1$



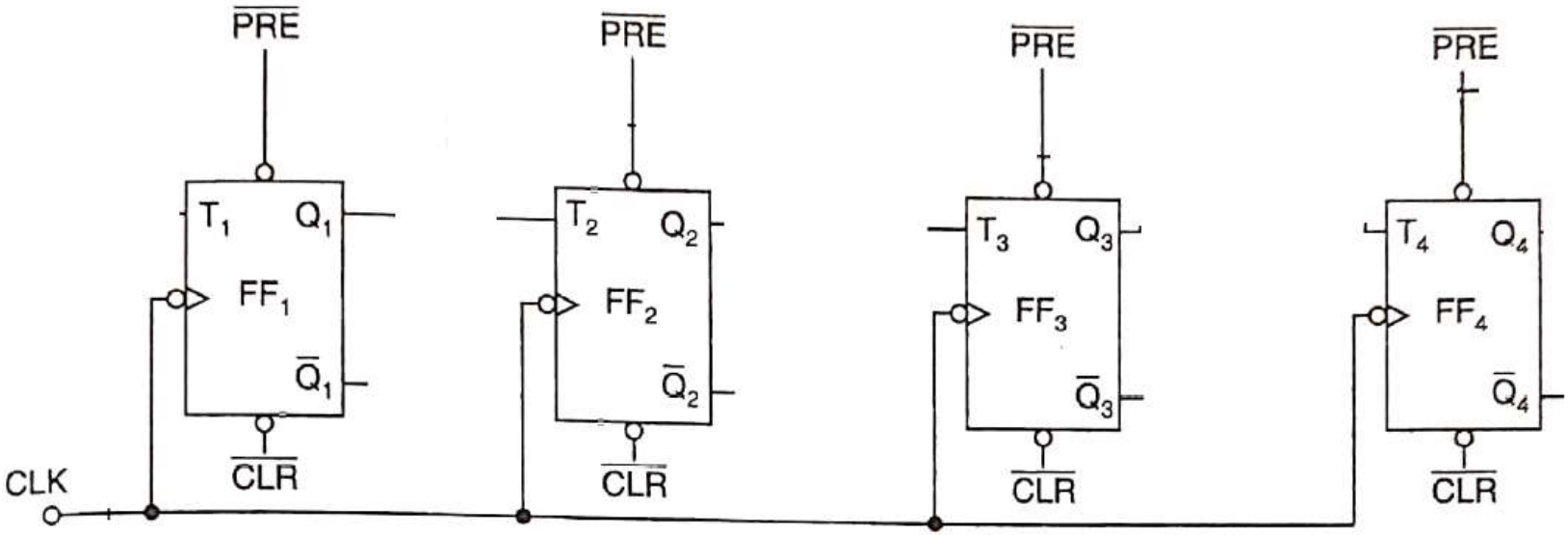
$T_2 = Q_1$



$T_1 = \overline{Q}_4$

K-maps for excitations of synchronous mod-9 counter using T flip-flops.

- Step 5. Logic Diagram:



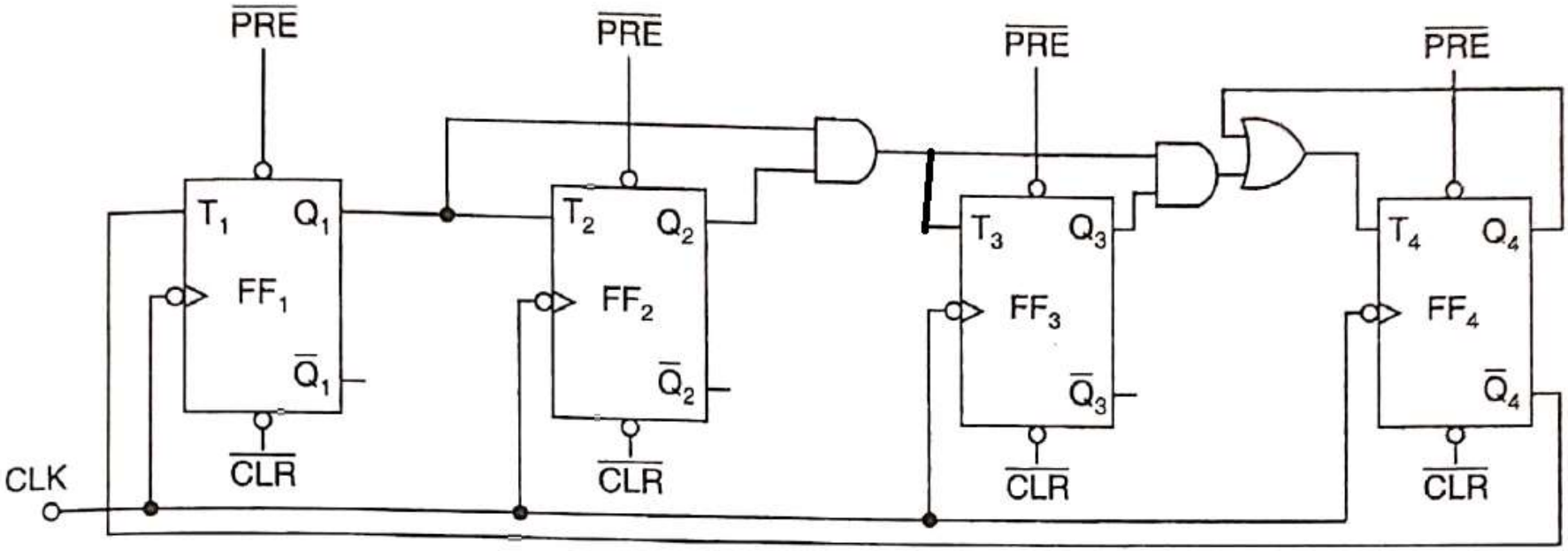
Logic diagram of the synchronous mod-9 counter using T flip-flops.

$T_4 = Q_4 + Q_3Q_2Q_1$

$T_3 = Q_2Q_1$

$T_2 = Q_1$

$T_1 = \overline{Q}_4$



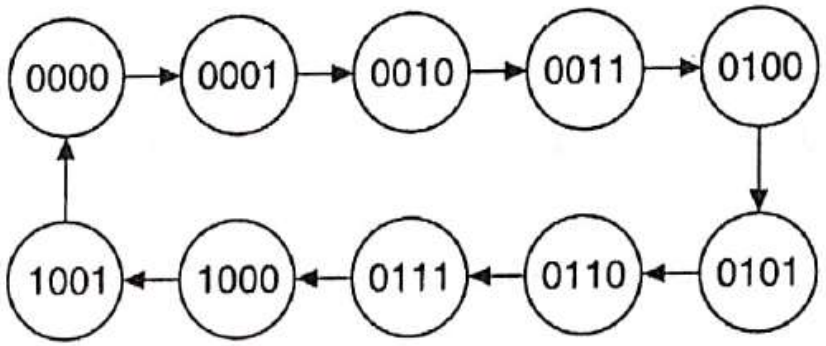
Logic diagram of the synchronous mod-9 counter using T flip-flops.

Design of a Synchronous BCD Counter using J-K Flip-Flop

- Step 1: The Counter has 10 states. Number of Flip-Flops: 4.

  Remaining 6 states are invalid and are treated as don't cares.

- Step 2: State Diagram



(a) State diagram

- Step 3: Type of Flip-Flops and Excitation Table

  J-K Flip-Flops are used and writing of excitation table for the counter using the Excitation table of J-K Flip-Flops.

