Unit 7

Registers and Counters

Contents:

- Resisters
- Shift registers
- Ripple Counters
- Synchronous Counters
- Timing Sequences
- The Memory Unit

Registers

- A group of flip-flops suitable for holding binary information.
- 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.
- The transfer of new information into a register is referred to as *loading* the register.
- If all the bits of the register are loaded simultaneously with a single clock pulse, we say that the loading is done in parallel.
- If the loading is one bit at time, the loading is said to be serial.

- Following fig. shows such a register constructed with four D-type flip-flops and a common clock-pulse input.

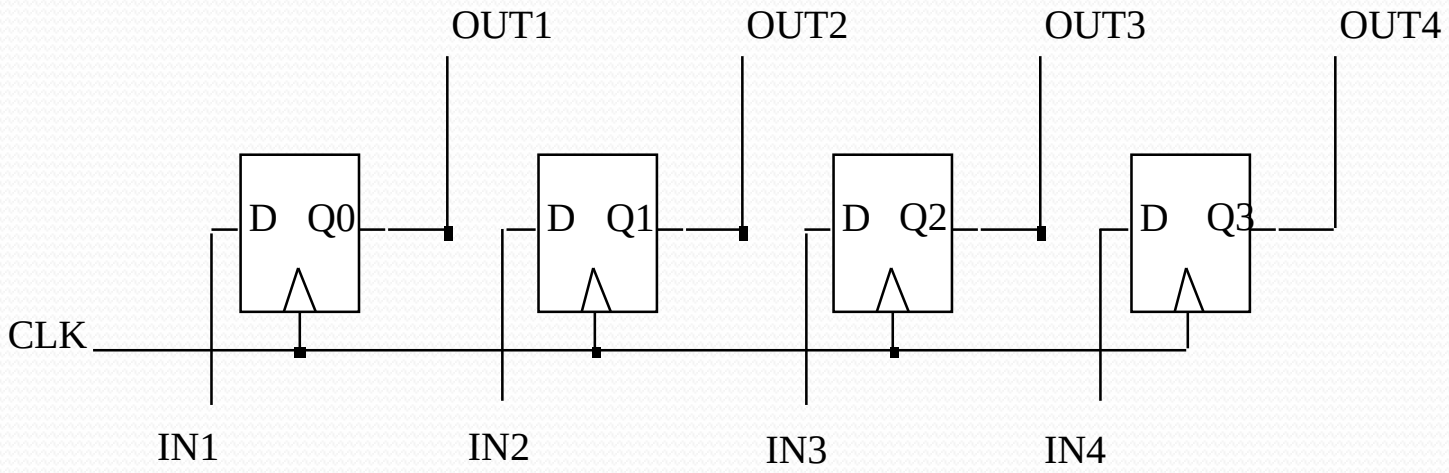


Fig: 4-bit register

- The clock pulse input, CP, enables all flip-flops, so that the information presently available at the four inputs can be transferred into the 4-bit register.
- The four outputs can be sampled to obtain the information presently stored in the register.

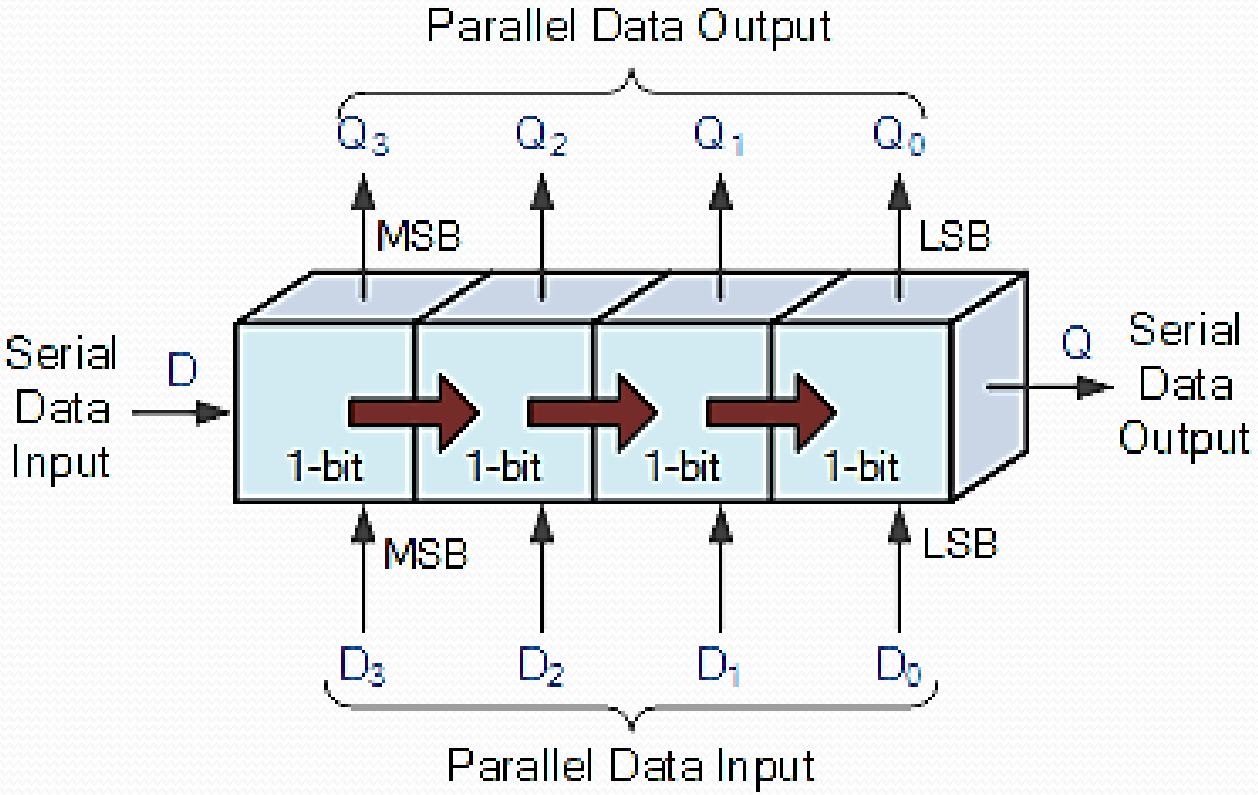
- A pulse applied to the *CP* input of the register of Fig. above will load all four inputs in parallel.
- When *CP* goes to 1, the input information is loaded into the register.
- If *CP* remains at 0, the content of the register is not changed.
- Note that the change of state in the outputs occurs at the positive edge of the pulse.

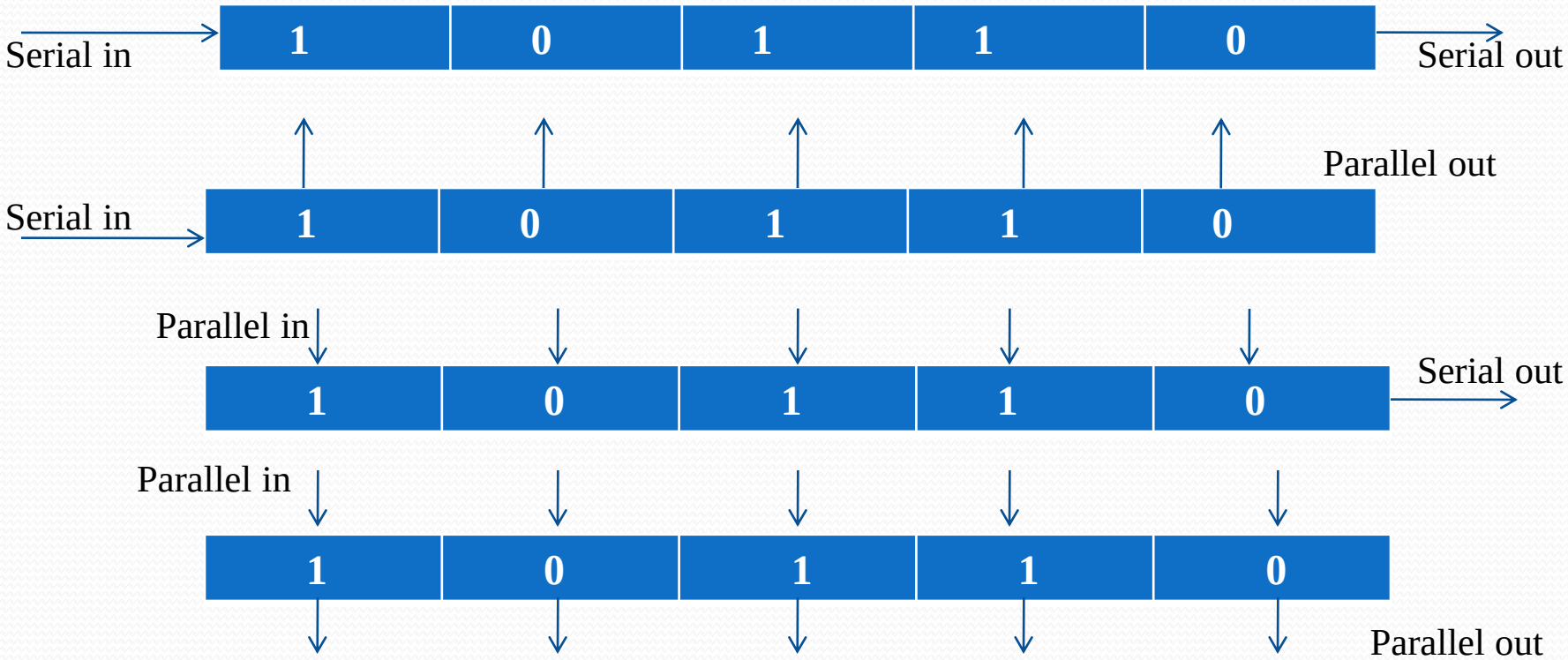
Shift Registers

- 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.
- Shift registers are type of sequential logic circuits mainly for storage of digital data. The word shift means the device which shift data right or left. i.e. calculator.

Shift registers operate in one of **four different mode.**

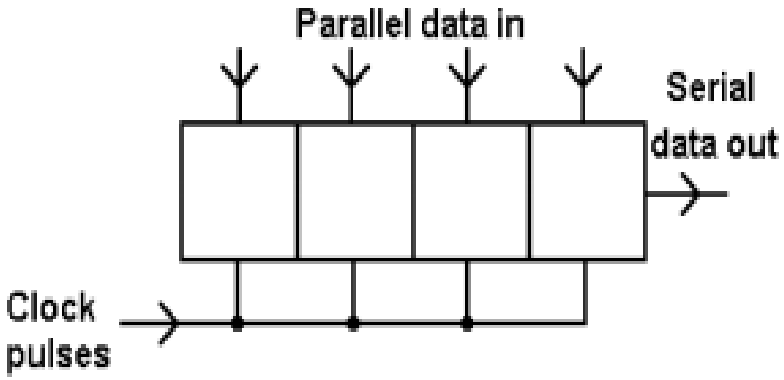
- Serial in, Serial out (SISO)
- Serial in, Parallel out (SIPO)
- Parallel in, Serial out (PISO)
- Parallel in, Parallel out (PIPO)

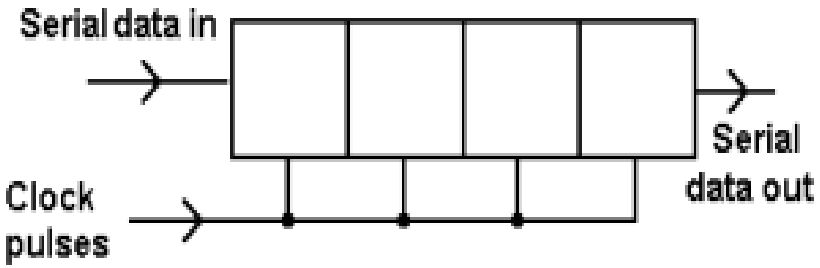



Serial in
1
0
1
1
0
Serial out
Parallel out
Serial in
1
0
1
1
0
Parallel in
1
0
1
1
0
Serial out
Parallel in
1
0
1
1
0
Parallel out

Serial Data Transfer

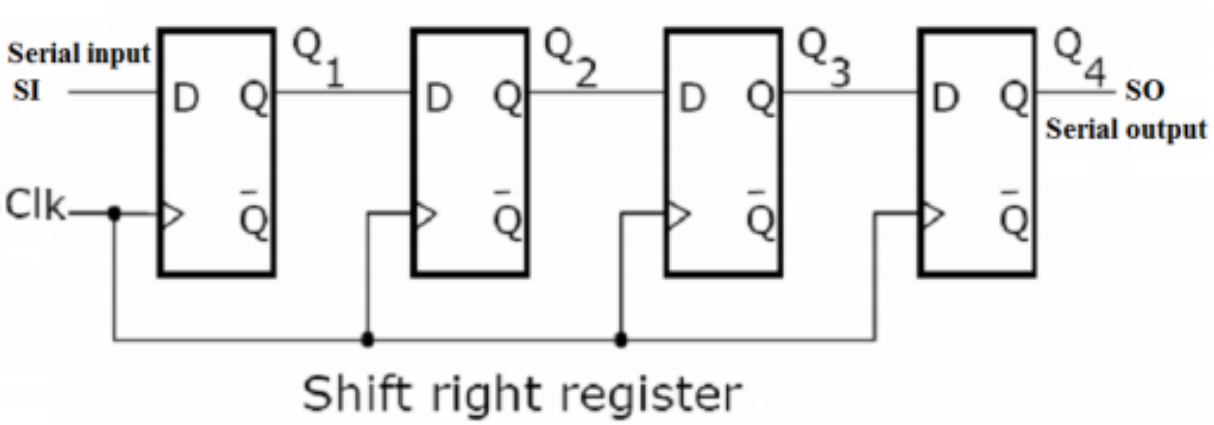
A digital system is said to operate in serial mode , when information is transferred and manipulated one bit at a time. Information is transferred one bit at a time by shifting the bits out of the source register into the destination register





Serial-in to Serial-out (SISO)

