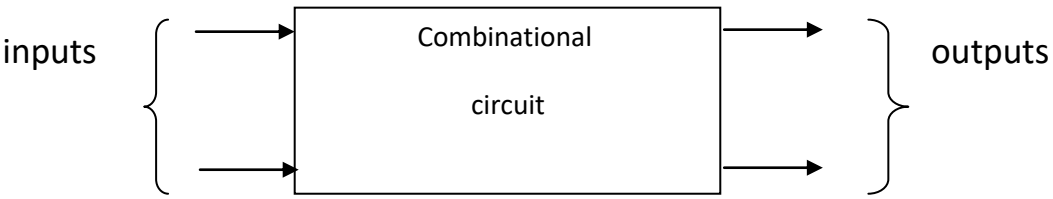
Chapter four

Combinational circuits

Combinational circuits:

A combinational circuit consists of logic gates whose output at any time are determined directly from the values of the present inputs.



Design procedure:

The procedure involves the following steps:

1-Determine the required number of inputs and outputs.

2-Drive the truth table.

3-Obtain the simplified Boolean function for each output.

4-Draw the logic diagram.

Half adder (H.A.):

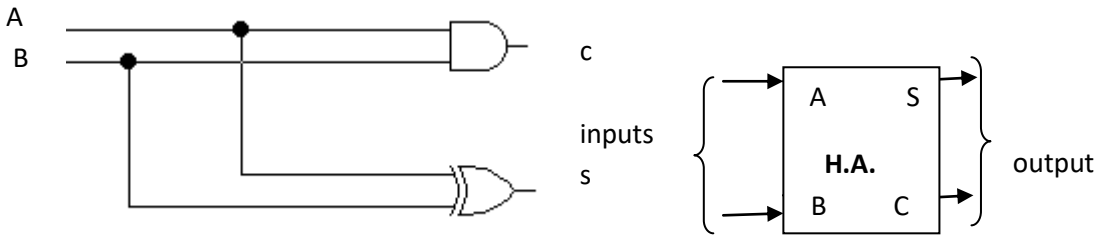
To design a circuit that adds two numbers each of one bit for example:

Two inputs	1	0	1	0
	1 +	0 +	0 +	1 +
Two output	1 0	0	1	1

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = \overline{A}\,B + A\,\overline{B} = A \oplus B$

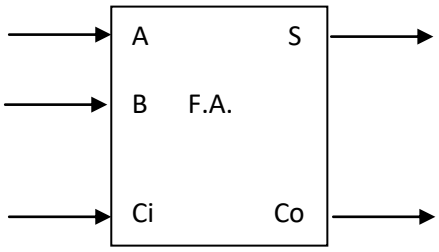
$C = A\,B$



Full adder (F. A):

في حالة جمع عددين كل واحد مكون من 2-bit أي لتصميم دائرة تجمع ال bits في المرحلة الثانية و ما بعدها:

C_i	1	
A	0	1
B	1	1 +
1	0	0
Carry	sum	



A	B	Ci	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$S = \overline{A}\,\overline{B}\,Ci + \overline{A}\,B\,\overline{Ci} + A\,\overline{B}\,\overline{Ci} + A\,B\,Ci$

$S = Ci(\overline{A}\,\overline{B} + A\,B) + \overline{Ci}(\overline{A}\,B + A\,\overline{B})$

$S = Ci\,\overline{(A \oplus B)} + \overline{Ci}\,(A \oplus B)$

Let $A \oplus B = K$

$S = Ci\,\overline{K} + \overline{Ci}\,K = Ci \oplus K$

$S = Ci \oplus A \oplus B$

$Co = \overline{A}\,B\,Ci + A\,\overline{B}\,Ci + A\,B\,\overline{Ci} + A\,B\,Ci$