PS				NS				Required excitations							
Q_4	Q_3	Q_2	Q_1	Q_4	Q_3	Q_2	Q_1	J_4	K_4	J_3	K_3	J_2	K_2	J_1	K_1
0	0	0	0	0	0	0	1	0	×	0	×	0	×	1	×
0	0	0	1	0	0	1	0	0	×	0	×	1	×	×	1
0	0	1	0	0	0	1	1	0	×	0	×	×	0	1	×
0	0	1	1	0	1	0	0	0	×	1	×	×	1	×	1
0	1	0	0	0	1	0	1	0	×	×	0	0	×	1	×
0	1	0	1	0	1	1	0	0	×	×	0	1	×	×	1
0	1	1	0	0	1	1	1	0	×	×	0	×	0	1	×
0	1	1	1	1	0	0	0	1	×	×	1	×	1	×	1
1	0	0	0	1	0	0	1	×	0	0	×	0	×	1	×
1	0	0	1	0	0	0	0	×	1	0	×	0	×	×	1

(b) Excitation table

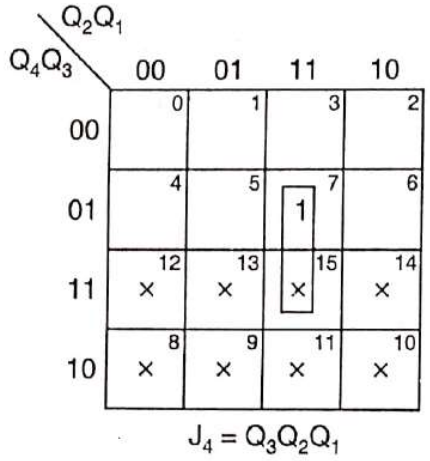
J-K truth table

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q}_n$

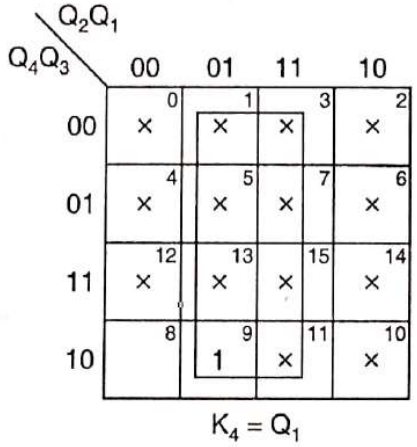
J-K excitation table

PS	NS	Required inputs	
Q_n	Q_{n+1}	J	K
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0

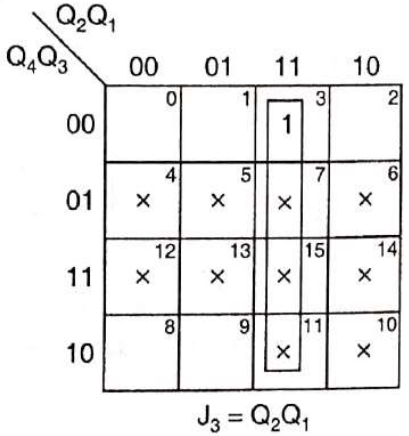
- Step 4 : Minimal Expressions:



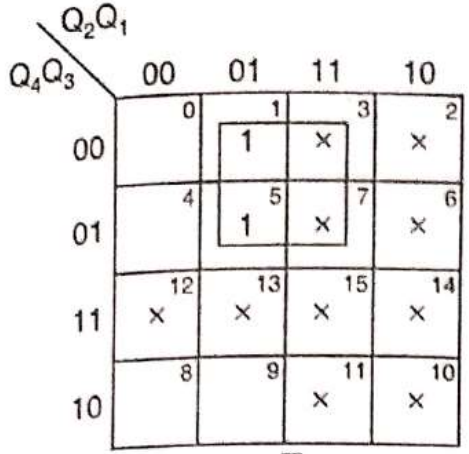
$J_4 = Q_3Q_2Q_1$



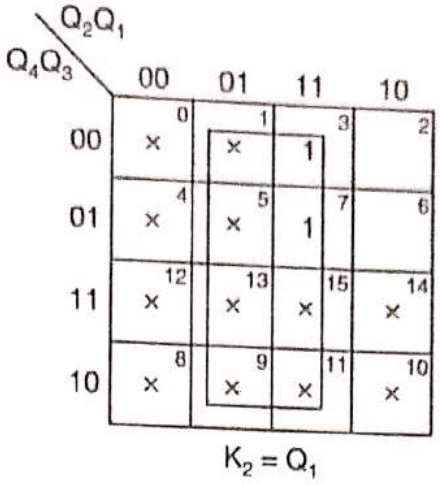
$K_4 = Q_1$



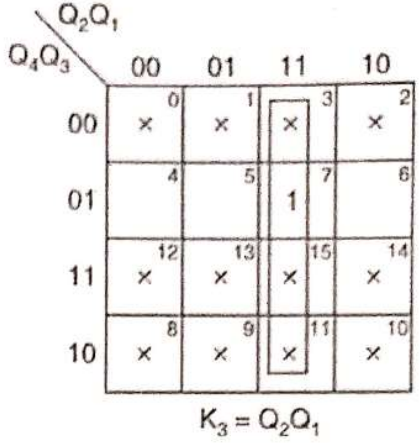
$J_3 = Q_2Q_1$



$J_2 = \overline{Q}_4Q_1$

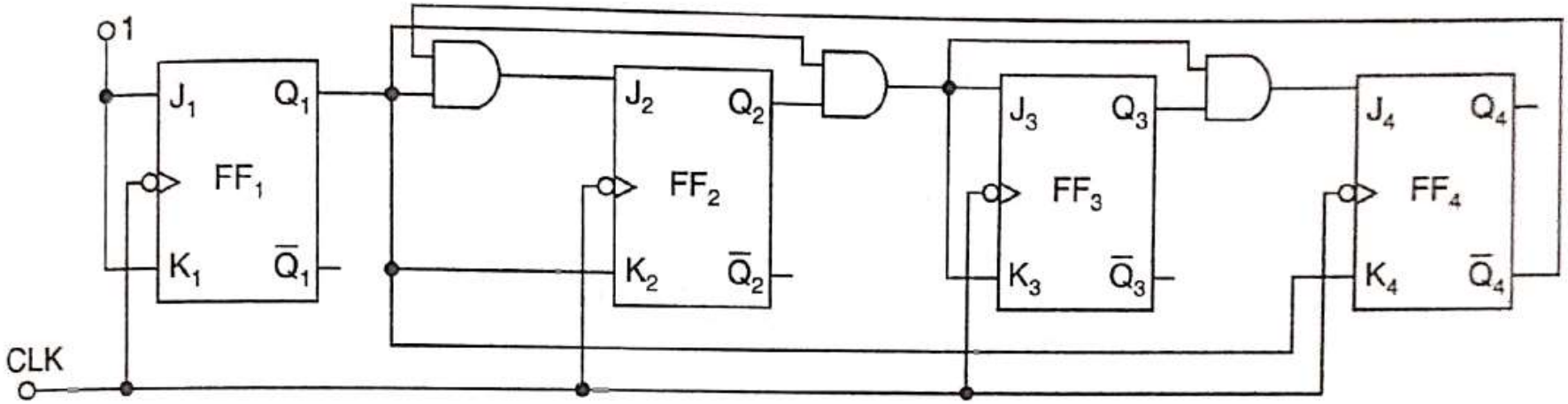


$K_2 = Q_1$



$K_3 = Q_2Q_1$

- Step 5. Logic Diagram:

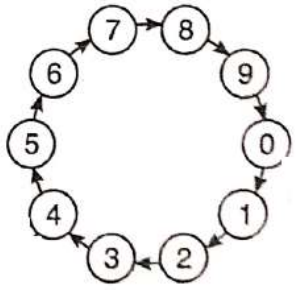


Logic diagram of synchronous BCD counter using J-K flip-flops.