	SI	Q_1	Q_2	Q_3	Q_4
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1



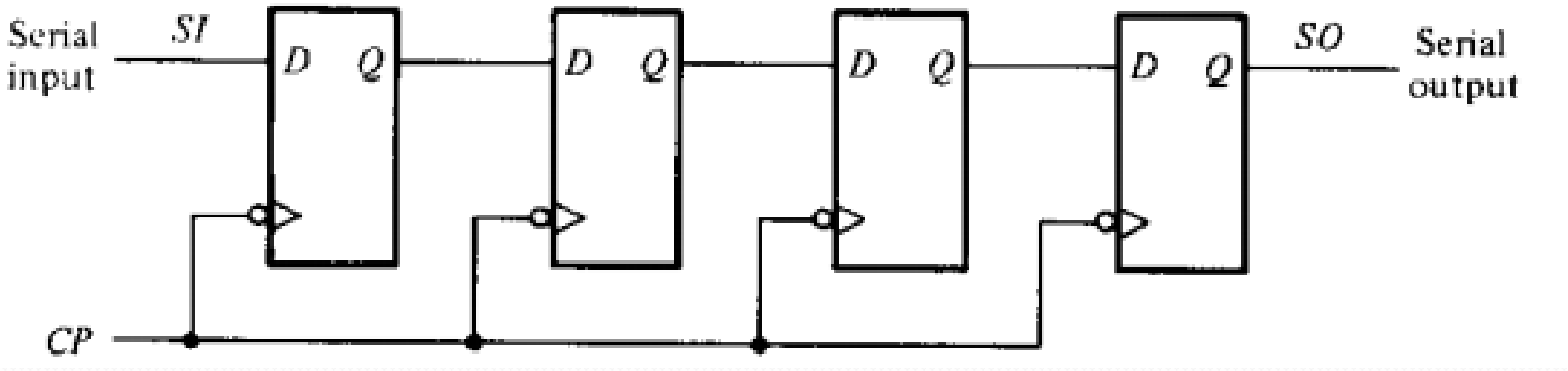
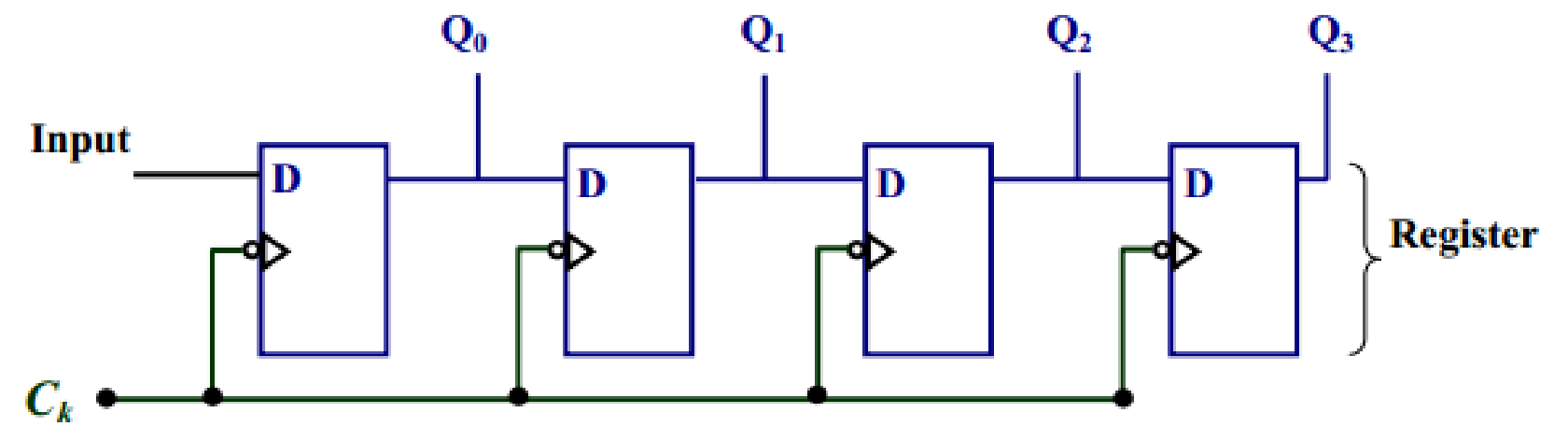


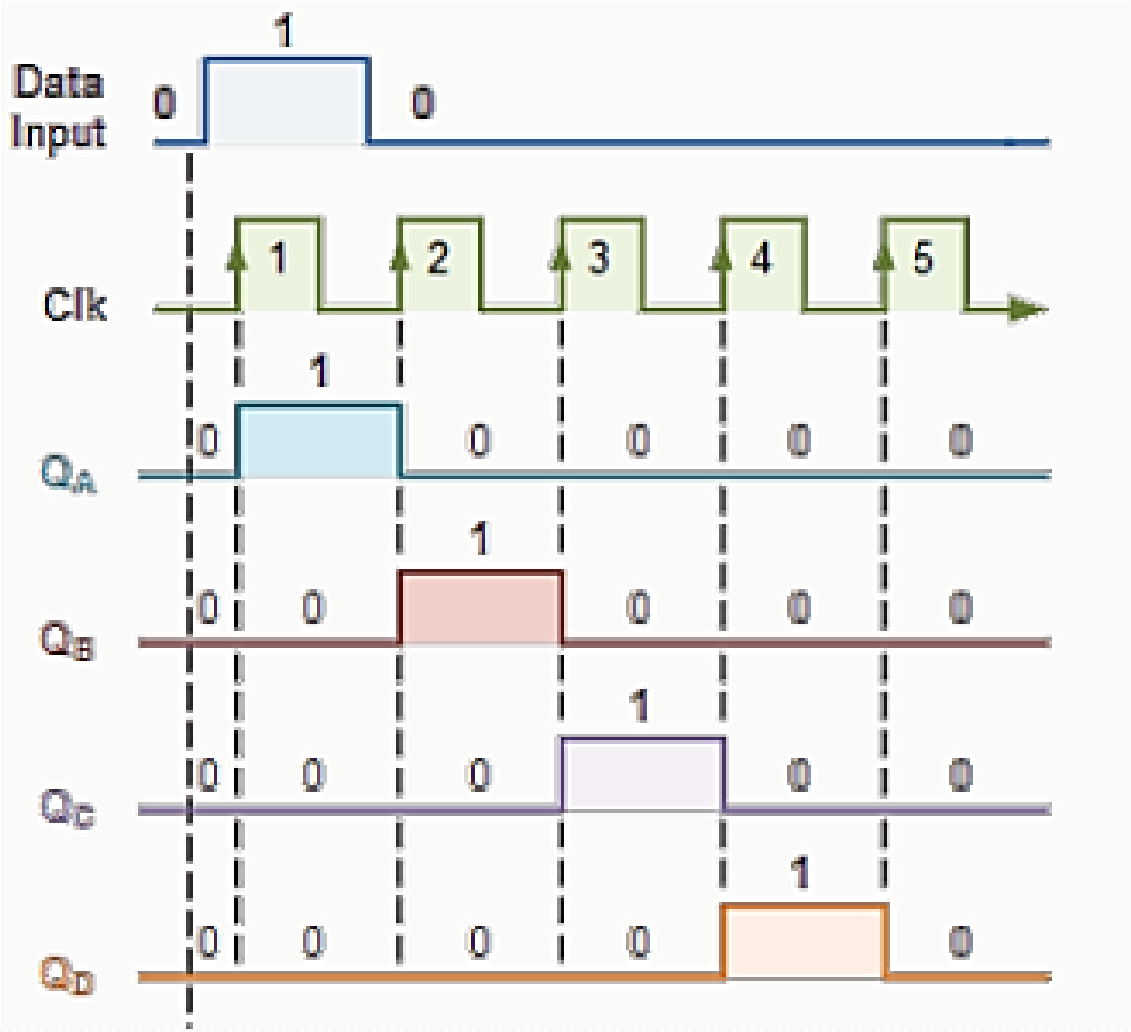
Fig: 4-bit SISO

What's the point of a SISO shift register if the output data is exactly the same as the input data?

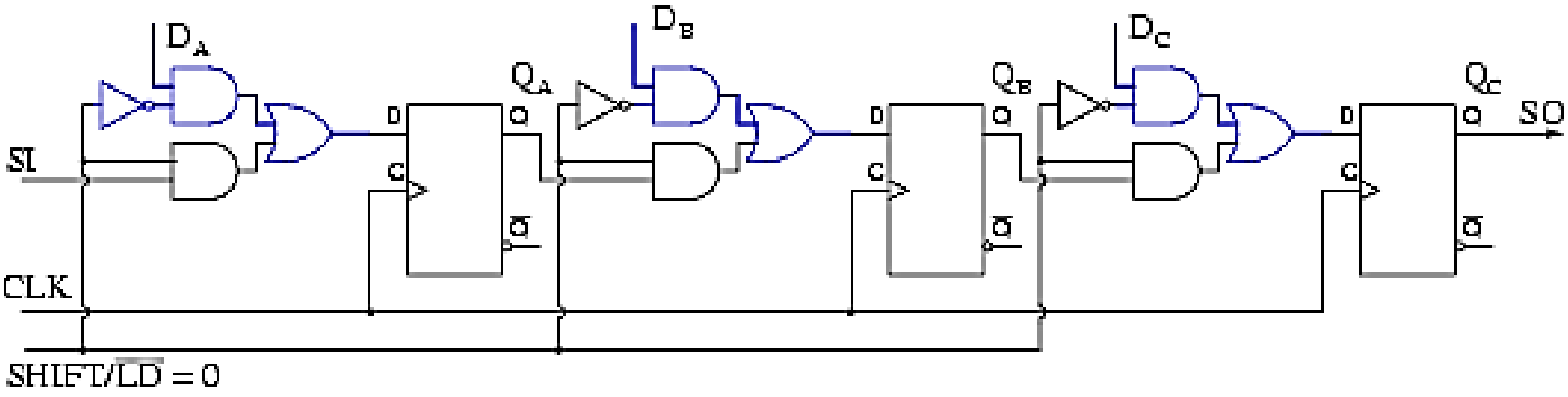
This type of Shift Register also acts as a temporary storage device or as a time delay device for the data.

Serial-in to Parallel-out (SIPO)




Data Input
Clk
Q_A
Q_B
Q_C
Q_D
1
2
3
4
5
0
1

Parallel-in to Serial-out (PISO)



Parallel-in/ serial-out shift register showing parallel load path

Advantage: As this type of shift register converts parallel data, such as an 8-bit data word into serial format, it can be used to multiplex many different input lines into a single serial DATA stream which can be sent directly to a computer or transmitted over a communications line.

Parallel-in to Parallel-out (PIPO)

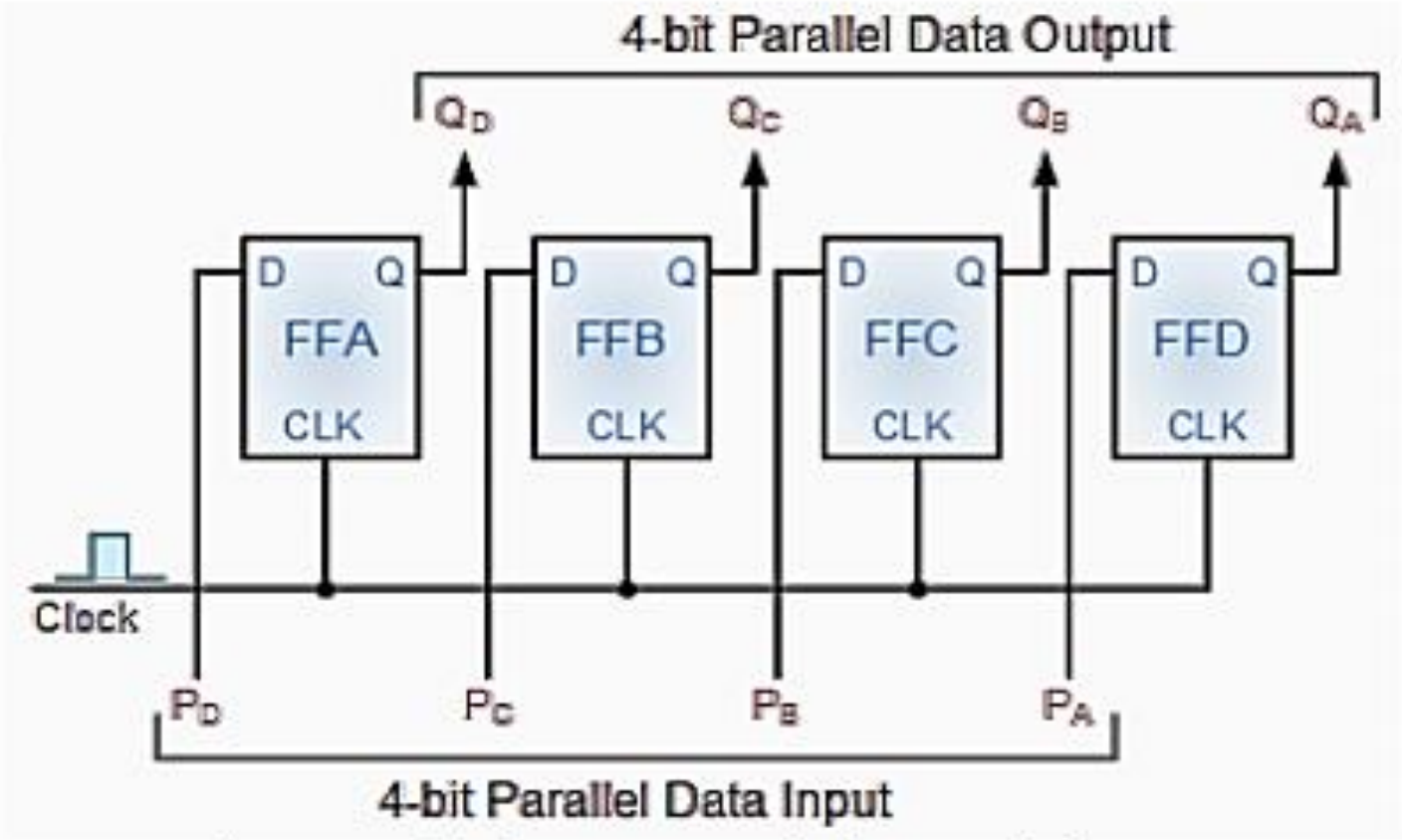


Fig: 4-bit Parallel-in to Parallel-out Shift Register

Application of shift register

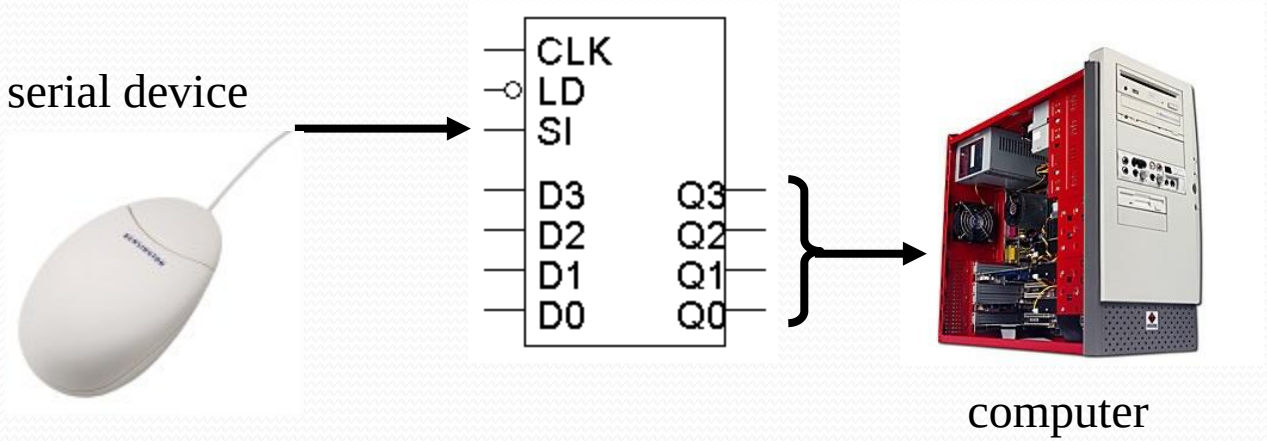
i. To produce time delay,
ii. One application of shift registers is converting between “serial data” and “parallel data”

Sometimes it’s necessary to send or receive data serially, or one bit at a time. Some examples include:

- Input devices such as keyboards and mice
- Output devices like printers

Receiving serial data

- To *receive* serial data using a shift register:
  - The serial device is connected to the register's SI input
  - The shift register outputs Q3-Q0 are connected to the computer
- The serial device transmits one bit of data per clock cycle
  - These bits go into the SI input of the shift register
  - After four clock cycles, the shift register will hold a four-bit word
- The computer then reads all four bits at once from the Q3-Q0 outputs.