$Co = Ci\,(\overline{A}B + A\overline{B}) + AB\,(\overline{Ci} + Ci)$

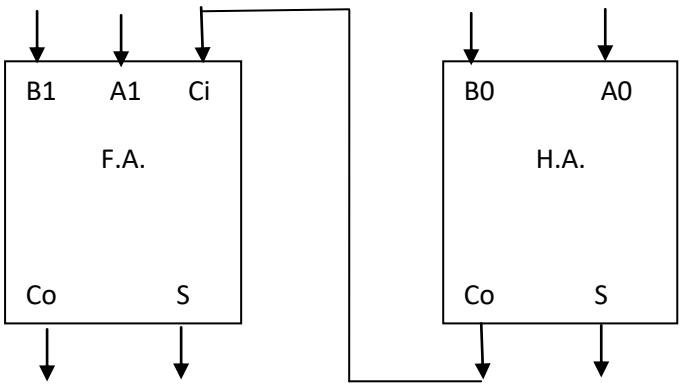
$Co = Ci\,.\,(A \oplus B) + AB$

Parallel binary adder:

لجمع رقمين كل منهما مكون من 2-bit

Ex: Design a logic circuit that add $(3 + 1)_D$:

1	Ci
1 1	A1 A0
0 1 +	B1 B0 +
1 0 0	Co S1 S0
0 1 1	1 1



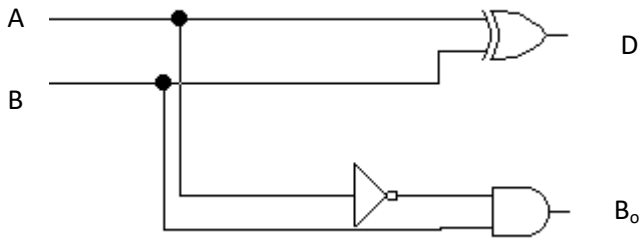
1 0 1 0

Half subtractor (H.S.):

A	B	Bo	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

$D = \overline{A}\,B + A\,\overline{B} = A \oplus B$

$Bo = \overline{A}$



Full subtractor (F.S):

A	B	Bi	Bo	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

$D = \overline{A}\,\overline{B}\,Bi + \overline{A}\,B\,\overline{Bi} + A\,\overline{B}\,\overline{Bi} + A\,B\,Bi$

$D = Bi\,(\overline{A}\,\overline{B} + A\,B) + \overline{Bi}\,(\overline{A}\,B + A\,\overline{B})$

$D = Bi\,(\overline{A \oplus B}) + \overline{Bi}\,(A \oplus B)$

$D = A \oplus B \oplus Bi$

$Bo = \overline{A}\,\overline{B}\,Bi + \overline{A}\,B\,\overline{Bi} + \overline{A}\,B\,Bi + A\,B\,Bi$

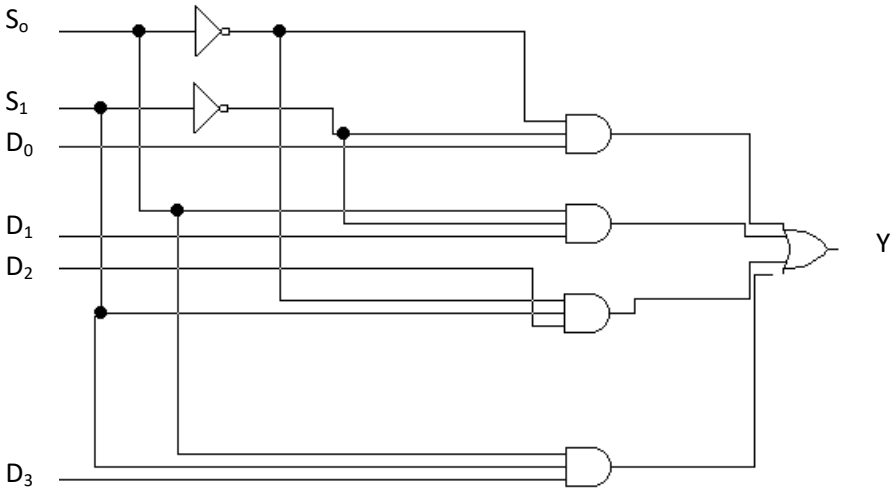
$Bo = Bi\,(\overline{A}\,\overline{B} + A\,B) + \overline{A}\,B\,(\overline{Bi} + Bi)$