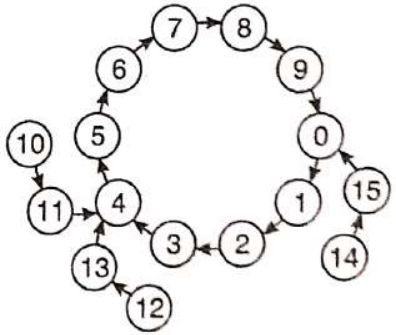
- To check for Lock-out condition

PS				Present inputs								NS			
Q_4	Q_3	Q_2	Q_1	J_4	K_4	J_3	K_3	J_2	K_2	J_1	K_1	Q_4	Q_3	Q_2	Q_1
1	0	1	0	0	0	0	0	0	0	1	1	1	0	1	1
1	0	1	1	0	1	1	1	0	1	1	1	0	1	0	0
1	1	0	0	0	0	0	0	0	0	1	1	1	1	0	1
1	1	0	1	0	1	0	0	0	1	1	1	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	1	1	1	1	1
1	1	1	1	1	1	1	1	0	1	1	1	0	0	0	0

(a) Table to check for lock-out



(b) State diagram



(b) State diagram

$J_4 = Q_3Q_2Q_1$

$K_4 = Q_1$

$J_3 = Q_2Q_1$

$K_3 = Q_2Q_1$

$J_2 = \overline{Q}_4Q_1$

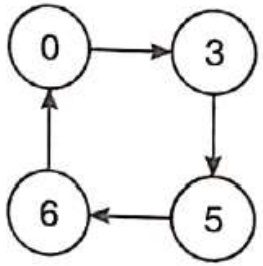
$K_2 = Q_1$

J-K truth table

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q}_n$

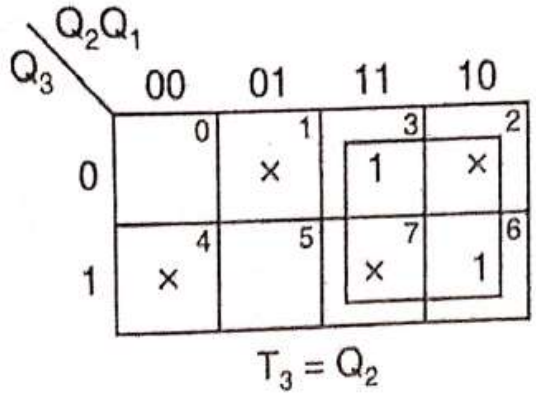
Design a Synchronous Counter that goes through 0,3,5,6,0…. Using T-Flip-Flop. Is the counter self starting?

- No of Flip-Flops: 3
- State Diagram and Excitation Table

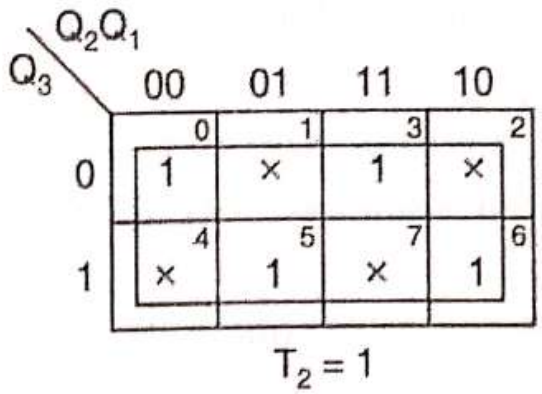


PS			NS			Required excitations		
Q_3	Q_2	Q_1	Q_3	Q_2	Q_1	T_3	T_2	T_1
0	0	0	0	1	1	0	1	1
0	1	1	1	0	1	1	1	0
1	0	1	1	1	0	0	1	1
1	1	0	0	0	0	1	1	0

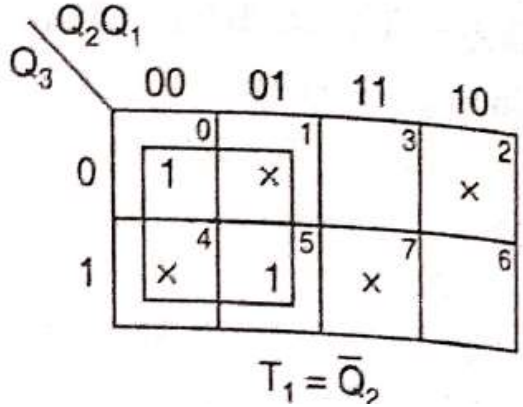
(b) Excitation table



$T_3 = Q_2$



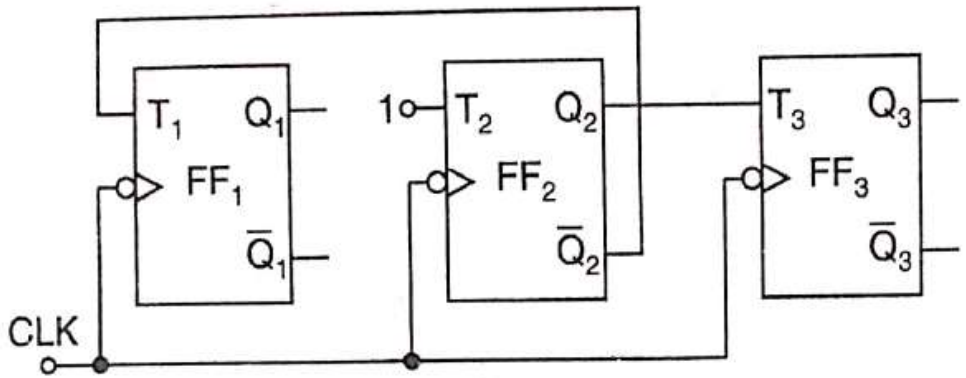
$T_2 = 1$



$T_1 = \bar{Q}_2$

K-maps for excitations of T type, 0, 3, 5, 6, 0... counter.

- Logic Diagram

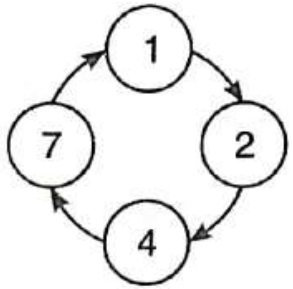


(a) Logic diagram

- Check for lock-out

PS			Present inputs			NS		
Q_3	Q_2	Q_1	T_3	T_2	T_1	Q_3	Q_2	Q_1
0	0	1	0	1	1	0	1	0
0	1	0	1	1	0	1	0	0
1	0	0	0	1	1	1	1	1
1	1	1	1	1	0	0	0	1

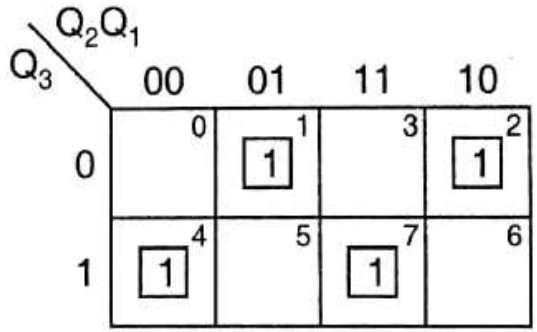
(b) Table to check for lock-out



- Elimination of lock-out
- Method 1:

PS			Reset
Q_3	Q_2	Q_1	R
0	0	1	1
0	1	0	1
1	0	0	1
1	1	1	1

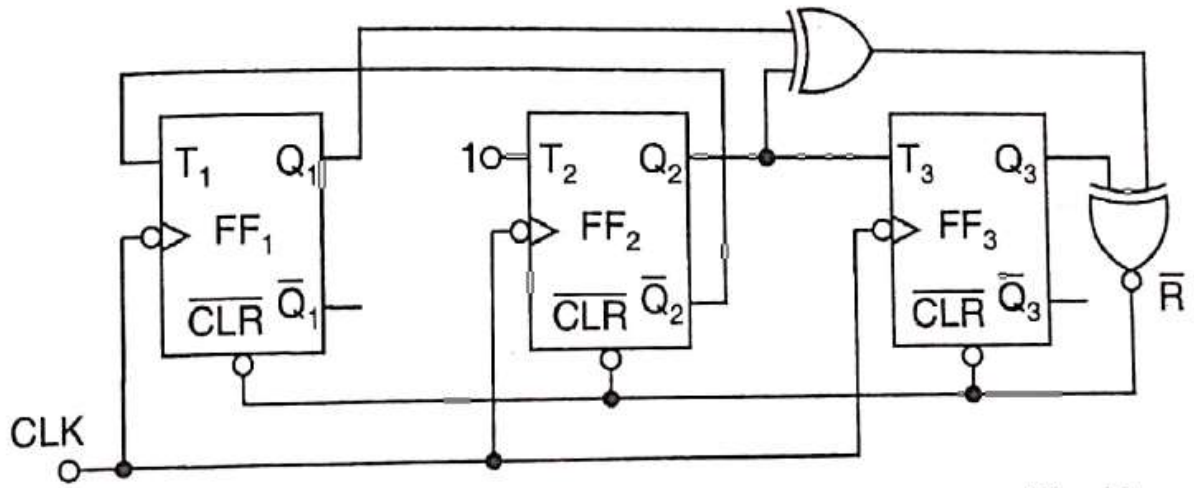
Table for R



K-map

$$R = \overline{Q}_3\overline{Q}_2Q_1 + \overline{Q}_3Q_2\overline{Q}_1 + Q_3\overline{Q}_2\overline{Q}_1 + Q_3Q_2Q_1$$
$$= \overline{Q}_3(Q_2 \oplus Q_1) + Q_3\overline{(Q_2 \oplus Q_1)}$$
$$= Q_3 \oplus Q_2 \oplus Q_1$$

minimal expression for R.

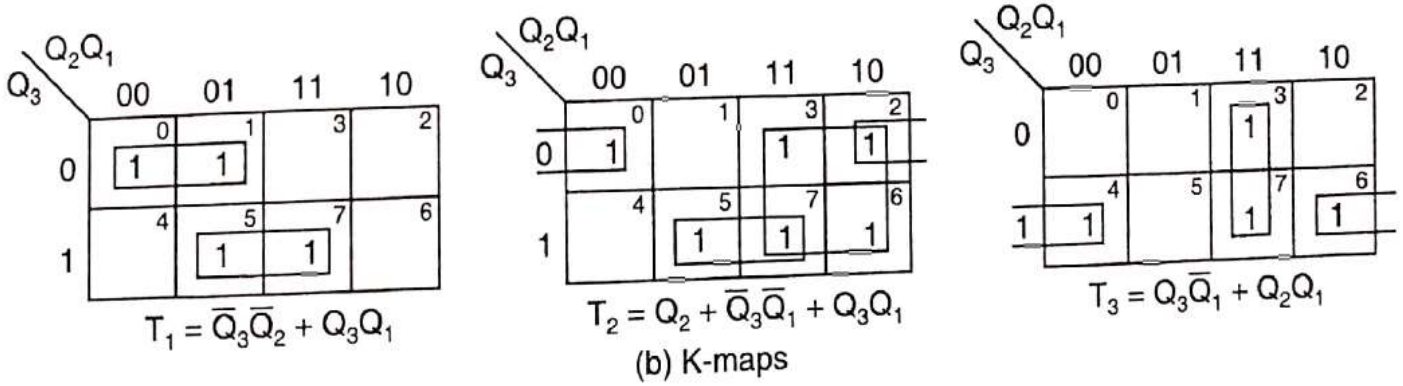


Logic diagram of the type T 0, 3, 5, 6, 0... counter modified to eliminate lock-out.

- Method 2:

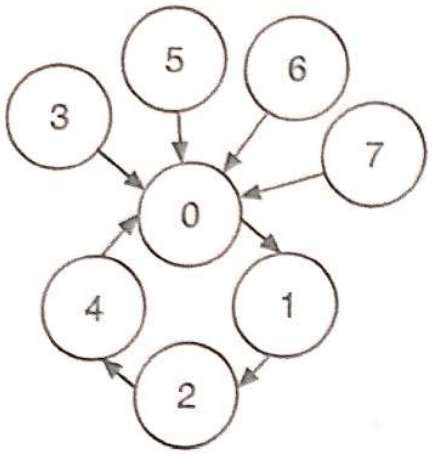
PS			NS			Required excitations		
Q_3	Q_2	Q_1	Q_3	Q_2	Q_1	T_3	T_2	T_1
0	0	0	0	1	1	0	1	1
0	0	1	0	0	0	0	0	1
0	1	0	0	0	0	0	1	0
0	1	1	1	0	1	1	1	0
1	0	0	0	0	0	1	0	0
1	0	1	1	1	0	0	1	1
1	1	0	0	0	0	1	1	0
1	1	1	0	0	0	1	1	1

(a) Excitation table



(b) K-maps

Design a type D counter that goes through 0,1,2,4,0…. states



(a) State diagram

PS			NS			Required excitations		
Q_3	Q_2	Q_1	Q_3	Q_2	Q_1	D_3	D_2	D_1
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	0
0	1	0	1	0	0	1	0	0
0	1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0

(b) Excitation table

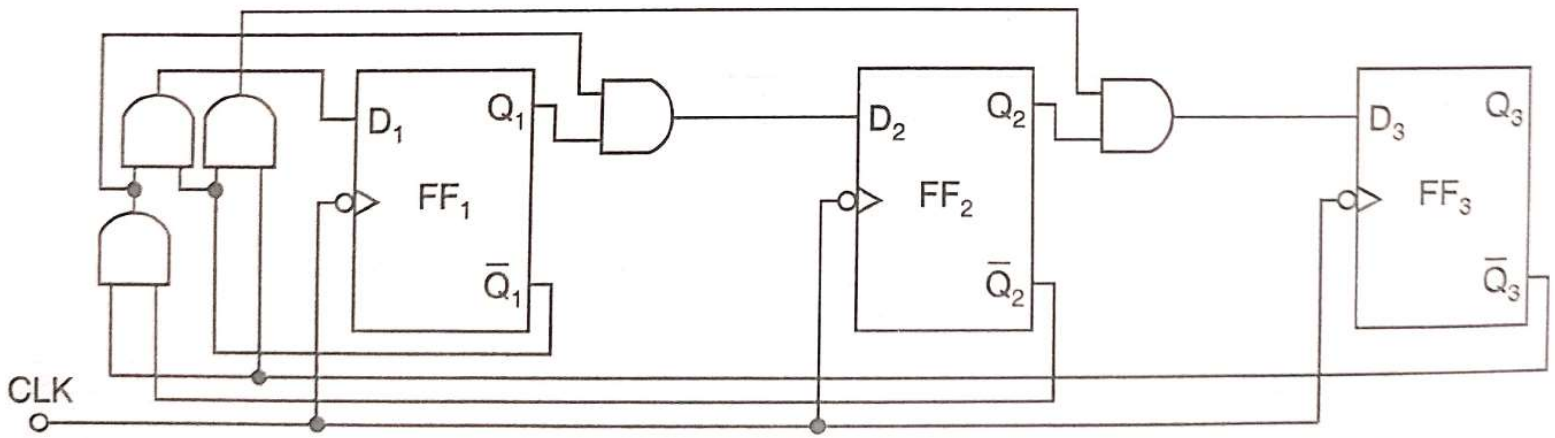
0, 1, 2, 4, 0, ... counter.