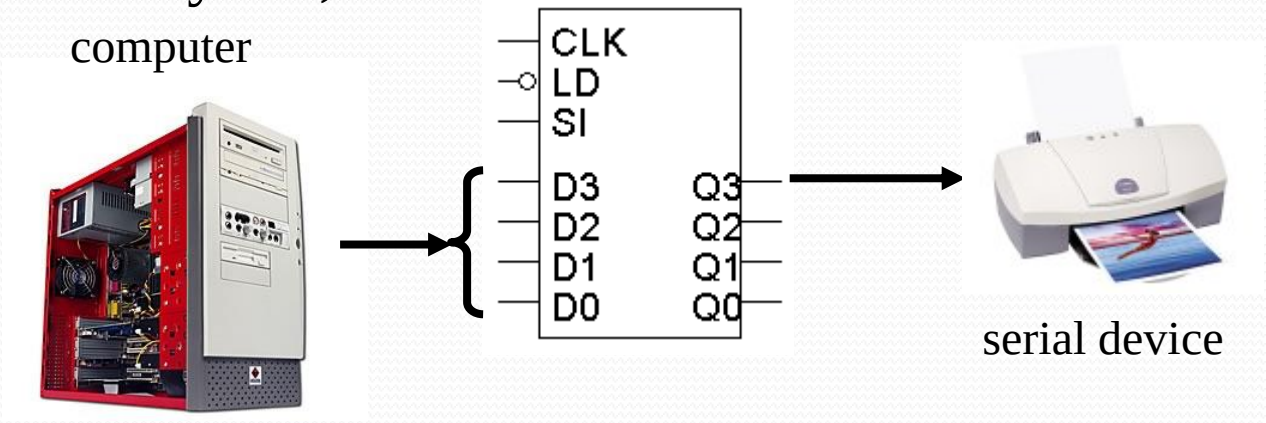


Sending data serially

- To *send* data serially with a shift register, you do the opposite:
  - The CPU is connected to the register's D inputs
  - The shift output (Q3 in this case) is connected to the serial device

- The computer first stores a four-bit word in the register, in one cycle.

- The serial device can then read the shift output
  - One bit appears on Q3 on each clock cycle
  - After four cycles, the entire four-bit word will have been sent



Counters

It is a sequential circuit consisting a set of flip-flop connected in a suitable manner to count the sequence of the input pulse presented to it in digital form.

- An example of a register.
- Each stored 0/1 combination is called the state of the counter.
- The total number of states is called its modulus.
  - If a counter has m distinct states then it is called a mod-m counter.
- The order in which states appear is referred to as its counting sequence.

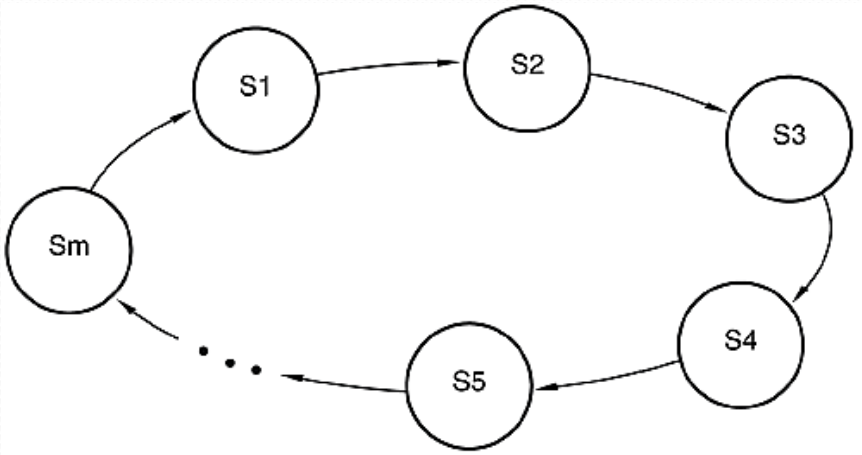


Fig: General Structure of a counter's state diagram

Counters are available in two types:

- Synchronous Counters
- Ripple counter (Asynchronous Counter)

Ripple Counter (Asynchronous Counter)

- flip-flop output transition serves as a source for triggering other flip-flops.
- In other words, the *clock* inputs of all flip-flops (except the first) are triggered not by the incoming pulses, but rather by the transition that occurs in other flip-flops.
- No common clock pulse.

Thus, when the output of flip-flop is used as the clock input for the next flip-flop, such counter is termed as ripple or asynchronous counter.

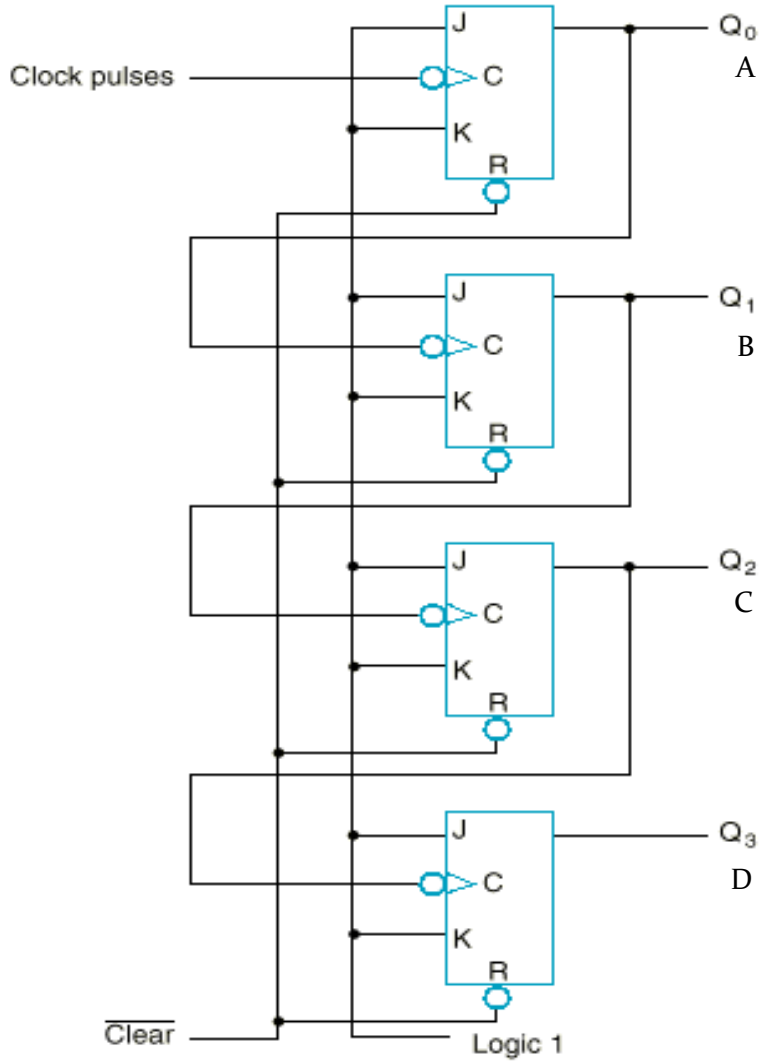
Binary Ripple Counter

- Counters whose counting sequence corresponds to that of the binary numbers are called binary counters.
- Modulus is 2^n, where n is the number of flip-flops in the counter.

Design of n-bit counter/ MOD 2^n Counter

- n flip-flops are required.
- Generally JK flip-flop are used.
- First Flip-flop is driven by a clock pulse and its (normal or complement) output drives second flip-flop and so on.
- All the JK flip-flop is provided with 1.
- The flip-flop can be either positive or negative edge triggered.
- If Negative-Edge Triggered is used:
  - Normal Output for Up-Counter.
  - Complement Output for Down-Counter
- If Positive-Edge Triggered is used:
  - Normal Output for Down-Counter.
  - Complement Output for UP-Counter
- First flip-flop is LSB and Last Flip-flop is MSB.

4-bit negative edge triggered Binary Ripple Up Counter using JK FF



D	C	B	A	State or Count
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15
0	0	0	0	0

Repeats So on.

A toggles on every clock cycle.
B toggles on those clock cycle when A changes from 1 to 0
C toggles on those clock cycle when A and B changes from 1 to 0
D toggles on those clock cycle when A, B and C changes from 1 to 0

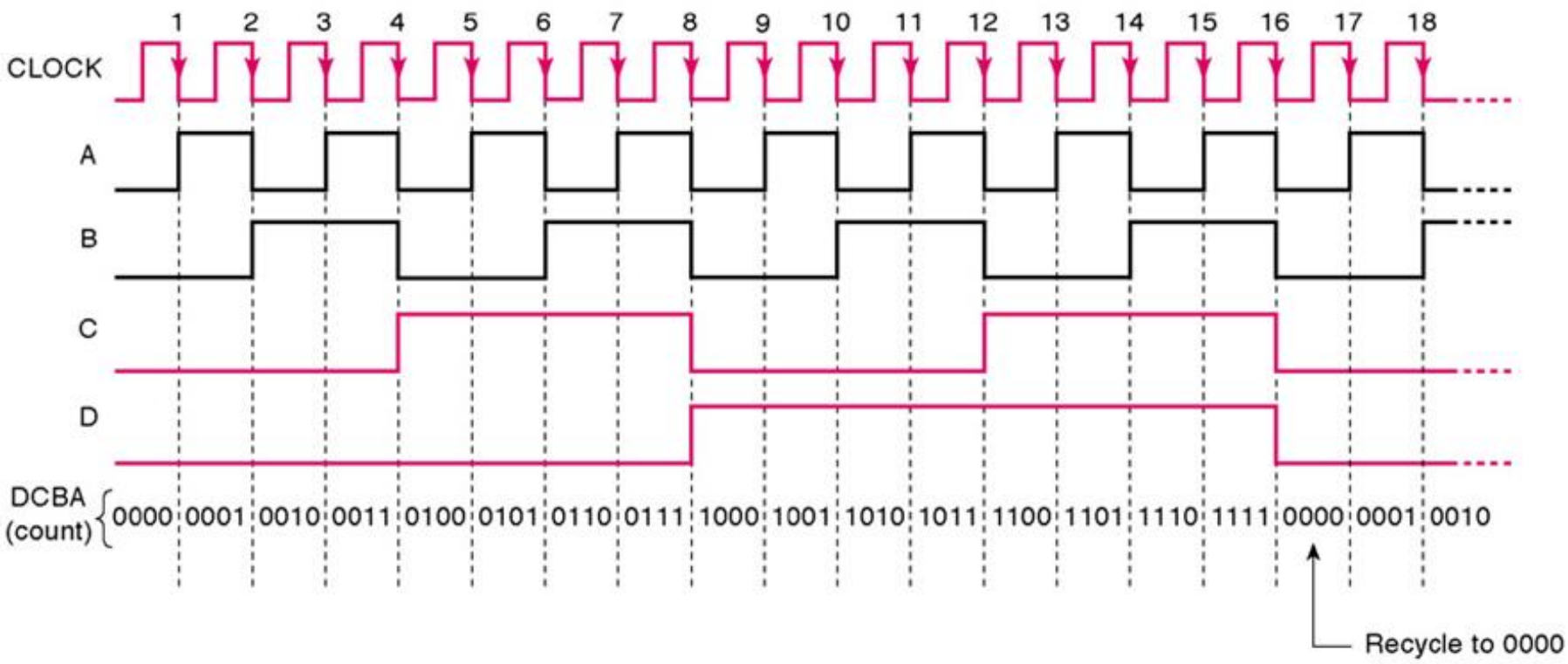
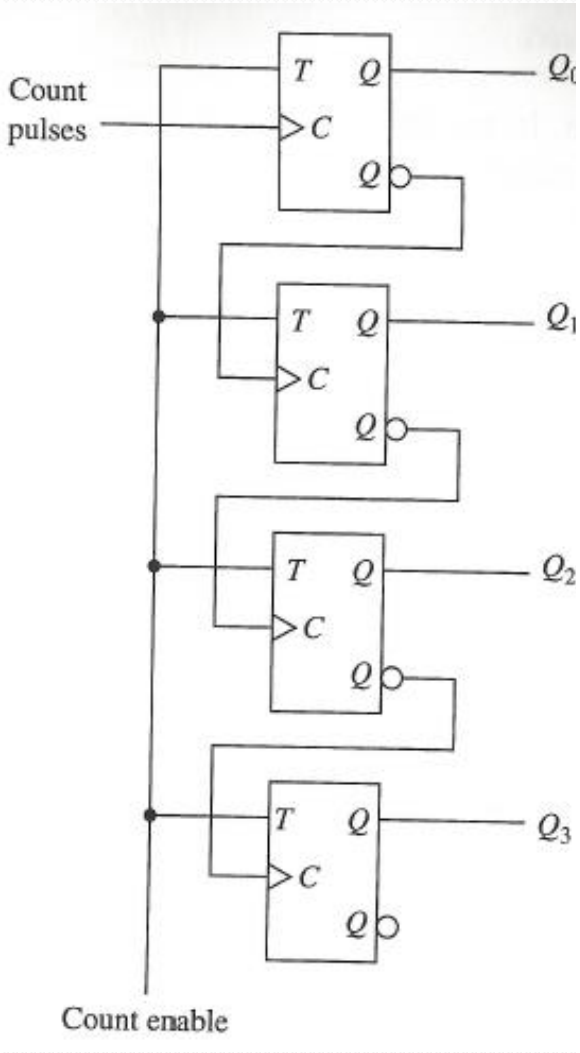


Fig: Timing diagram of 4-bit negative edge triggered up-Binary Ripple Counter

4-bit positive edge-triggered Binary Ripple Up Counter using T FF

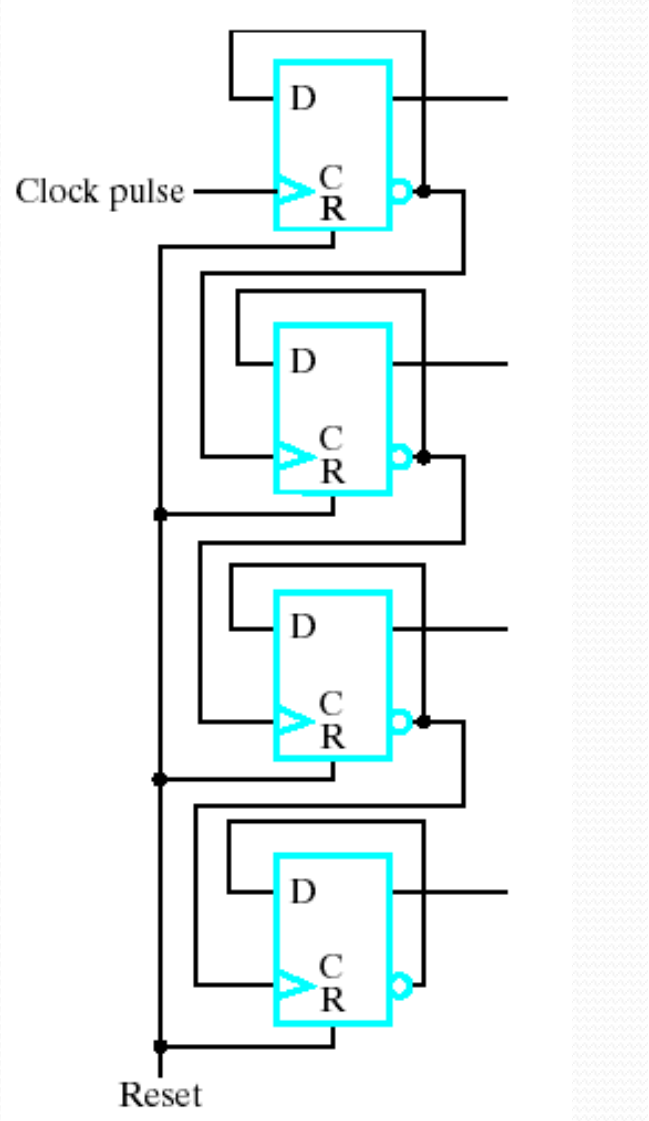
- Each positive transition from logic-0 to logic-1 causes the flip-flop to toggle.



Q_3	Q_2	Q_1	Q_0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15
0	0	0	0	

etc.