$$D_1 = \bar{Q}_3\bar{Q}_2\bar{Q}_1$$

$$D_2 = \bar{Q}_3\bar{Q}_2Q_1$$

$$D_3 = \bar{Q}_3Q_2\bar{Q}_1$$

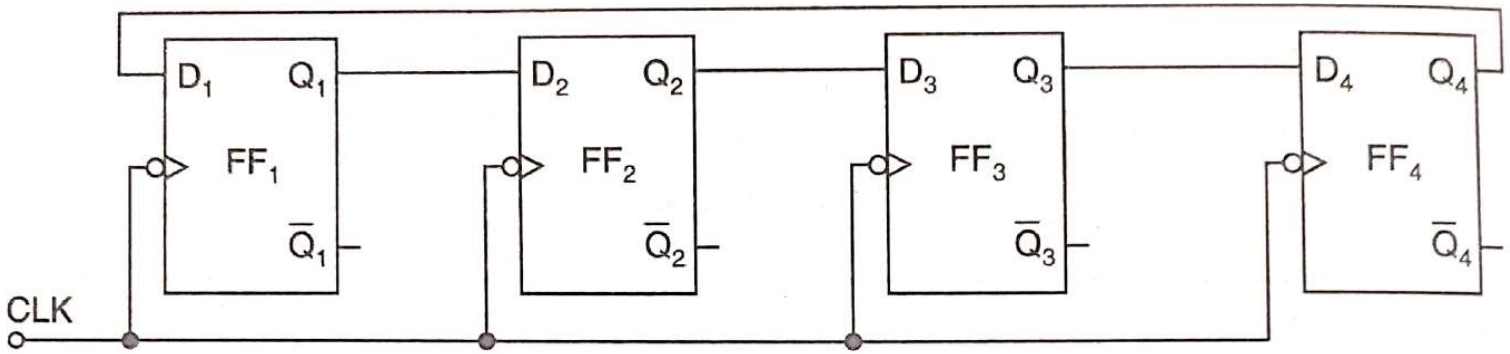
$$D_3 = \bar{Q}_3 Q_2 \bar{Q}_1 \qquad D_2 = \bar{Q}_3 \bar{Q}_2 Q_1 \qquad D_1 = \bar{Q}_3 \bar{Q}_2 \bar{Q}_1$$



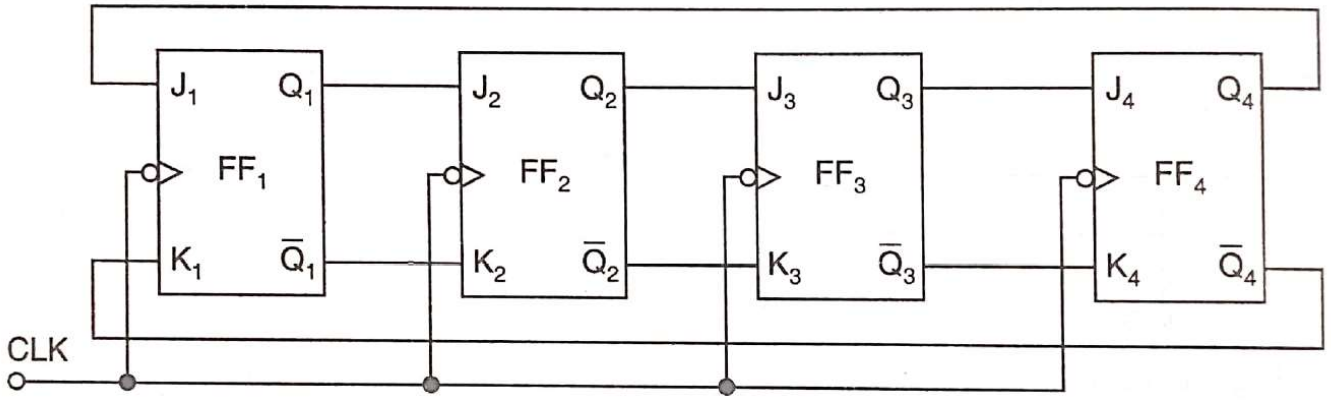
Logic diagram of type D counter that goes through states 0, 1, 2, 4, 0, ...

Shift Register Counter

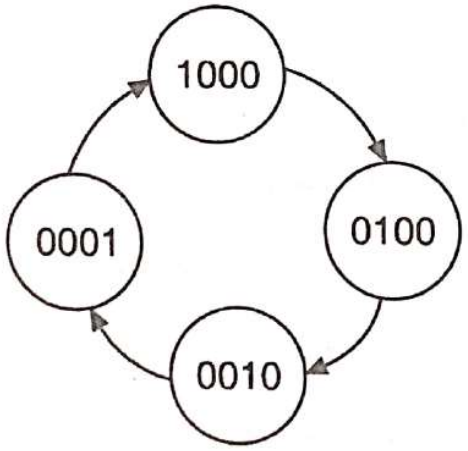
- Ring Counter



Logic diagram of a 4-bit ring counter using D flip-flops.



Logic diagram of a 4-bit ring counter using J-K flip-flops.

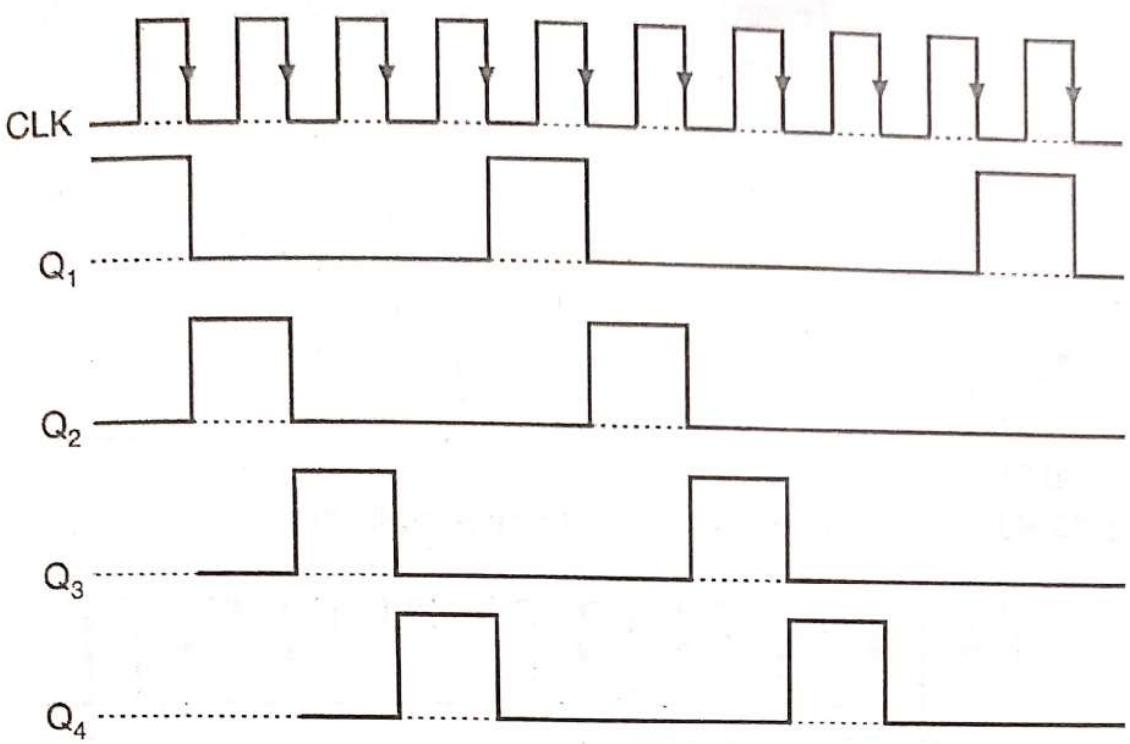


(a) State diagram

Q_1	Q_2	Q_3	Q_4	After clock pulse
1	0	0	0	0
0	1	0	0	1
0	0	1	0	2
0	0	0	1	3
1	0	0	0	4
0	1	0	0	5
0	0	1	0	6
0	0	0	1	7

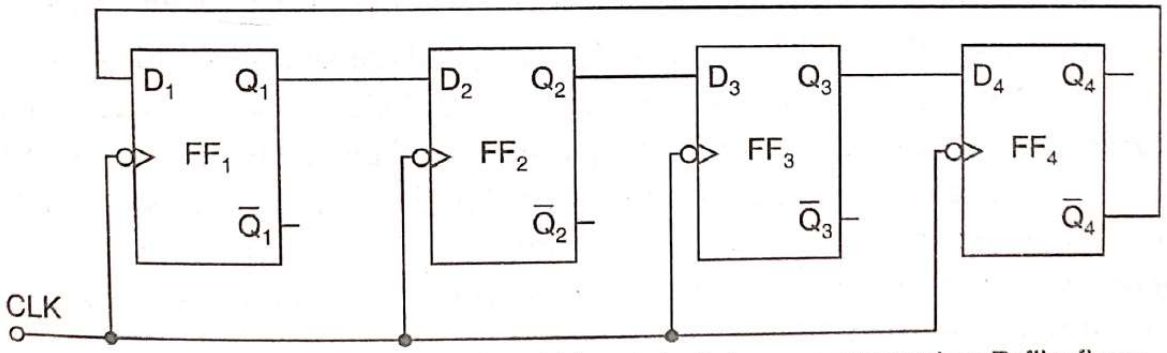
(b) Sequence table

State diagram and sequence table of a 4-bit ring counter.