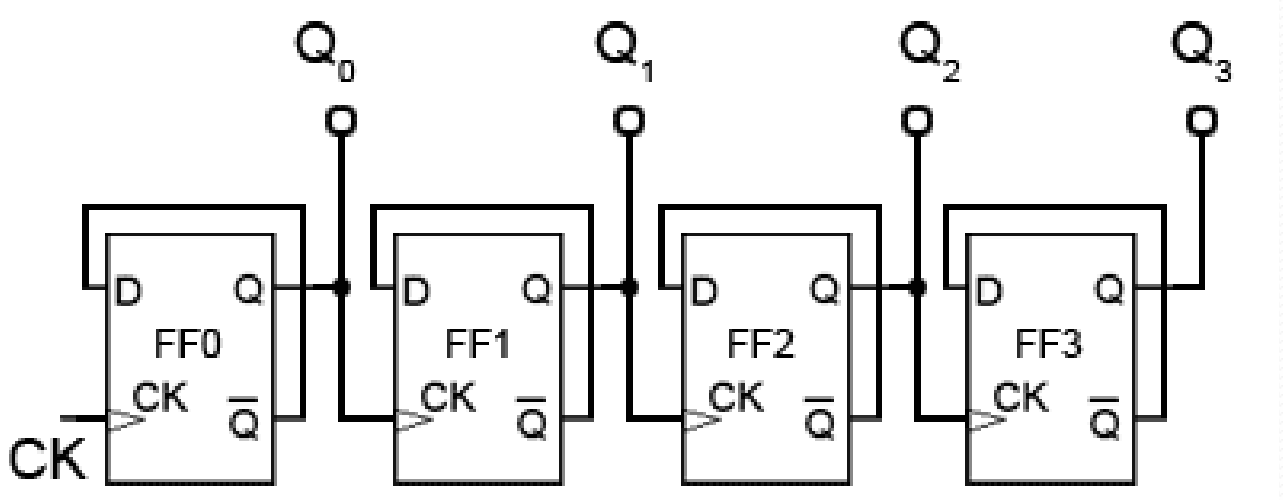
4-bit Positive Edge Triggered Binary Ripple Up Counter using D FF



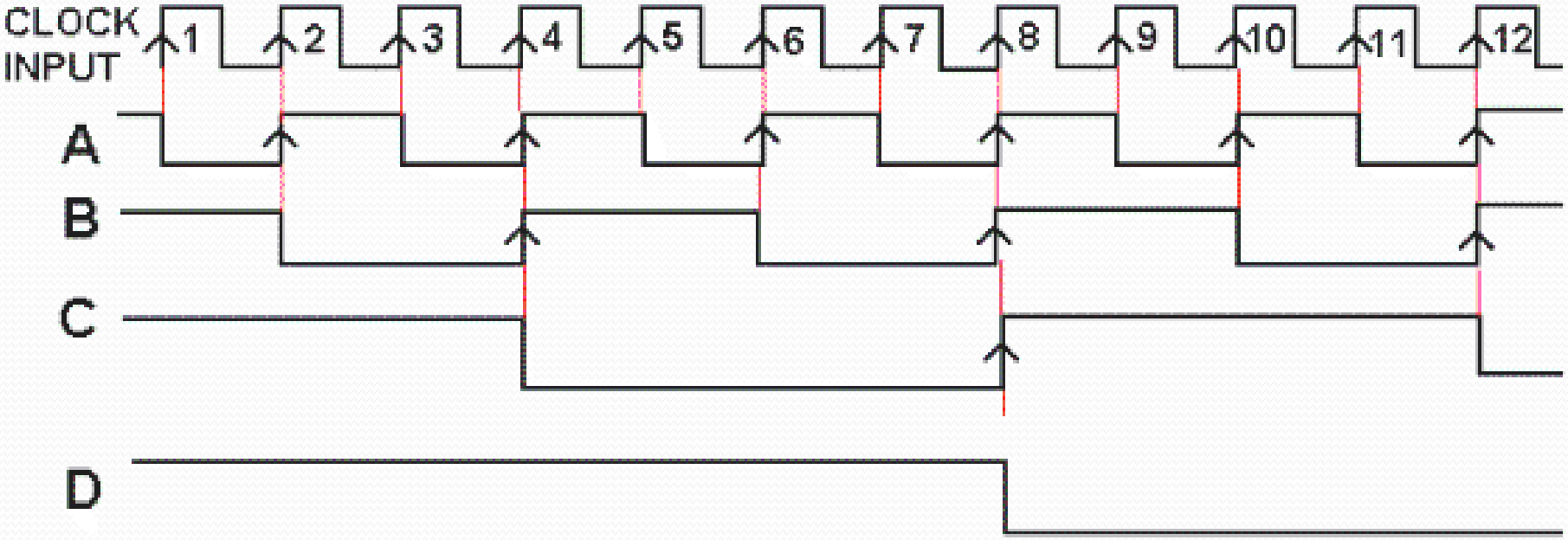
Upward Counting Sequence

Q_3	Q_2	Q_1	Q_0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

4-bit Positive Edge Triggered Binary Ripple Down Counter using D FF



4-bit Positive edge triggered Binary Ripple Down Counter



Timing Diagram

4-bit negative edge triggered Binary Ripple Down Counter

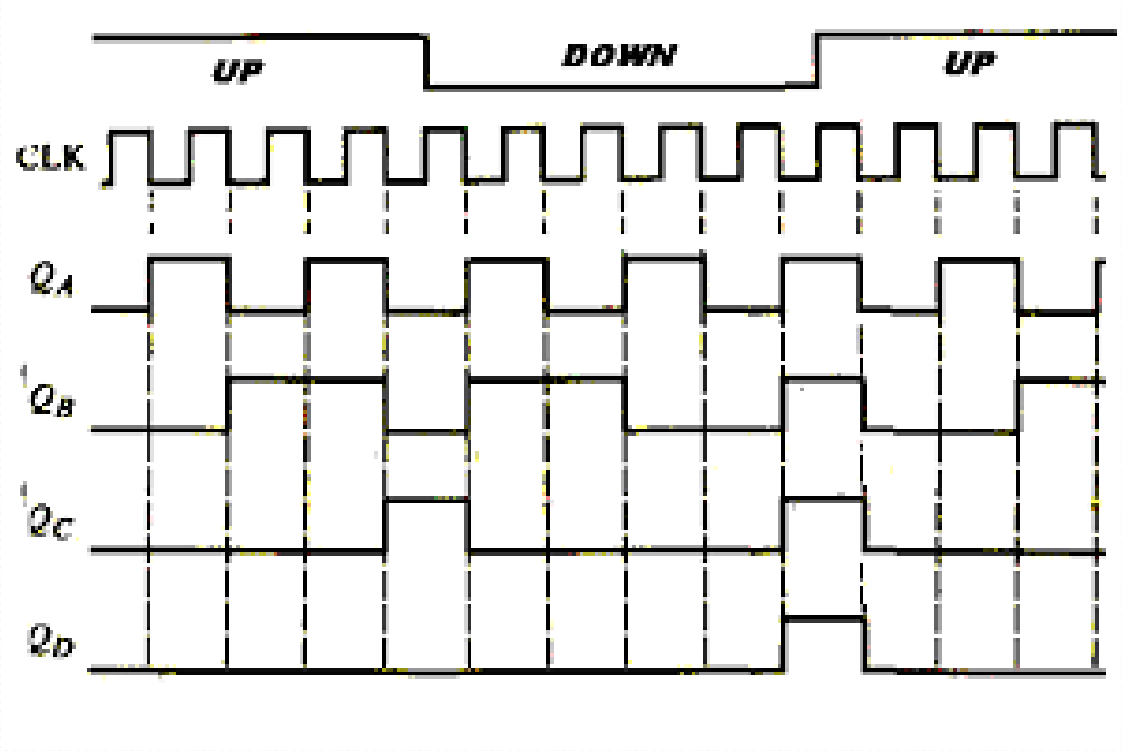
1111→1110→1101→1100→1011→1010→1001→1000→0111→0110
→0101→0100→0011→0010→0001→0000→1111→So on…

A toggles on every clock cycle.
B toggles on those clock cycle when A changes from 0 to 1.
C toggles on those clock cycle when A and B changes from 0 to 1.
D toggles on those clock cycle when A, B and C changes from 0 to 1.

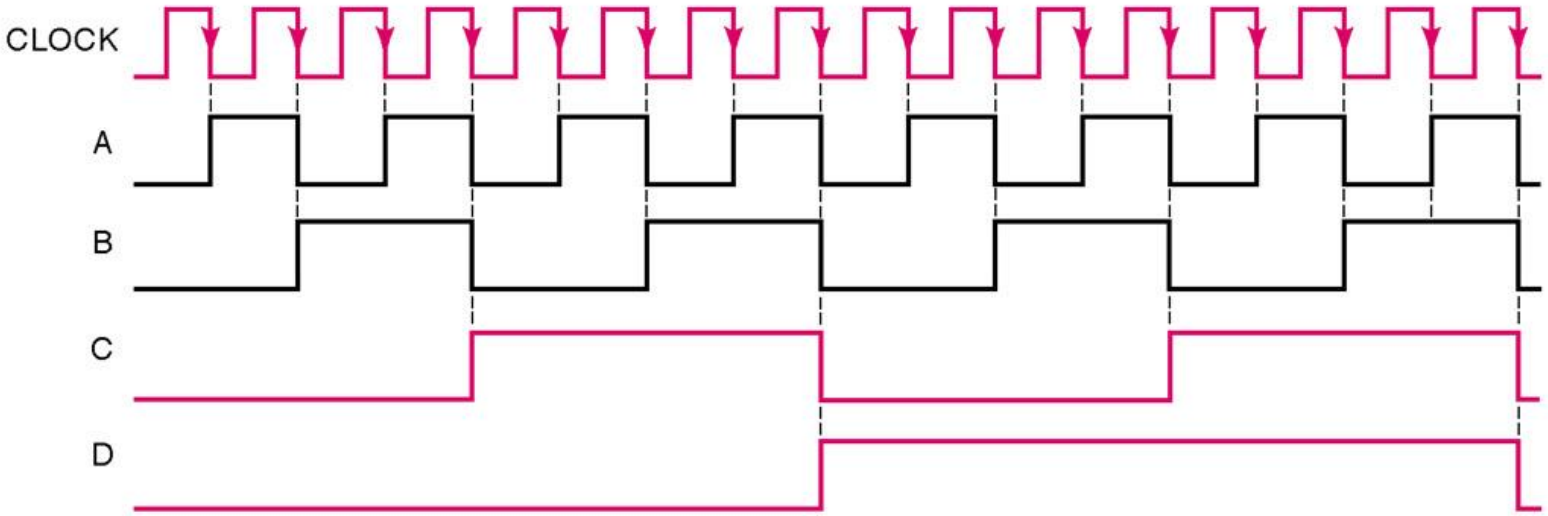
- If Negative-Edge Triggered is used:
  - Complement Output for Down-Counter
- If Positive-Edge Triggered is used:
  - Normal Output for Down-Counter.

4-bit negative edge triggered Binary Ripple Up/Down Counter

Similar logic as before for individual Up and Down Counter with the control option for selection of up and down respectively.



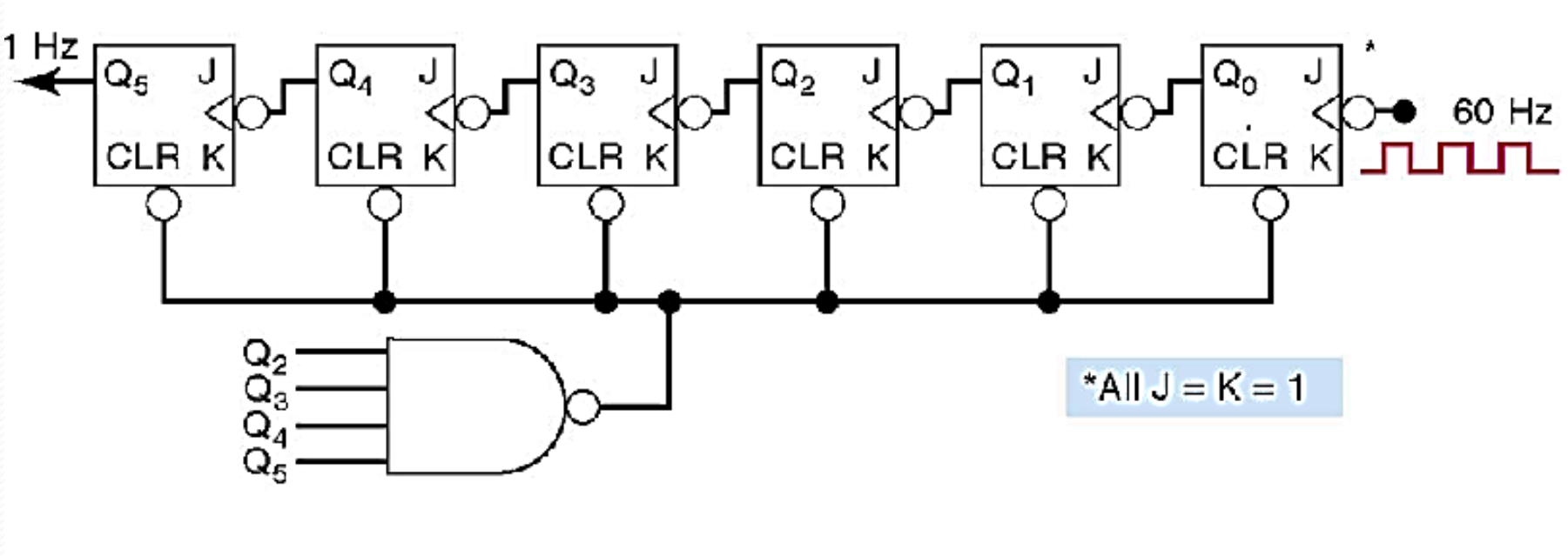
Frequency division



In any counter, the signal at the output of the last F/F(i.e., the MSB) will have a frequency equal to the input clock frequency divided by the MOD number of the counter. Such circuits are known as *divide-by-N counters*.

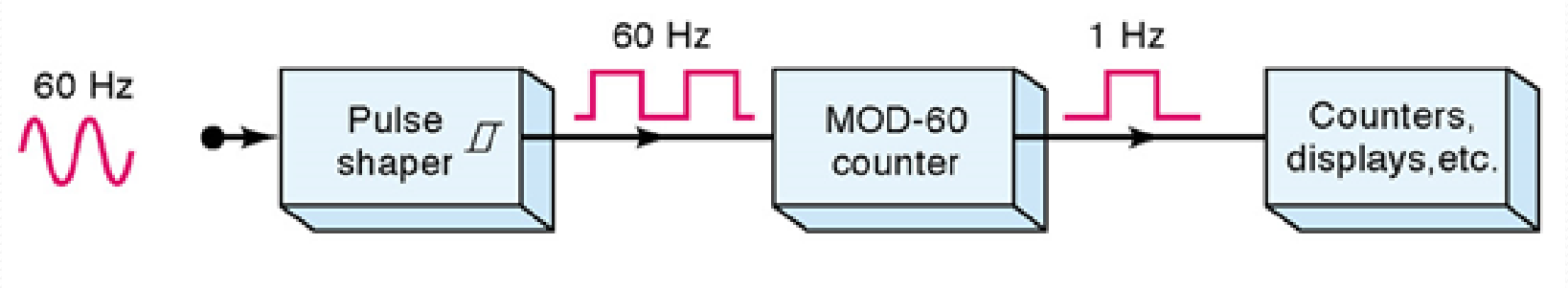
Q. Construct an appropriate MOD-60 counter.

60 = 111100



Example

- The first step involved in building a digital clock is to take the 60-Hz signal and feed it into a Schmitt-trigger, pulse-shaping circuit to produce a square wave as illustrated in Figure below. The 60-Hz square wave is then put into a MOD-60 counter, which is used to divide the 60-Hz frequency by exactly 60 to produce a 1-Hz waveform. This 1-Hz waveform is fed to a series of counters, which then count seconds, minutes, hours, and so on. How many F/Fs are required for the MOD-60 counter?