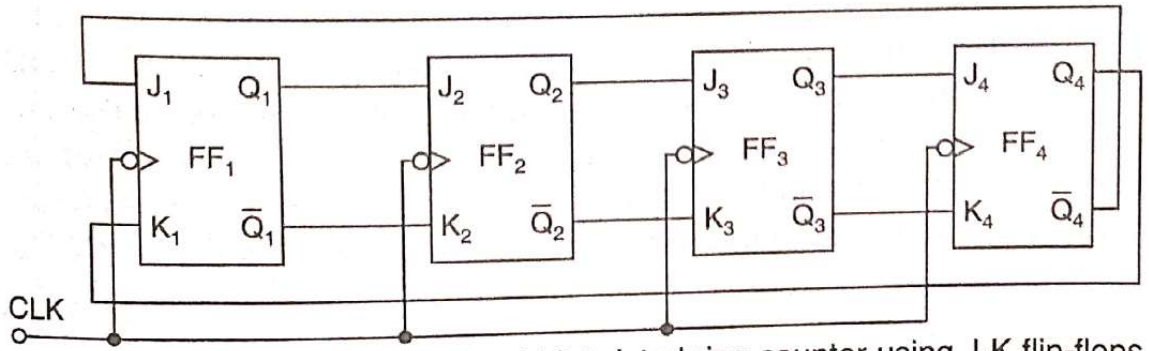


Timing diagram of a 4-bit ring counter.

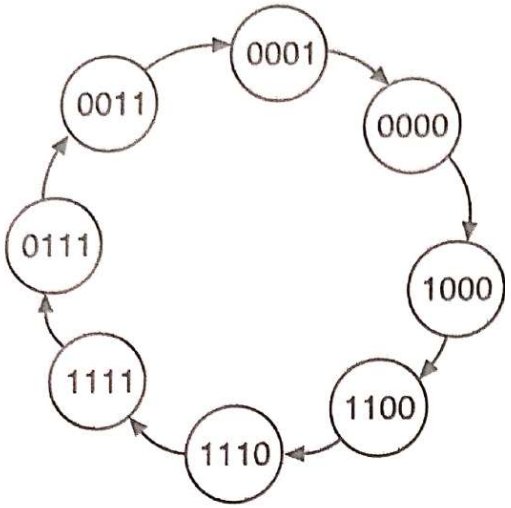
Twisted Ring/ Johnson Counter



Logic diagram of a 4-bit twisted ring counter using D flip-flops.



Logic diagram of a 4-bit twisted ring counter using J-K flip-flops.

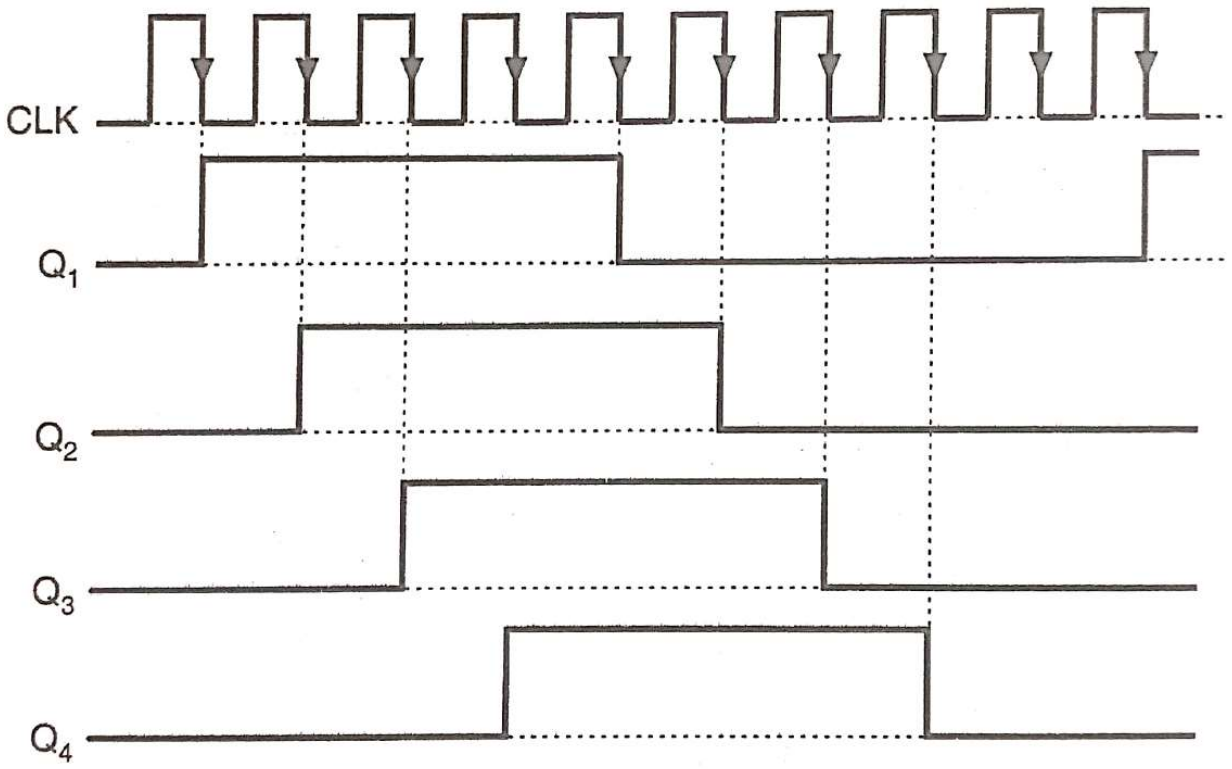


(a) State diagram

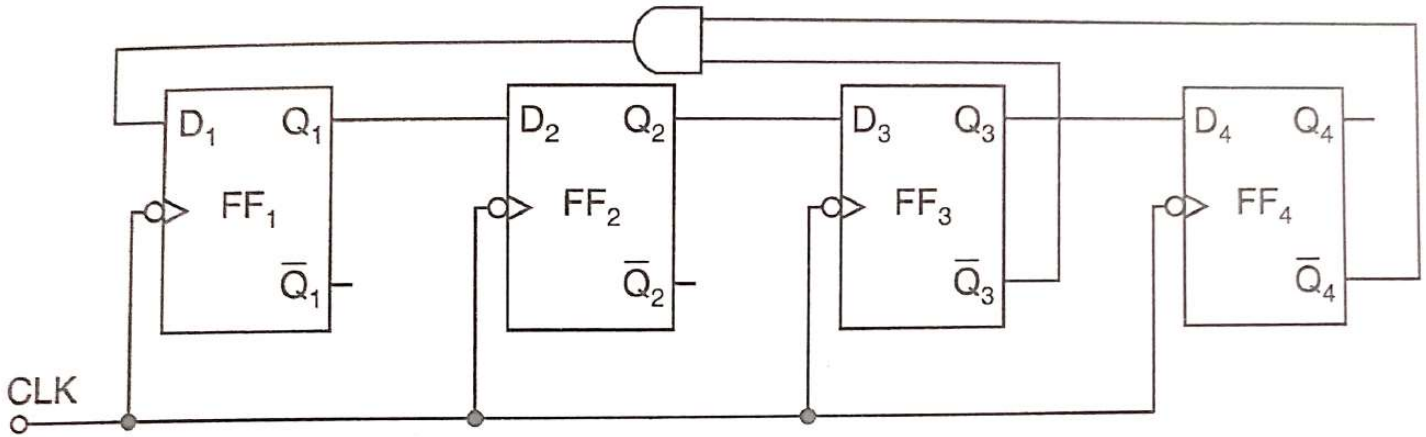
Q_1	Q_2	Q_3	Q_4	After clock pulse
0	0	0	0	0
1	0	0	0	1
1	1	0	0	2
1	1	1	0	3
1	1	1	1	4
0	1	1	1	5
0	0	1	1	6
0	0	0	1	7
0	0	0	0	8
1	0	0	0	9

(b) Sequence table

State diagram and sequence table of a twisted ring counter.