BCD Ripple Counter/ Asynchronous Decade Counter/MOD-10 Counter

- **Decade counter**

  Any counter has 10 distinct states, no matter what the sequence.

- **BCD counter**

  A decade counter that counts through 0-9 i.e. 0000 -1001.

  When the output is 1010, all the flip-flop are cleared.

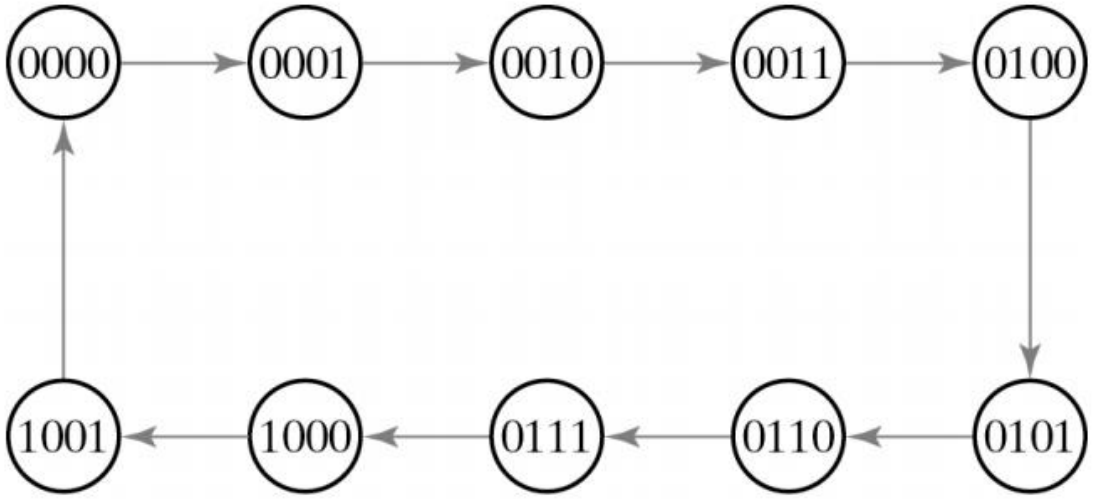


Fig. 6-9 State Diagram of a Decimal BCD-Counter

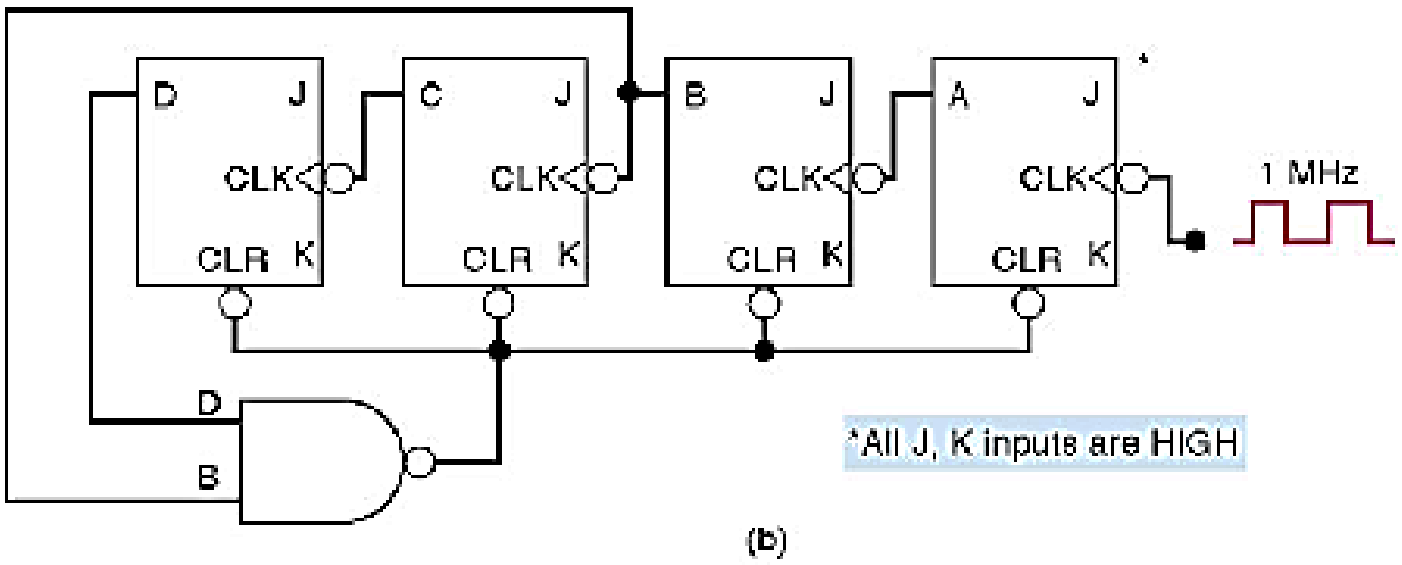


Fig: BCD Ripple Counter/ Asynchronous Decade Counter/MOD-10 Counter

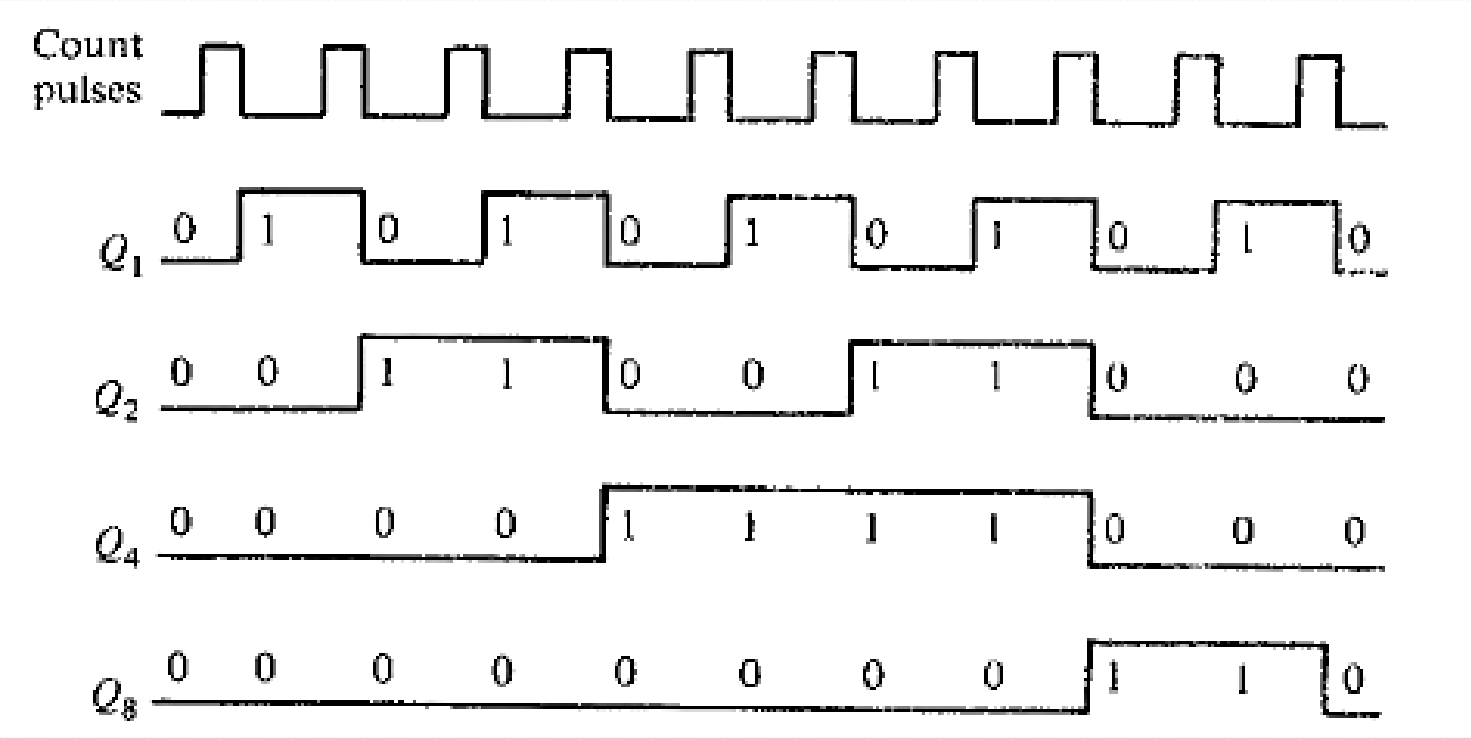


Fig: Timing Diagram of Negative Edge Triggered Decade / BCD counter

To count in decimal from 0 to 999, we need a three-decade counter connected as below. Multiple decade counters can be constructed by connecting BCD counters in cascade, one for each decade.

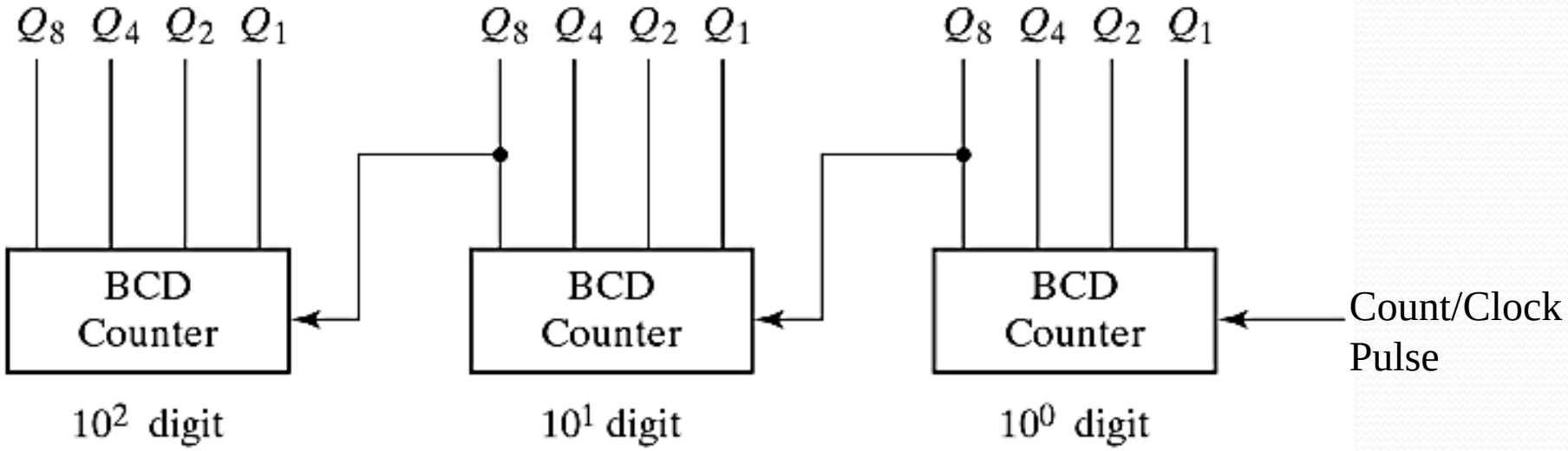
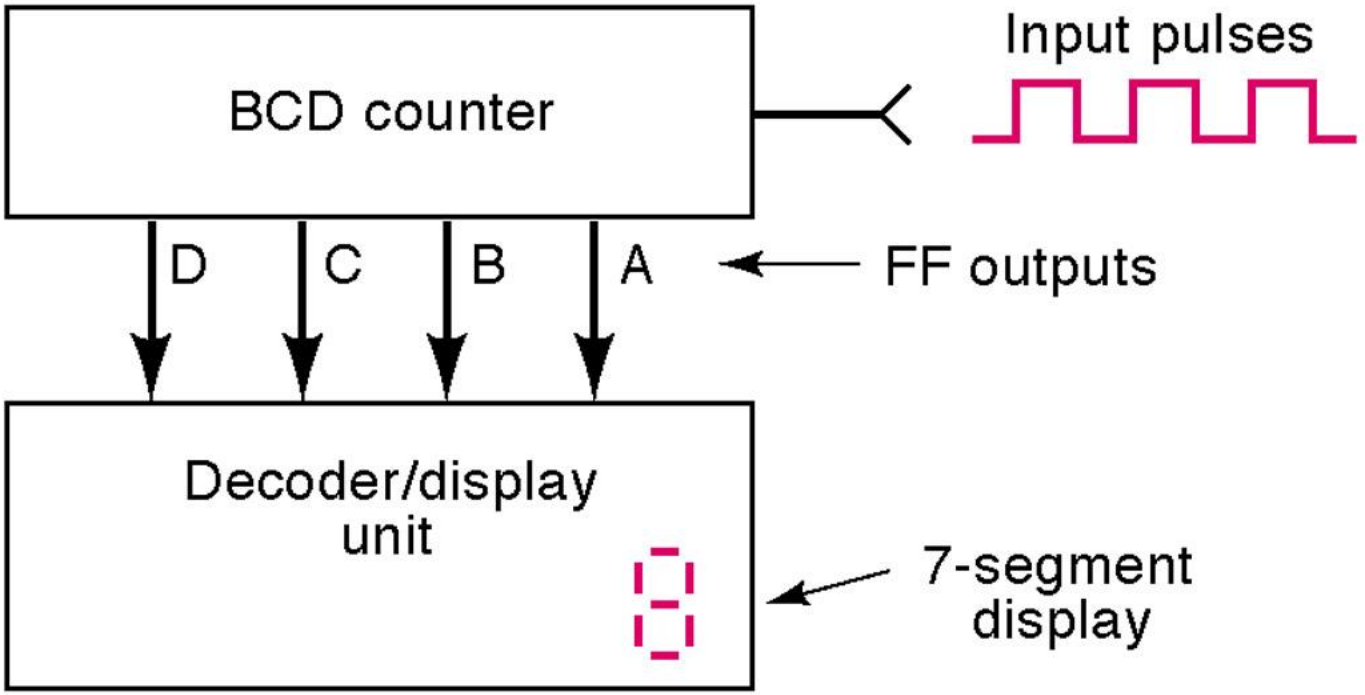


Fig. 6-11 Block Diagram of a Three-Decade Decimal BCD Counter

BCD Counter Decoding



Cascading BCD counters

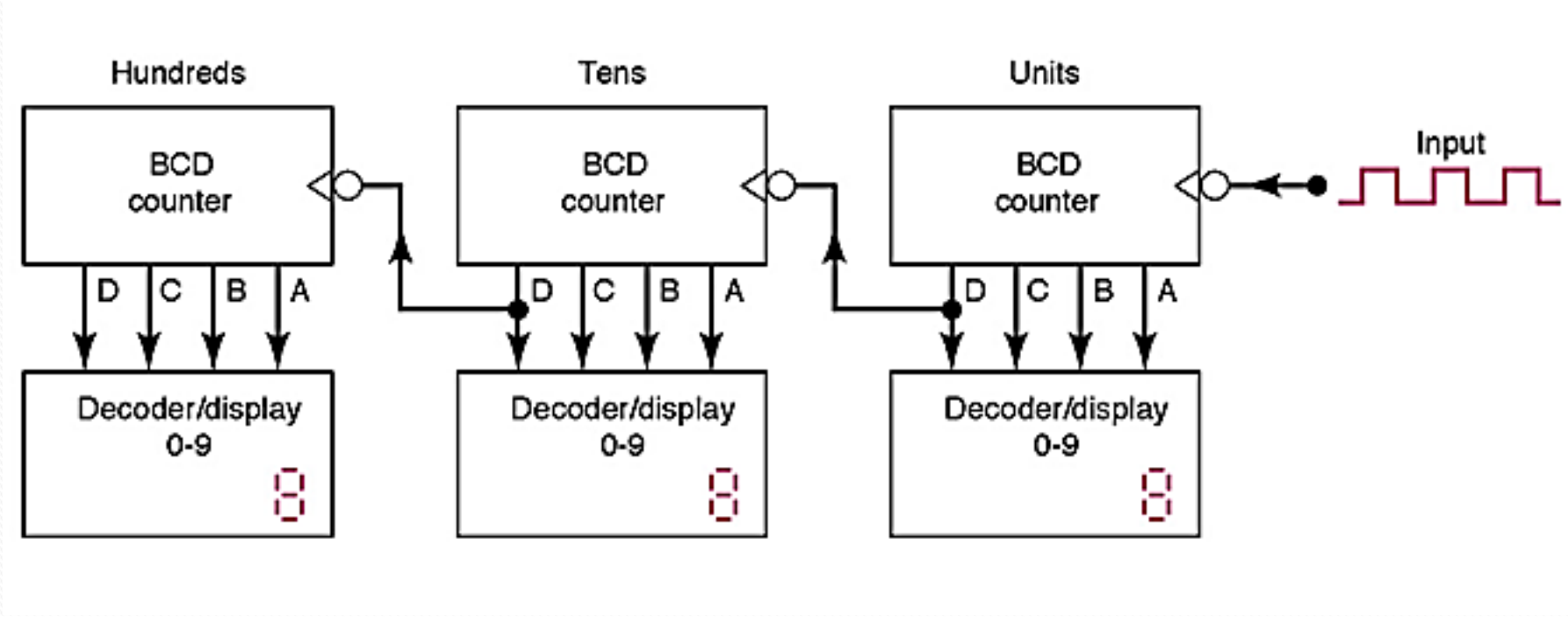


Fig: Three-decade counter with display

Synchronous Counter

- All flip-flops change simultaneously after the appropriate propagation delay associated with a single flip-flop.
- Count pulses are applied directly to the control inputs, C, of all the clocked flip-flops.