Timing diagram of a 4-bit twisted ring counter.



Logic diagram of a 4-bit Johnson counter designed to prevent lock-out.

Recall: Sensitivity List of always block

- Syntax:

```
always @(sensitivity list) begin
  procedural statements
end
```

- Sensitivity list is a list of signals: **`@(signal1, signal2, …)`**
- The sensitivity list triggers the execution of the **`always`** block

  When there is a ***change of value in any listed signal***

  Otherwise, the **`always`** block does nothing until another change occurs on a signal in the sensitivity list

Guidelines for Sensitivity List

- For combinational logic, the sensitivity list must include **ALL** the signals **that are read** inside the `always` block

  Combinational logic can also use: `@(*)` or `@*`

- For sequential logic, the sensitivity list may not include all the signals that are read inside the `always` block

- For **edge-triggered** sequential logic use:

```
always @(posedge signal1, negedge signal2, …)
```

- The ***positive edge*** or ***negative edge*** of each signal can be specified in the sensitivity list

Modeling a D Latch with Enable

```
// Modeling a D Latch with Enable and output Q
// Output Q must be of type reg
// Notice that the if statement does NOT have else
// If Enable is 0, then value of Q does not change
// The D_latch stores the old value of Q

module D_latch (input D, Enable, output reg Q);
  always @(D, Enable)
    if (Enable) Q <= D;  // Non-blocking assignment
    // No else means a latch (Q does not change)
endmodule
```

Modeling a D-type Flip-Flop

```
// Modeling a D Flip-Flop with outputs Q and Qbar
module D_FF (input D, Clk, output reg Q, Qbar);
  // Q and Qbar change at the positive edge of Clk
  // Notice that always is NOT sensitive to D input
  always @(posedge Clk)
  begin
    Q <= D;          // Non-blocking assignment
    Qbar <= ~D;      // Non-blocking assignment
  end
endmodule
```

Negative-Edge Triggered D-type Flip-Flop

```
// Modeling a Negative-Edge Triggered D Flip-Flop
// The only difference is the negative edge of Clk
module D_FF2 (input D, Clk, output reg Q, Qbar);
  // Q and Qbar change at the negative edge of Clk
  always @(negedge Clk)
  begin
    Q <= D;          // Non-blocking assignment
    Qbar <= ~D;      // Non-blocking assignment
  end
endmodule
```

D-type Flip-Flop with Synchronous Reset

```
// Modeling a D Flip-Flop with Synchronous Reset input
module D_FF3 (input D, Clk, Reset, output reg Q, Qbar);
  // always block is NOT sensitive to Reset or D
  // Updates happen only at positive edge of Clk
  // Reset is Synchronized with Clk
  always @(posedge Clk)
    if (Reset)
      {Q, Qbar} <= 2'b01;   // Non-blocking assignment
    else
      {Q, Qbar} <= {D, ~D}; // Non-blocking assignment
endmodule
```

D-type Flip-Flop with Asynchronous Reset

```
// Modeling a D Flip-Flop with Asynchronous Reset input
module D_FF4 (input D, Clk, Reset, output reg Q, Qbar);
  // Q and Qbar change at the positive edge of Clk
  // Or, at the positive edge of Reset
  // Reset is NOT synchronized with Clk
  always @(posedge Clk, posedge Reset)
    if (Reset)
      {Q, Qbar} <= 2'b01;   // Non-blocking assignment
    else
      {Q, Qbar} <= {D, ~D}; // Non-blocking assignment
endmodule
```

Procedural Assignment

- Procedural assignment is used inside a **procedural block** only
- Two types of procedural assignments:
- **Blocking assignment:**

```
variable = expression;    // = operator
```

  Variable is updated **before** executing next statement

  Similar to an assignment statement in programming languages

- **Non-Blocking assignment:**

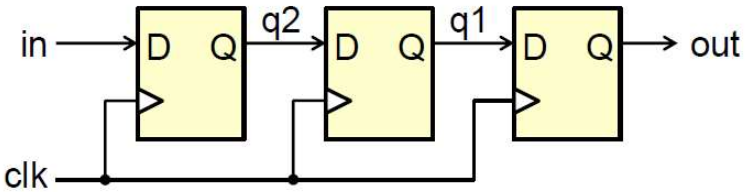
```
variable <= expression;   // <= operator
```

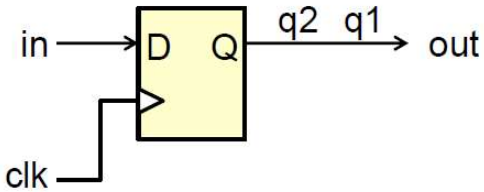
  Variable is updated **at the end** of the procedural block

  Does not block the execution of next statements

Blocking versus Non-Blocking Assignment

Guideline: Use Non-Blocking Assignment for Sequential Logic






```
module nonblocking
(input in, clk, output reg out);
  reg q1, q2;
  always @ (posedge clk) begin
    q2  <= in;
    q1  <= q2;
    out <= q1;
  end
endmodule
```

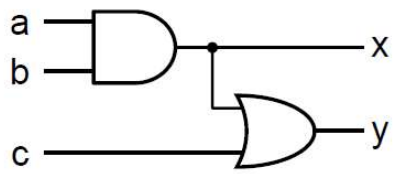
Read: in, q2, q1

Parallel Assignment at the end

```
module blocking
(input in, clk, output reg out);
  reg q1, q2;
  always @ (posedge clk) begin
    q2  = in;
    q1  = q2;  // q1 = in
    out = q1;  // out = in
  end
endmodule
```

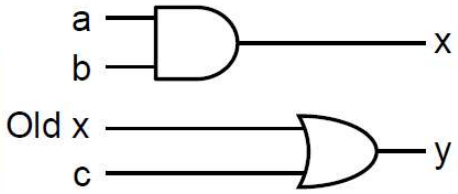
Blocking versus Non-Blocking Assignment

Guideline: Use Blocking Assignment for Combinational Logic




```
module blocking
(input a,b,c, output reg x,y);
  always @ (a, b, c) begin
    x = a & b;  // update x
    y = x | c;  // y = a&b | c;
  end
endmodule
```

Old x is Latched



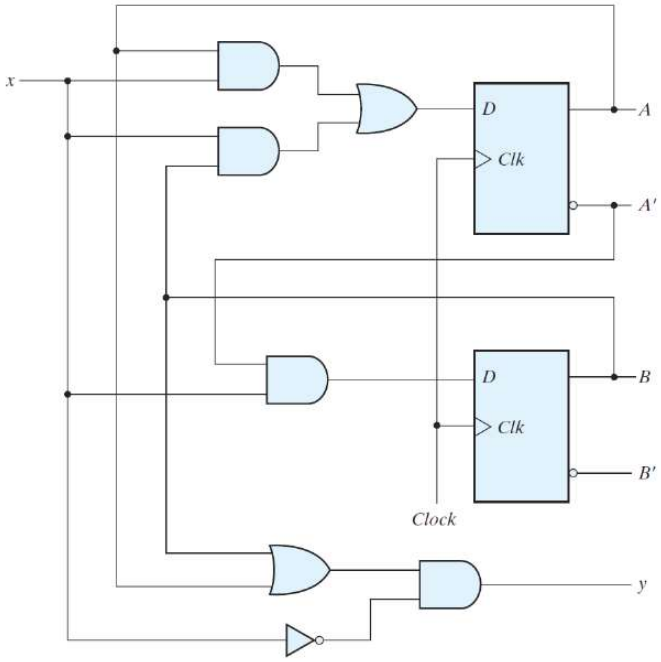

```
module nonblocking
(input a,b,c, output reg x,y);
  always @ (a, b, c) begin
    x <= a & b;
    y <= x | c;
  end
endmodule
```