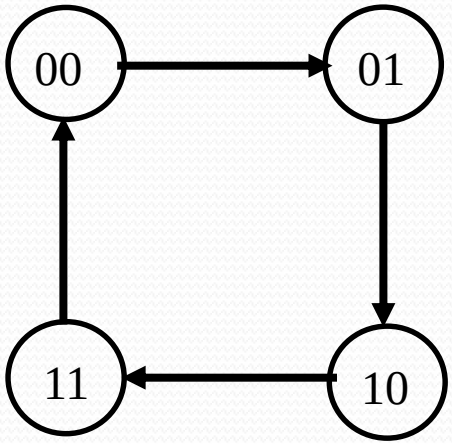
Synchronous Binary Up Counter

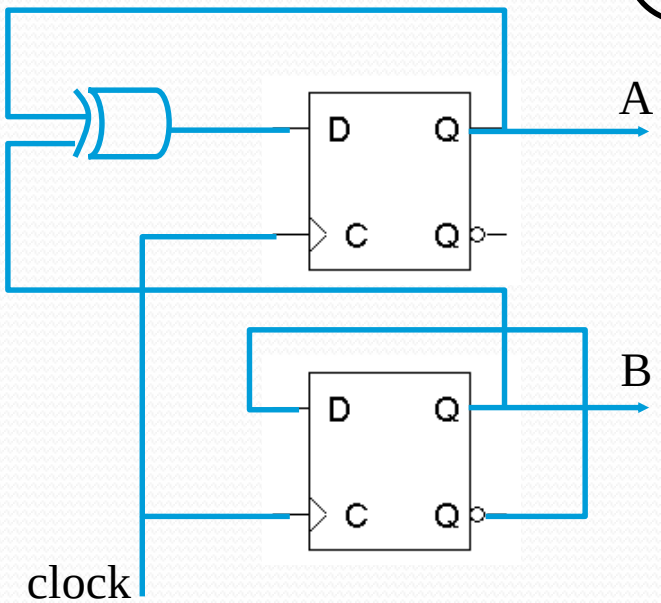
Example: MOD-2 or 2-bit Positive Edge Triggered Up counter using D-flip-flop

Present State		Next State	
A	B	A	B
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

D1= A'B + AB'

D0= B'





Now try using
JK F/F and T F/F

Synchronous Binary Counters:

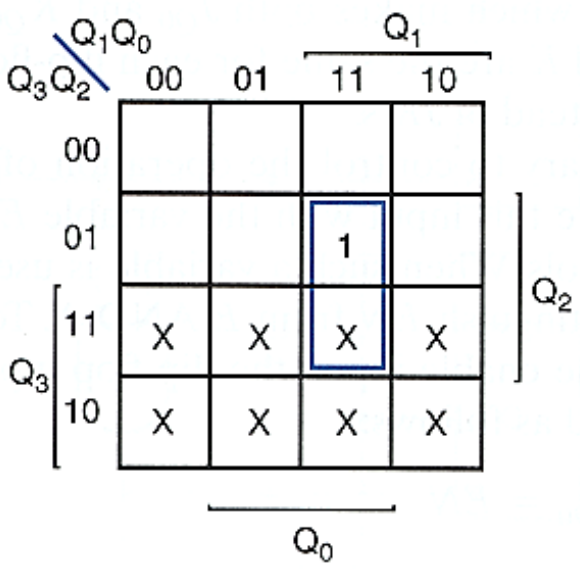
Design of a 4-bit Binary Up Counter by using J-K Flip Flop

TABLE 4-10
Flip-Flop Excitation Tables

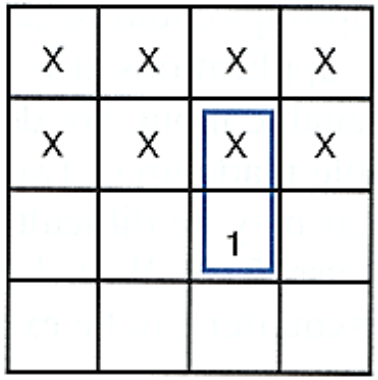
(a) *JK* Flip-Flop

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

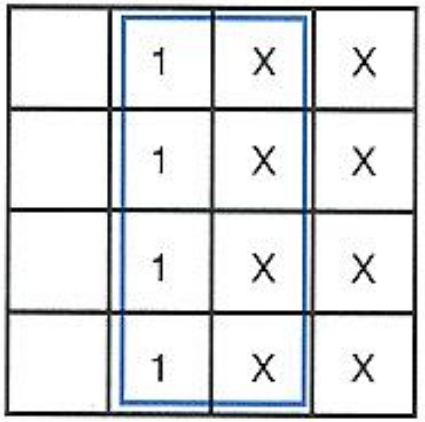
Present state				Next state				Flip-flop inputs							
Q_3	Q_2	Q_1	Q_0	Q_3	Q_2	Q_1	Q_0	J_{Q3}	K_{Q3}	J_{Q2}	K_{Q2}	J_{Q1}	K_{Q1}	J_{Q0}	K_{Q0}
0	0	0	0	0	0	0	1	0	×	0	×	0	×	1	×
0	0	0	1	0	0	1	0	0	×	0	×	1	×	×	1
0	0	1	0	0	0	1	1	0	×	0	×	×	0	1	×
0	0	1	1	0	1	0	0	0	×	1	×	×	1	×	1
0	1	0	0	0	1	0	1	0	×	×	0	0	×	1	×
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	1	0	1	0	×	0	0	×	1	×	×	1
1	0	1	0	1	0	1	1	×	0	0	×	×	0	1	×
1	0	1	1	1	1	0	0	×	0	1	×	×	1	×	1
1	1	0	0	1	1	0	1	×	0	×	0	0	×	1	×
1	1	0	1	1	1	1	0	×	0	×	0	1	×	×	1
1	1	1	0	1	1	1	1	×	0	×	0	×	0	1	×
1	1	1	1	0	0	0	0	×	1	×	1	×	1	×	1



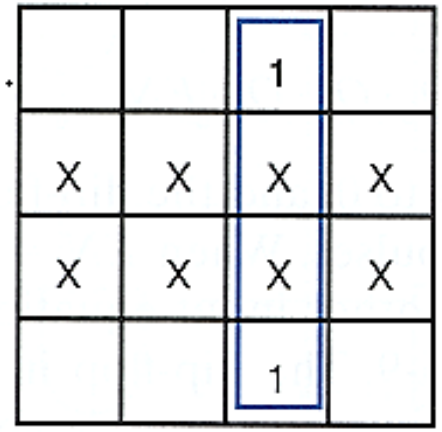
$J_{Q3} = Q_0 Q_1 Q_2$



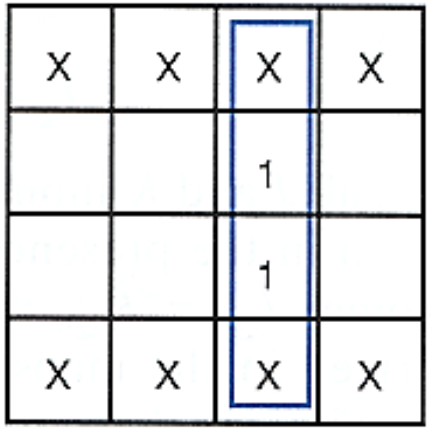
$K_{Q3} = Q_0 Q_1 Q_2$



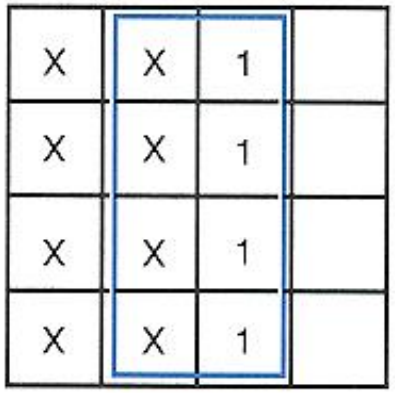
$J_{Q1} = Q_0$



$J_{Q2} = Q_0 Q_1$

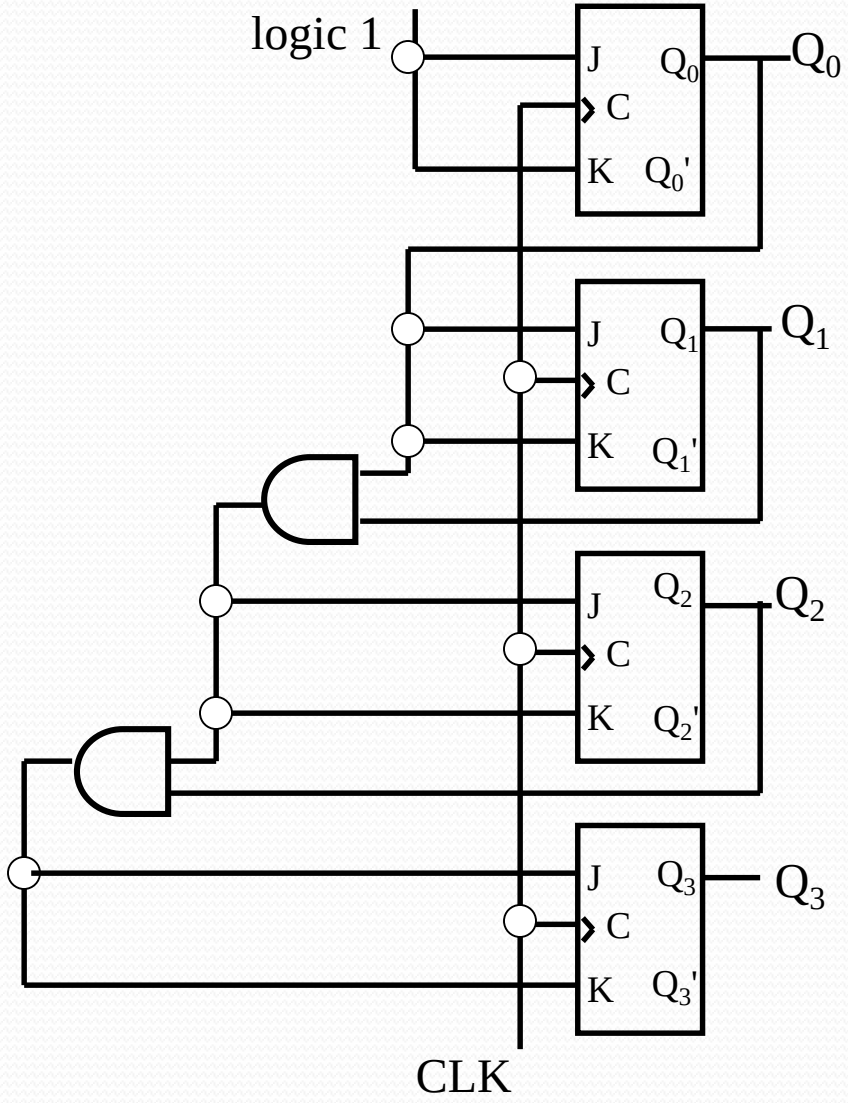


$K_{Q2} = Q_0 Q_1$



$K_{Q1} = Q_0$

J-K FF Design of a 4- Bit Positive Edge Triggered Synchronous Binary Up Counter

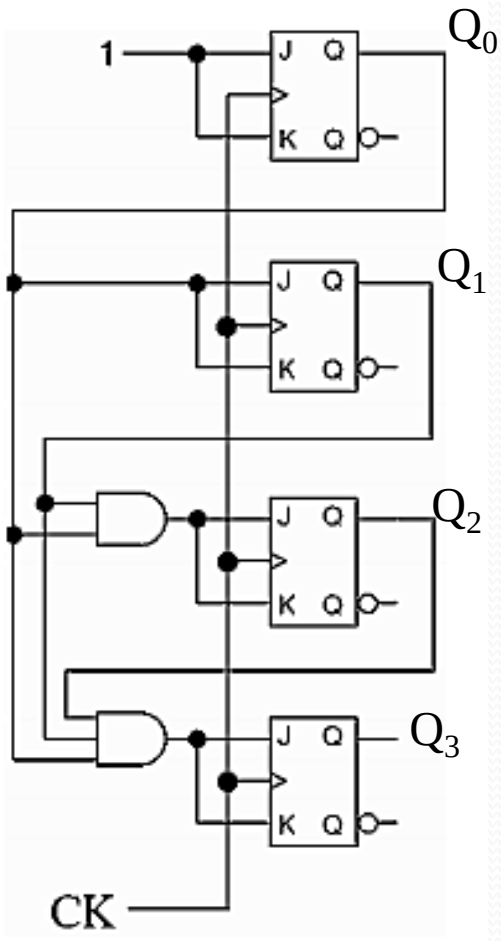


$J_{Q0} = 1$
$K_{Q0} = 1$

$J_{Q1} = Q_0$
$K_{Q1} = Q_0$

$J_{Q2} = Q_0\,Q_1$
$K_{Q2} = Q_0\,Q_1$

$J_{Q3} = Q_0\,Q_1\,Q_2$
$K_{Q3} = Q_0\,Q_1\,Q_2$

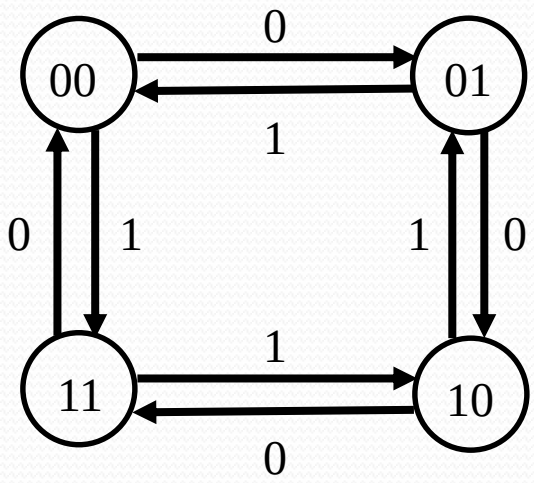


Binary Up Counter

Synchronous Binary Up/Down Counter

Example: 2-bit Up/Down counter using D flip-flop.

- Counter outputs will be 00, 01, 10 and 11
- There is a single input, X.
  - X= 0, the counter counts up
  - X= 1, the counter counts down



Present State		Inputs	Next State	
Q_1	Q_0	X	Q_1	Q_0
0	0	0	0	1
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	0

$$D_1 = Q_1 \oplus Q_0 \oplus X$$

$$D_0 = Q_0'$$

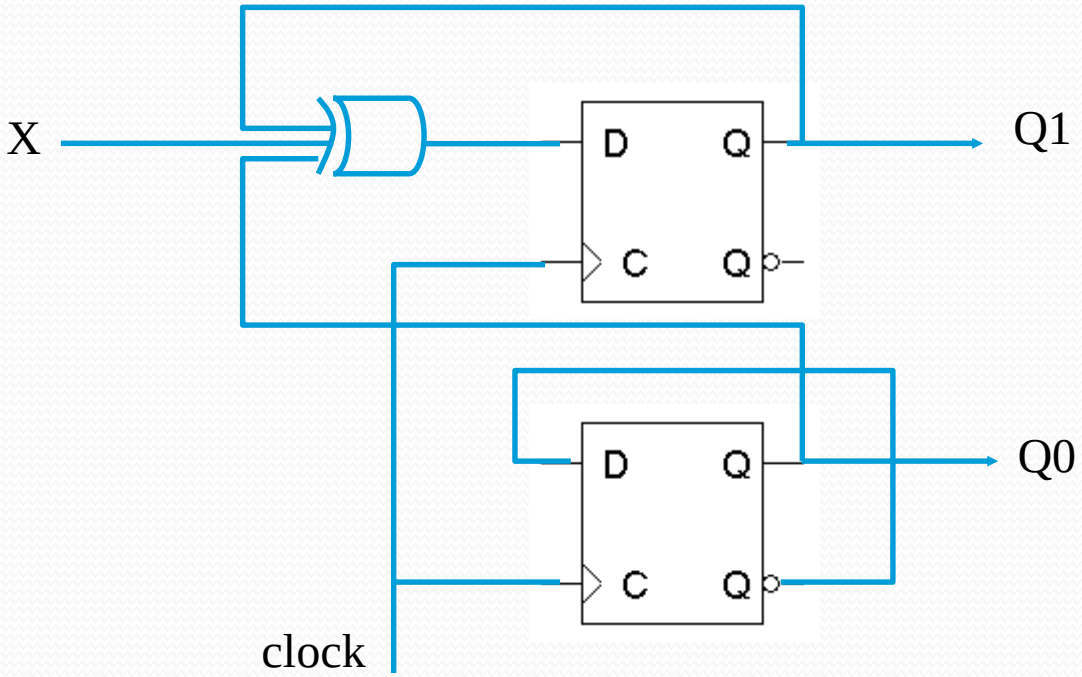
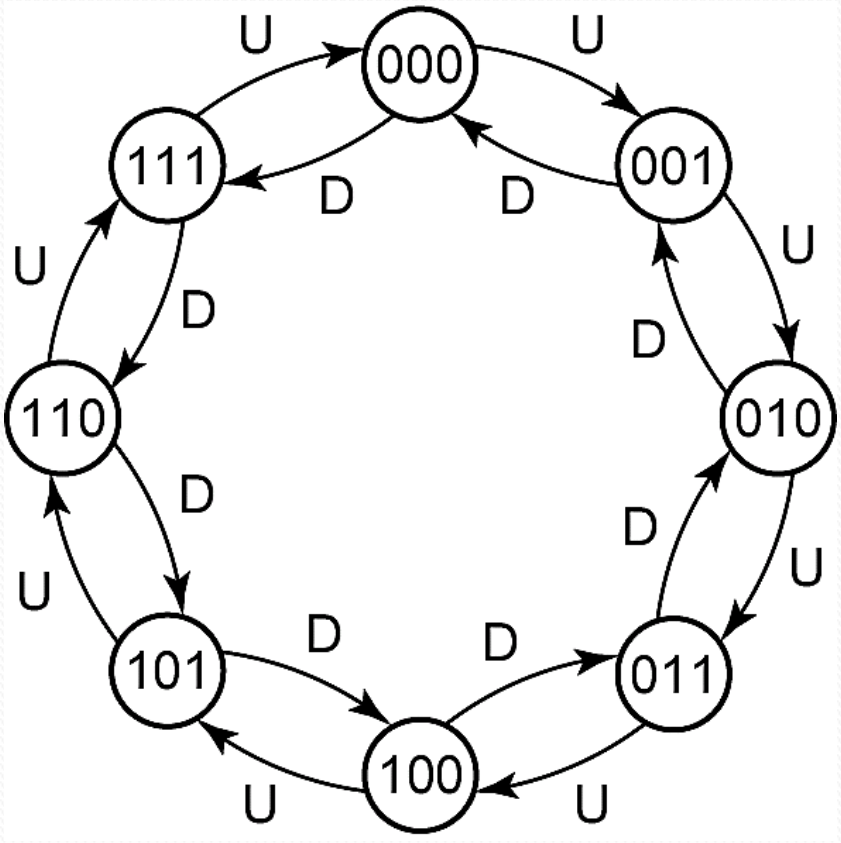


Fig: 2-bit Positive Edge Triggered Synchronous Up/Down counter using D flip-flop

Now try using
JK F/F and T F/F

Binary Up/Down Counter



State Graph and Table for Up-Down Counter

CBA	$C^+B^+A^+$	
	U	D
000	001	111
001	010	000
010	011	001
011	100	010
100	101	011
101	110	100
110	111	101
111	000	110

Synchronous BCD or Decade Counter using T Flip-flops

An output *y* is also shown in the table. This output is equal to 1 when the counter present state is 1001. In this way, *y* can enable the count of the next-higher-order decade while the same pulse switches the present decade from 1001 to 0000.

Present State				Next State				Output	Flip-Flop Inputs			
Q_8	Q_4	Q_2	Q_1	Q_8	Q_4	Q_2	Q_1	y	TQ_8	TQ_4	TQ_2	TQ_1
0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	1	0	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	0	1
0	0	1	1	0	1	0	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	0	1
0	1	0	1	0	1	1	0	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	0	1
0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	0	1
1	0	0	1	0	0	0	0	1	1	0	0	1

The unused states for minterms 10 to 15 are taken as don't-care terms. The simplified functions are:

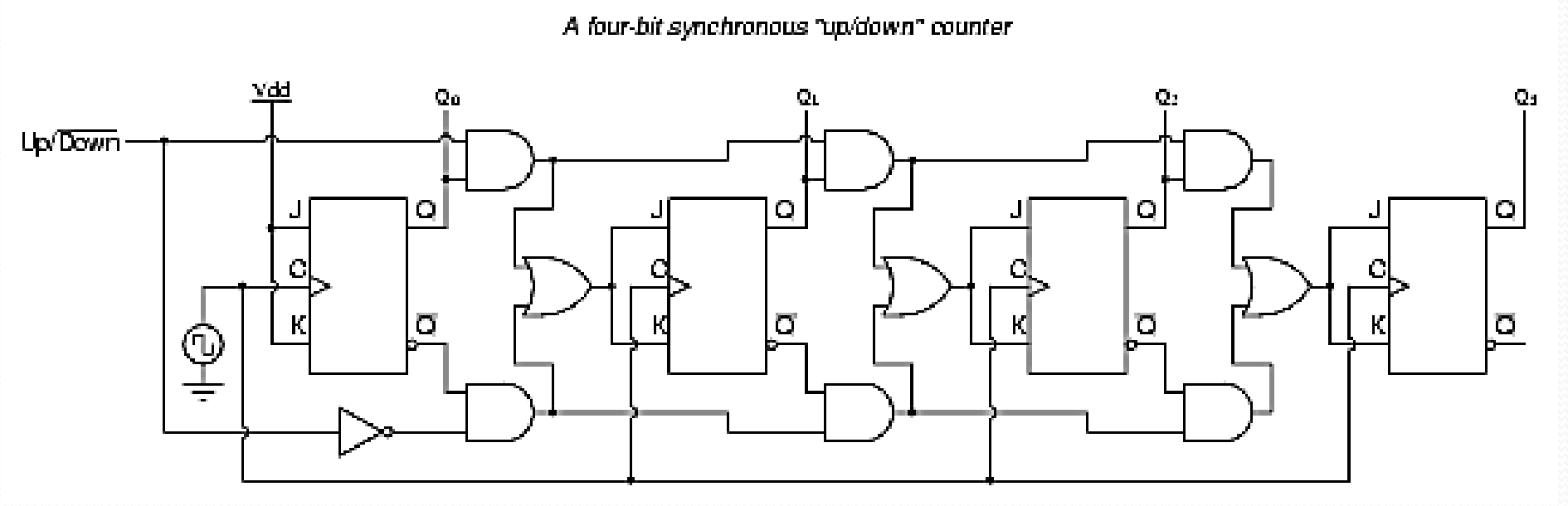
$$TQ_1 = 1$$
$$TQ_2 = Q_8'Q_1$$
$$TQ_4 = Q_2Q_1$$
$$TQ_8 = Q_8Q_1 + Q_4Q_2Q_1$$
$$y = Q_8Q_1$$

Draw the diagram yourself.

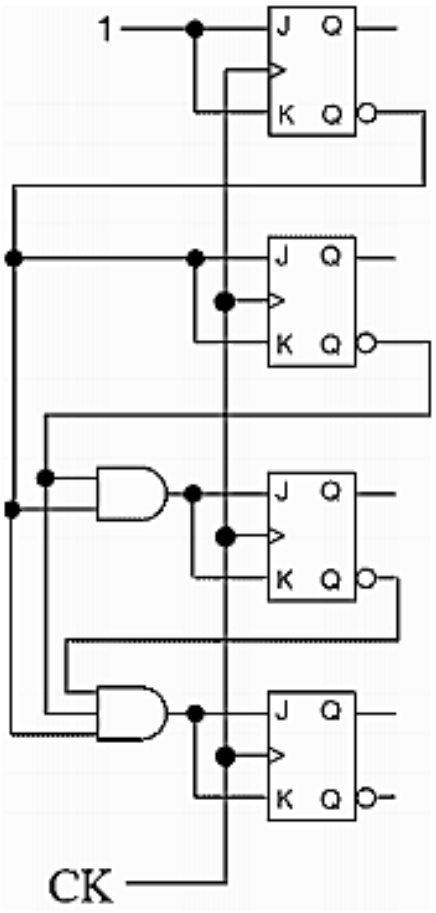
Assignment:

- Design of a 4- Bit Synchronous Binary Down Counter with timing diagram.
- Design the 4 bit Synchronous up/down counter with timing diagram, logic diagram and truth table.
- Design a mod-6 synchronous counter using T Flip-flops.

4- Bit Positive Edge Triggered Synchronous Binary Up/Down Counter



4- Bit Positive Edge Triggered Synchronous Binary Down Counter



Binary Down Counter

Timing Sequences

- The sequences of operations in a digital system are specified by a control unit. The control unit that supervises the operations in a digital system would normally consist of timing signals that determine the time sequence in which the operations are executed. The timing sequences in the control unit can be easily generated by means of counters or shift registers.

Timing Signals

- In a parallel mode of operation, a single clock pulse can specify the time at which an operation should be executed. The control unit in a digital system that operates in the parallel mode must generate timing signals that stay on for only one clock pulse period, but these timing signals must be distinguished from each other.
- Timing signals that control the sequence of operations in a digital system can be generated with a shift register or a counter with a decoder.

Word-Time Generation

- The control unit in a serial computer must generate a *word-time* signal that stays on for a number of pulses equal to the number of bits in the shift registers. The word-time signal can be generated by means of a counter that counts the required number of pulses.

Example:

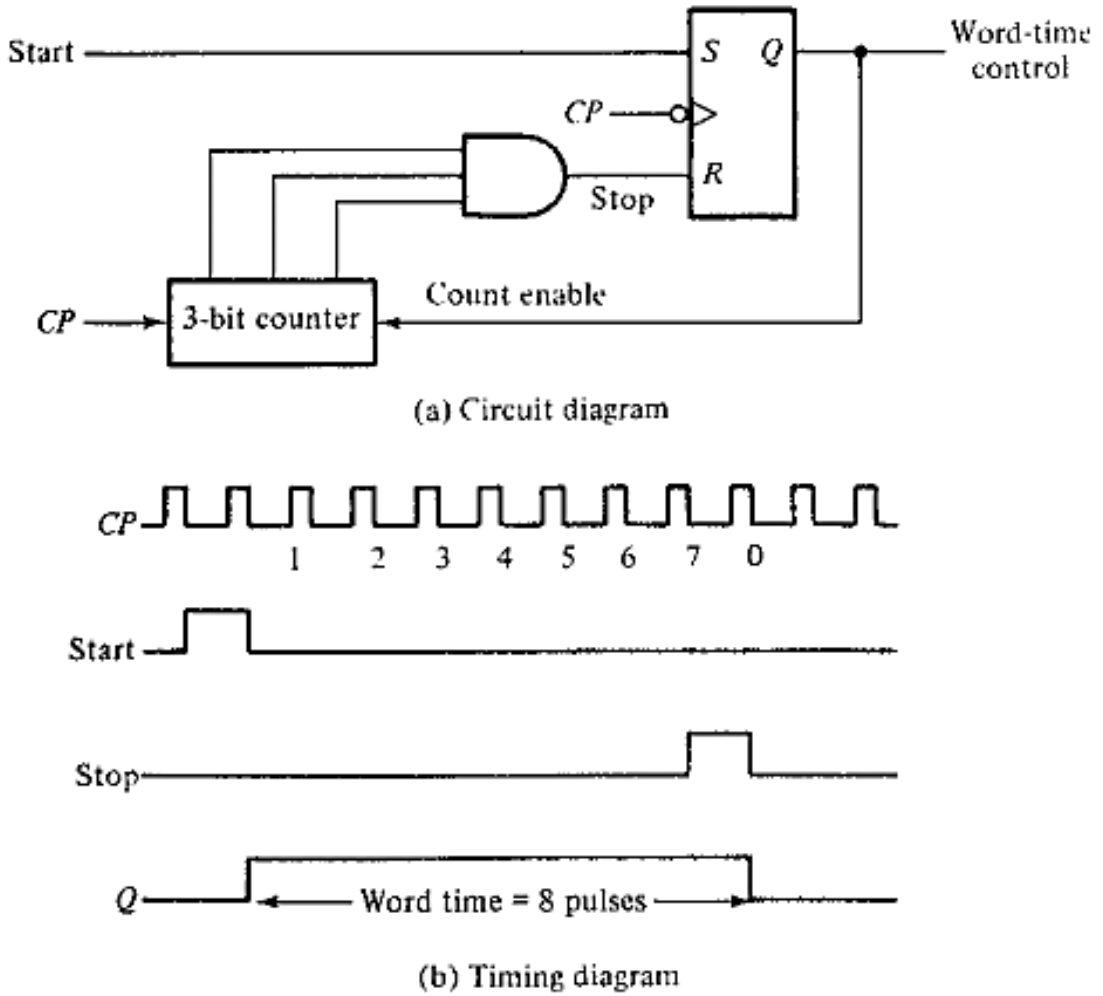


FIGURE 7-21
Generation of a word-time control for serial operations

Ring Counter

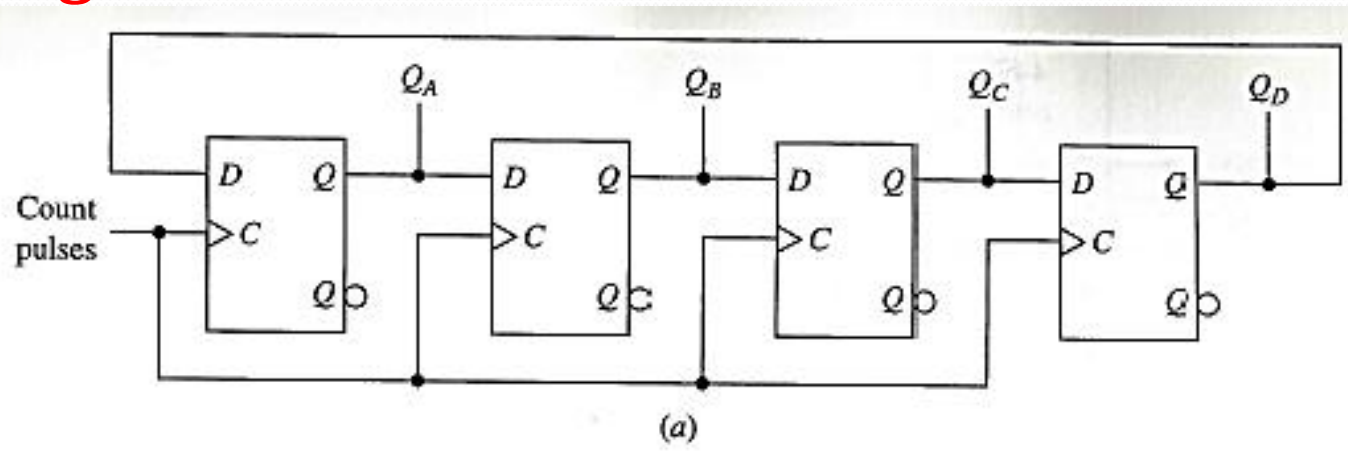
- A *ring counter* is a circular shift register with only one flip-flop being set at any particular time; all others are cleared. The single bit is shifted from one flip-flop to the other to produce the sequence of timing signals.