Evaluate all expressions

Parallel Assignment at the end

Structural Modeling of Sequential Circuit

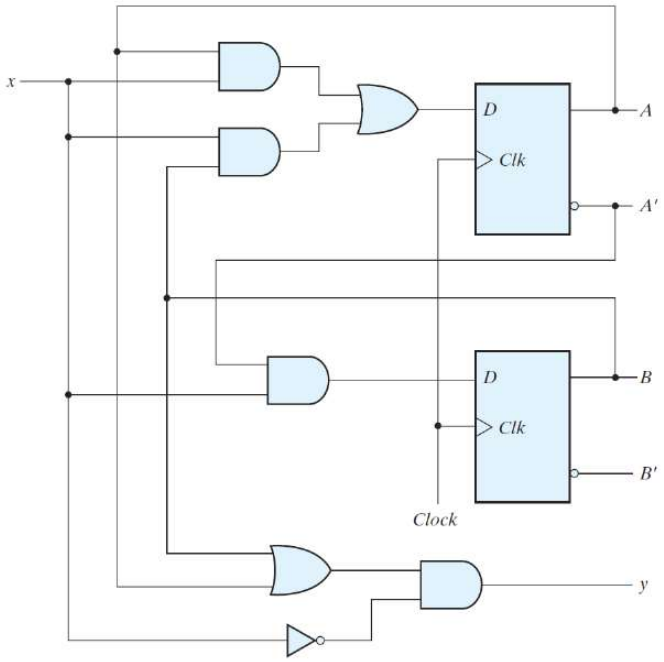
Modeling the Circuit Structure




```
// Mixed Structural and assign
module Seq_Circuit_Structure
 (input x, Clock, output y);

 wire DA, DB, A, Ab, B, Bb;

 // Instantiate two D Flip-Flops
 D_FF FFA(DA, Clock, A, Ab);
 D_FF FFB(DB, Clock, B, Bb);

 // Next state and output logic
 assign DA = (A & x) | (B & x);
 assign DB = Ab & x;
 assign y  = (A | B) & ~x;

endmodule
```

Behavioral Modeling of Sequential Circuit

Modeling the Circuit Behavior




```
module Seq_Circuit_Behavior
 (input x, Clock, output y);
 // reg A, B for the Flip-Flops
 reg A, B;
 wire DA, DB;
 // Modeling D FFs with always
 // Update A, B at positive edge
 // Non-Blocking assignment
 always @(posedge Clock)
  {A, B} <= {DA, DB};
 // Next state and output logic
 assign DA = (A & x) | (B & x);
 assign DB = ~A & x;
 assign y  = (A | B) & ~x;
endmodule
```

- **Design Code for JK Flip-flop**

```
module JK_FF(output q,qb, input j,k,clk);
reg q=0;
always@(posedge clk)
case({j,k})
    2'b00: q<=q;
    2'b01: q<=0;
    2'b10: q<=1;
    2'b11: q<=~q;
endcase
assign qb=~q;
endmodule
```

Test Bench Code for JK Flip-flop

```
module test_jkff();
reg j,k,clk=0;
always #10 clk=~clk;
JK_FF DUT(q,qb,j,k,clk);
initial begin
j<=0;k<=0;#5;
j<=0;k<=1;#20;
j<=1;k<=0;#20;
j<=1;k<=1;#20;
$finish;
end
endmodule
```

- **D flip-flop Based SISO shift register- Behavioral Modelling**

```
module SISO(input Din,clock,reset, output reg Q3);
//SISO
wire Q0,Q1,Q2;
DFF A(Q0,Din,clock,reset);
DFF B(Q1,Q0,clock,reset);
DFF C(Q2,Q1,clock,reset);
DFF D(Q3,Q2,clock,reset);
endmodule
```

```
//D FLIP FLOP

module DFF ( output reg Q, input D, Clk, rst);

always @ ( posedge Clk, negedge rst)

if (!rst) Q <= 1'b0;

else Q <= D;

endmodule
```

```
module siso_tb();
reg clk,din;
wire q3;
SISO uut(din,clk,rst,q3);
initial begin clk=1'b0;
forever #5clk=~clk;
end
initial begin
din=1; #10; din=0; #10; din=1; #10; din=0; #50;
$finish;
end
endmodule
```

- **D flip-flop Based SIPO shift register- Behavioral Modelling**

```
module SIPO(input Din,clock,reset, output reg Q0,Q1,Q2,Q3);
//SIPO
DFF A(Q0,Din,clock,reset);
DFF B(Q1,Q0,clock,reset);
DFF C(Q2,Q1,clock,reset);
DFF D(Q3,Q2,clock,reset);
endmodule
```

```
//D FLIP FLOP

module DFF ( output reg Q, input D, Clk, rst);

always @ ( posedge Clk, negedge rst)

if (!rst) Q <= 1'b0;

else Q <= D;

endmodule
```

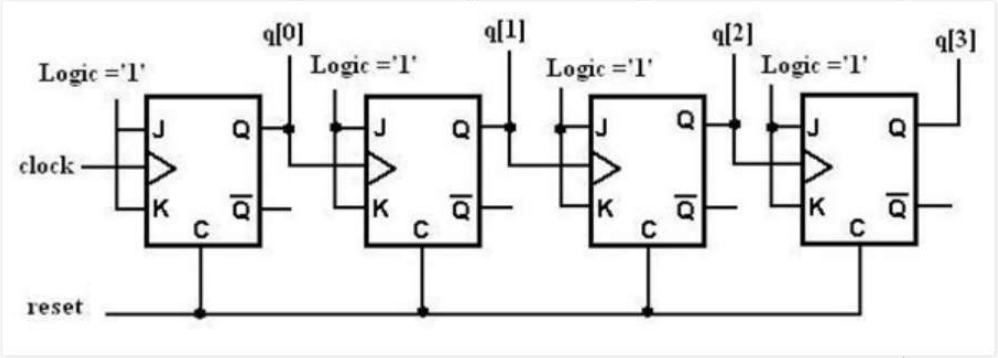
```
module sipo_tb();
reg clk,din;
wire q0,q1,q2,q3;
SIPO uut(din,clk,rst,q0,q1,q2,q3);
initial begin clk=1'b0;
forever #5clk=~clk;
end
initial begin
din=1; #10; din=0; #10; din=1; #10; din=0; #50;
$finish;
end
endmodule
```

Asynchronous Counters Up counter

Verilog Code for jkff: (Behavioural model)

```
module jkff(j,k,clock,reset,q,qb);
input j,k,clock,reset;
output reg q,qb;
always@(negedge clock)
begin
case({reset,j,k})
3'b100 :q=q;
3'b101 :q=0;
3'b110 :q=1;
3'b111 :q=~q;
default :q=0;
endcase
qb<=~q;
end
endmodule
```

Circuit Diagram for 4-bit Asynchronous up counter using JK-FF:



Verilog Code for 4-bit Asynchronous up counter using JK-FF (Structural model):

```
module ripple_count(j,k,clock,reset,q,qb);
input j,k,clock,reset;
output wire [3:0]q,qb;


jkff JK1(j,k,clock,reset,q[0],qb[0]);
jkff JK2(j,k,q[0],reset,q[1],qb[1]);
jkff JK3(j,k,q[1],reset,q[2],qb[2]);
jkff JK4(j,k,q[2],reset,q[3],qb[3]);


endmodule
```

- Mod-N counter(Please look into the code in lab files for Mod-N Counter)