Johnson Counter

- A **Johnson counter** is a k-bit **switch-tail/twisted ring counter** with *2k* decoding gates to provide outputs for *2k* timing signals.

Counters Based on Shift Registers

Ring Counter

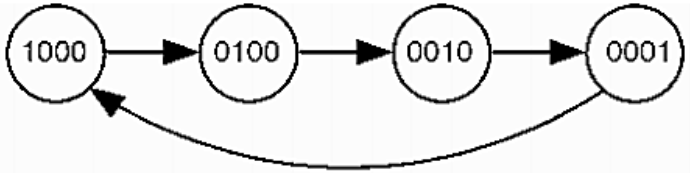


Q_A	Q_B	Q_C	Q_D
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
1	0	0	0

etc.

(b)

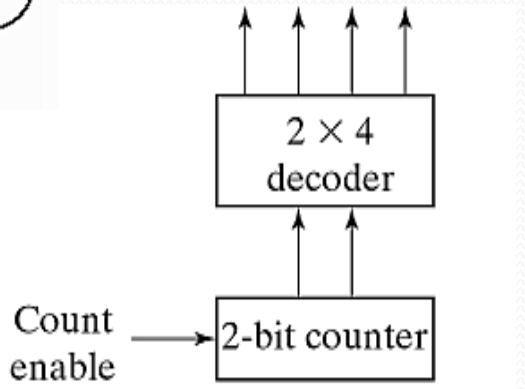
Figure 6.37 Mod-4 ring counter. (*a*) Logic diagram. (*b*) Counting sequence.



Note:

The sequence is 1000, 0100, 0010, 0001 provided that one of the given patterns is forced as initial state. (by loading or set/reset)

NOTE: with 4 FF we make only 4 patterns



(b) Counter and decoder

Counters Based on Shift Registers

Johnson(Moebius/Twisted-ring) Counter

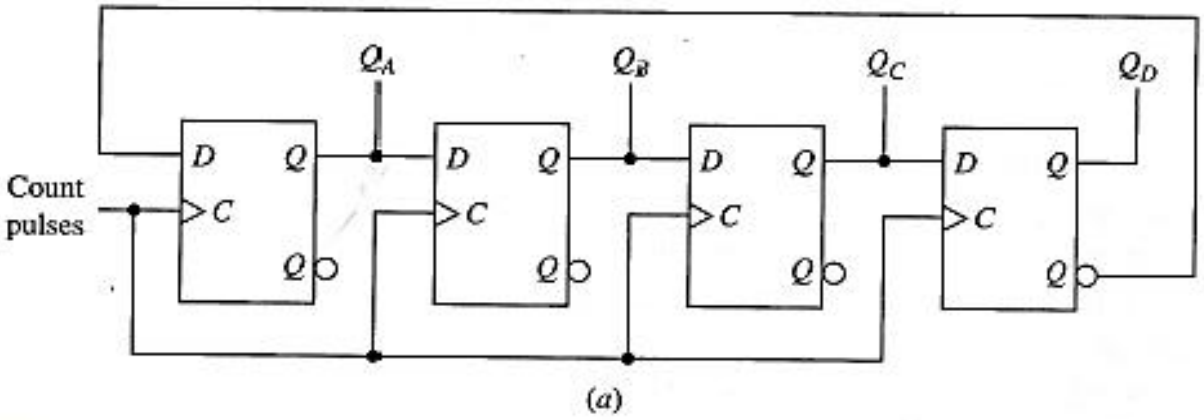
How does this counter work?

we can use 0000 or 1111 as reset state

- Counts through the sequence:
  1000, 1100, 1110, 1111, 0111, 0011, 0001, 0000

NOTE: with 4 FF we make 8 patterns.

Number of states of a ring counter can be doubled



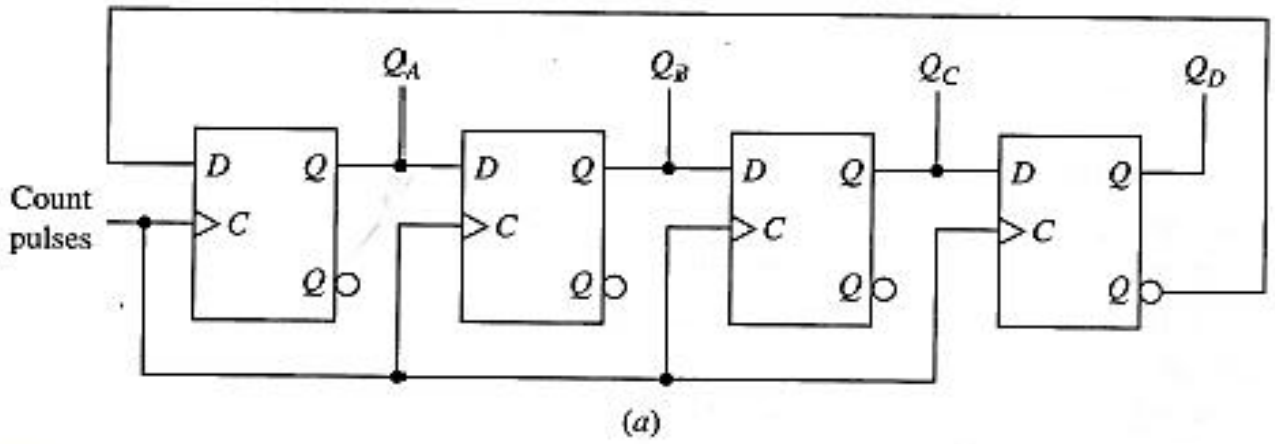
(a)

Q_A	Q_B	Q_C	Q_D	And-gate inputs
0	0	0	0	$\bar{Q}_A\bar{Q}_D$
1	0	0	0	$Q_A\bar{Q}_B$
1	1	0	0	$Q_B\bar{Q}_C$
1	1	1	0	$Q_C\bar{Q}_D$
1	1	1	1	Q_AQ_D
0	1	1	1	$\bar{Q}_AQ_B$
0	0	1	1	$Q_B\bar{Q}_C$
0	0	0	1	$\bar{Q}_CQ_D$
0	0	0	0	

etc.

(b)

Figure 6.38 Mod-8 twisted-ring counter. (a) Logic diagram. (b) Counting sequence.



Q_A	Q_B	Q_C	Q_D	And-gate inputs
0	0	0	0	$\bar{Q}_A\bar{Q}_D$
1	0	0	0	$Q_A\bar{Q}_B$
1	1	0	0	$Q_B\bar{Q}_C$
1	1	1	0	$Q_C\bar{Q}_D$
1	1	1	1	Q_AQ_D
0	1	1	1	$\bar{Q}_AQ_B$
0	0	1	1	$\bar{Q}_BQ_C$
0	0	0	1	$\bar{Q}_CQ_D$
0	0	0	0	

etc.

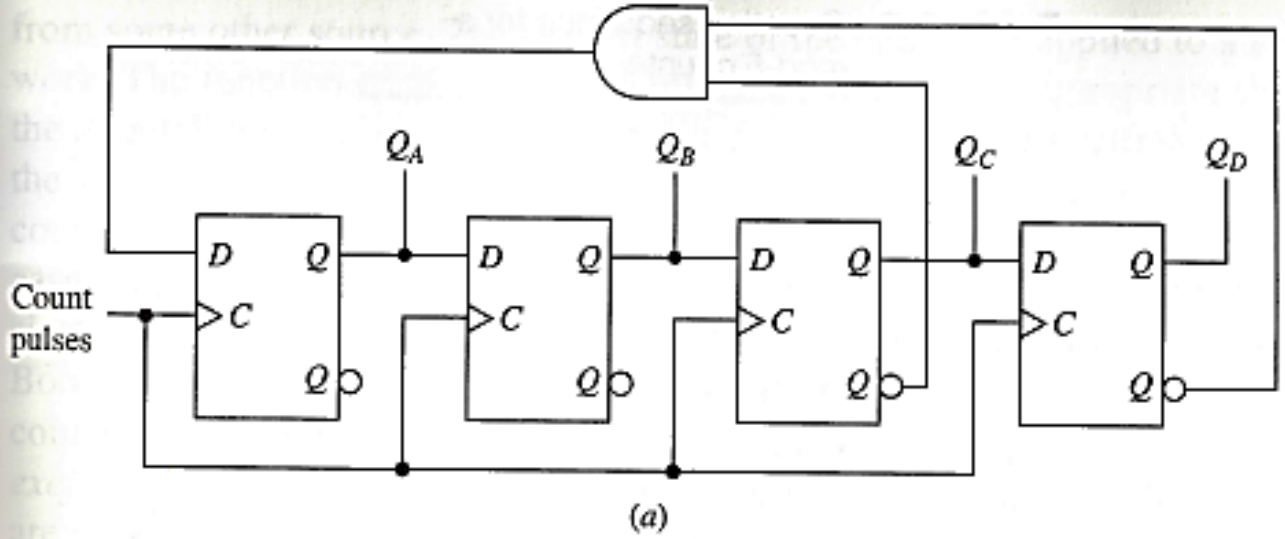
(b)

Figure 6.38 Mod-8 tw sted-ring counter. (*a*) Logic diagram. (*b*) Counting sequence.

The eight AND gates listed in the table, when connected to the circuit will complete the construction of the Johnson counter. Since each gate is enabled during one particular state sequence, the outputs of the gates generate eight timing sequences in succession.

Counters Based on Shift Registers

Twisted ring counter for odd number of states



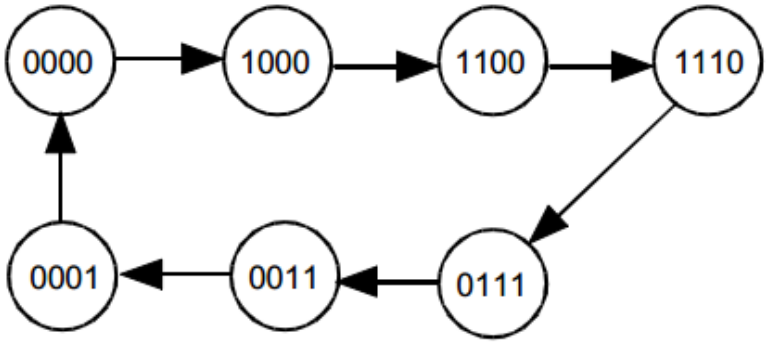
(a)

Q_A	Q_B	Q_C	Q_D	And-gate inputs
0	0	0	0	$\bar{Q}_A\bar{Q}_D$
1	0	0	0	$Q_A\bar{Q}_B$
1	1	0	0	$Q_B\bar{Q}_C$
1	1	1	0	$Q_C\bar{Q}_D$
0	1	1	1	$\bar{Q}_AQ_B$
0	0	1	1	$\bar{Q}_BQ_C$
0	0	0	1	$\bar{Q}_CQ_D$
0	0	0	0	

etc.

(b)

Figure 6.39 Mod-7 twisted-ring counter. (*a*) Logic diagram. (*b*) Counting sequence.



Memory unit (Random Access Memory-RAM)

- A memory unit is a collection of storage cells together with associated circuits needed to transfer information in and out of the device. Memory cells can be accessed for information transfer to or from any desired random location and hence the name *random access memory*, abbreviated RAM.
- A memory unit stores binary information in groups of bits called ***words.*** A word in memory is an entity of bits that move in and out of storage as a unit. A memory word is a group of 1's and 0's and may represent a number, an instruction, one or more alphanumeric characters, or any other binary-coded information.

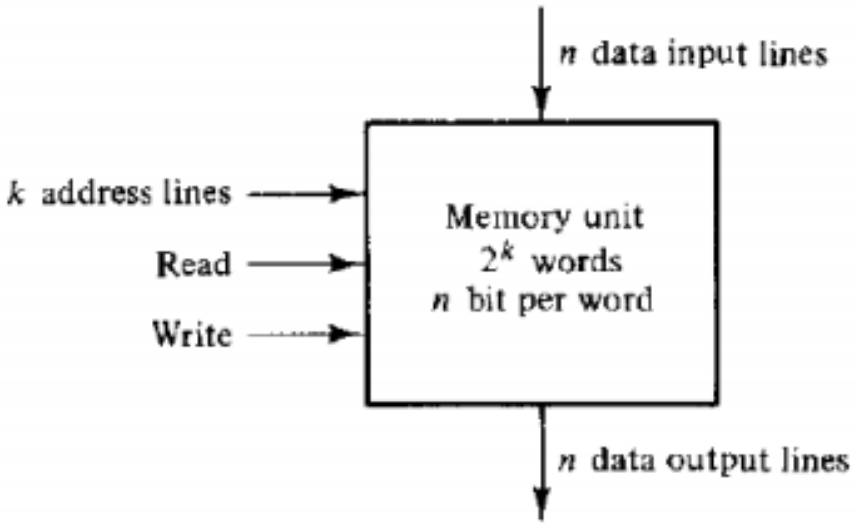


Fig: Block Diagram of a memory unit

- The communication between a memory and its environment is achieved through:
  - *n* data input lines : provide information to be stored in memory
  - ***n* data output lines**: supply the information coming out of memory.
  - ***k* address lines:** specify particular word chosen among the many available.
  - **two control inputs**: specify the direction of transfer desired(Read or Write).
- Each word in memory is assigned an identification number, called an address, starting from 0 and continuing with 1, 2, 3, up to 2k - I, where k is the number of address lines.
- Computer memories may range from 1024 words, requiring an address of 10 bits, to 2^{32} words, requiring 32 address bits.

Write and Read Operations

The two operations that a random-access memory can perform are the write and read operations. The write signal specifies a transfer-in operation and the read signal specifies a transfer-out operation. On accepting one of these control signals, the internal circuits inside the memory provide the desired function.

Write Operation: transferring a new word to be stored into memory

1. Transfer the binary address of the desired word to the address lines.
2. Transfer the data bits that must be stored in memory to the data input lines.
3. Activate the *write* input.

Read Operation: transferring a stored word out of memory.

1. Transfer the binary address of the desired word to the address lines.
2. Activate the read input.