$Bo = Bi\,\overline{(A \oplus B)} + \overline{A}\,B$

Multiplexers (MUX):

Multiplexer is a combinational circuit that selects one of many input lines and direct it to a single output line. The selection of a particular input line is controlled by a set of select variables. Normally there are (2^n) input lines and (n) select variables whose bit combination determines which input is selected. The 4-to-1 multiplexer is shown below:

S1	S0	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3



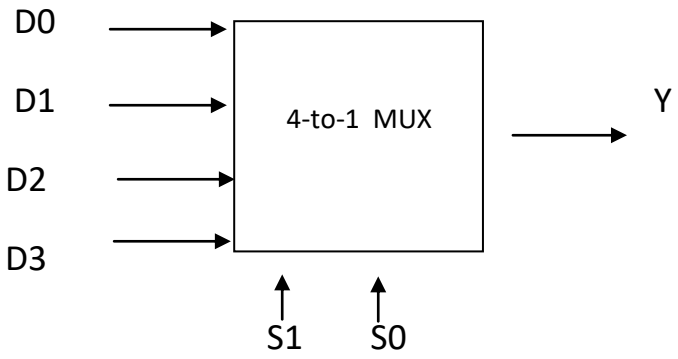
There are: 2-to-1 MUX with 1 select variable

4-to-1 MUX with 2 select variable.

8-to-1 MUX with 3 select variable.

. 16-to-1 MUX with 4 select variable

The circuit above can be implemented as an MSI chip, such a chip has four data inputs, two select variables and one output.



Boolean function implementation using MUX :

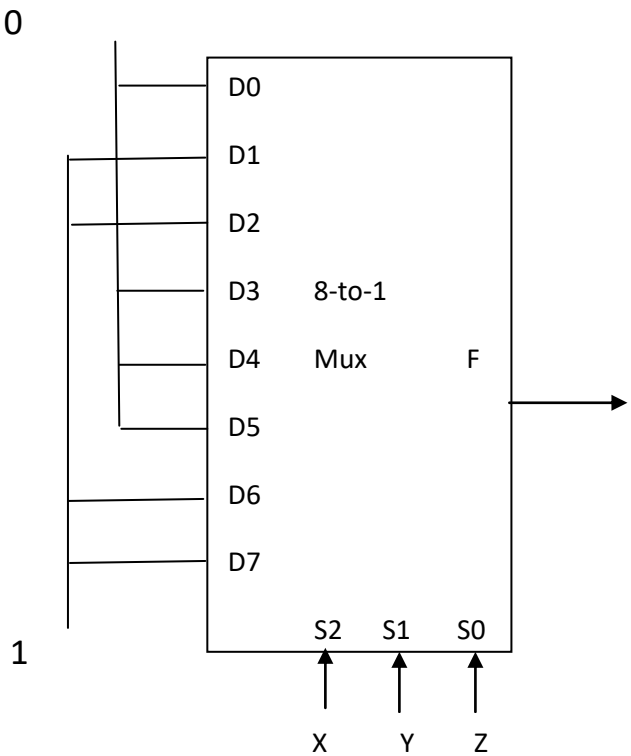
Boolean function of (n)variables can be implemented with a multiplexer of either n, n-1, n-2…… select variables.

Ex: Implement the following function with (8-to1) and (4-to-1) MUX:

F(X,Y,Z)= ∑(1,2,6,7).

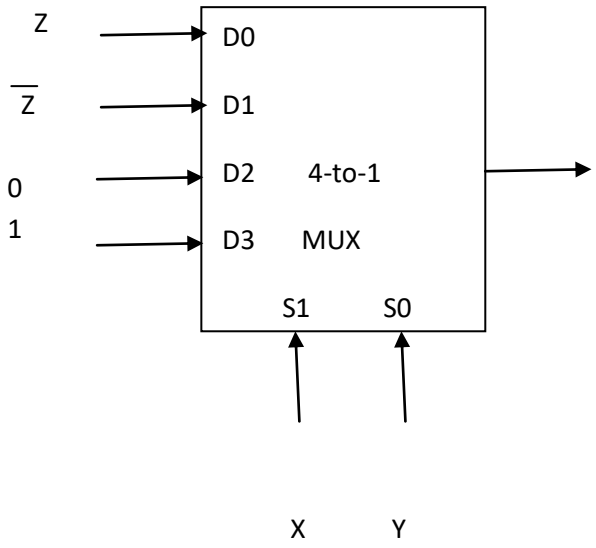
First: by using (8-t0-1) MUX:

S_2 X	S_1 Y	S_0 Z	F	
0	0	0	0	D0=0
0	0	1	1	D1=1
0	1	0	1	D2=1
0	1	1	0	D3=0
1	0	0	0	D4=0
1	0	1	0	D5=0
1	1	0	1	D6=1
1	1	1	1	D7= 1



Second: by using (4-t0-1) MUX:

X	Y	Z	F	
0	0	0	0	
0	0	1	1	D0=Z
0	1	0	1	
O	1	1	0	D1=$\overline{Z}$
1	0	0	0	
1	0	1	0	D2=0
1	1	0	1	
1	1	1	1	D3=1

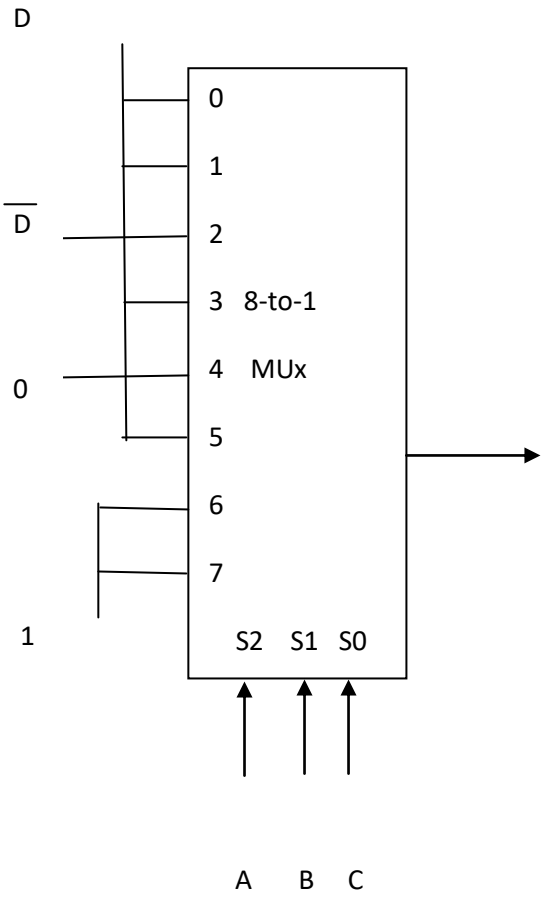


***EX*:** Implement the following function using (8-to-1) and (4-to-1) MUX:

F(A,B,C,D)=∑(1, 3, 4, 7, 11, 12, 13, 14, 15).

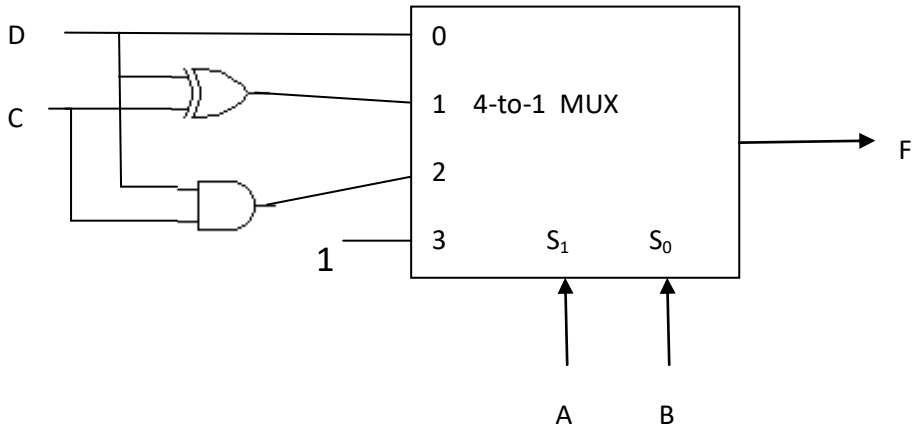
First: using (8-to-1) MUX:

A	B	C	D	F	
0	0	0	0	0	
O	0	0	1	1	D0= D
0	0	1	0	0	
0	0	1	1	1	D1= D
0	1	0	0	1	
0	1	0	1	0	D2= $\overline{D}$
0	1	1	0	0	
0	1	1	1	1	D3= D
1	0	0	0	0	
1	0	0	1	0	D4=0
1	0	1	0	0	
1	0	1	1	1	D5= D
1	1	0	0	1	
1	1	0	1	1	D6= 1
1	1	1	0	1	
1	1	1	1	1	D7= 1



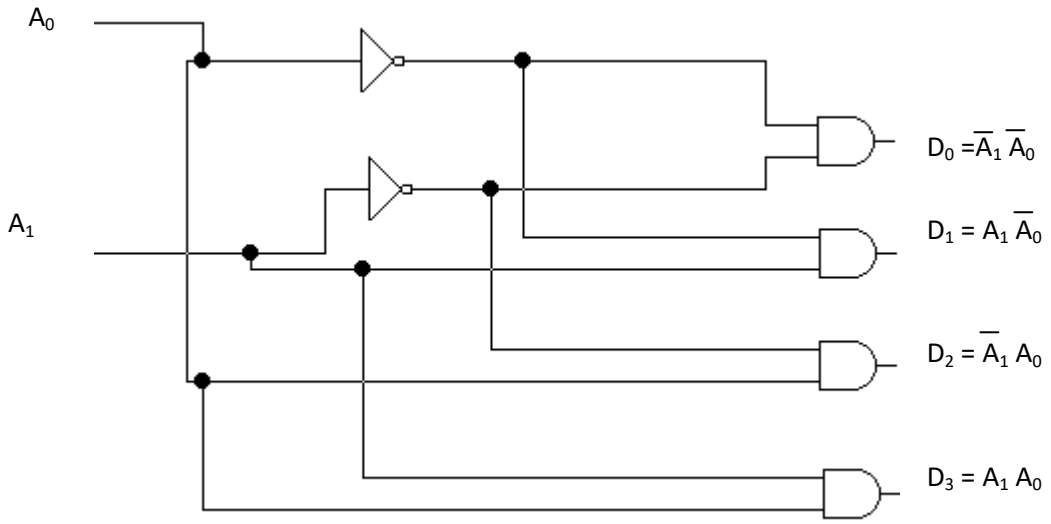
second: Using (4-to-1) MUX:

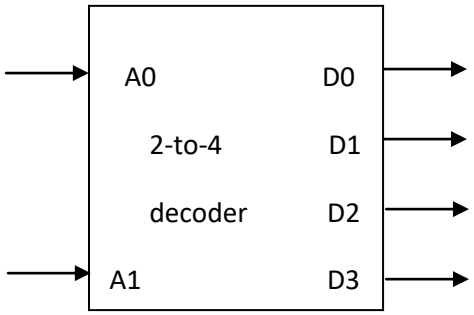
A	B	C	D	F	
0	0	0	0	0	
O	0	0	1	1	
0	0	1	0	0	
0	0	1	1	1	D0= D
0	1	0	0	1	
0	1	0	1	0	
0	1	1	0	0	
0	1	1	1	1	D1= $\overline{C + D}$
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	0	
1	0	1	1	1	D2= C . D
1	1	0	0	1	
1	1	0	1	1	
1	1	1	0	1	
1	1	1	1	1	D3= 1



Decoder :

It is a combinational circuit that converts (n) inputs to a maximum of 2^n unique outputs. A 2-to-4 decoder is shown below:





Boolean function implementation using decoder :

Any combinational circuit with (n) inputs and (m) outputs can be implemented with an n-to- 2^n decoder and (m) OR gates. The Boolean function should be expressed in sum of product.

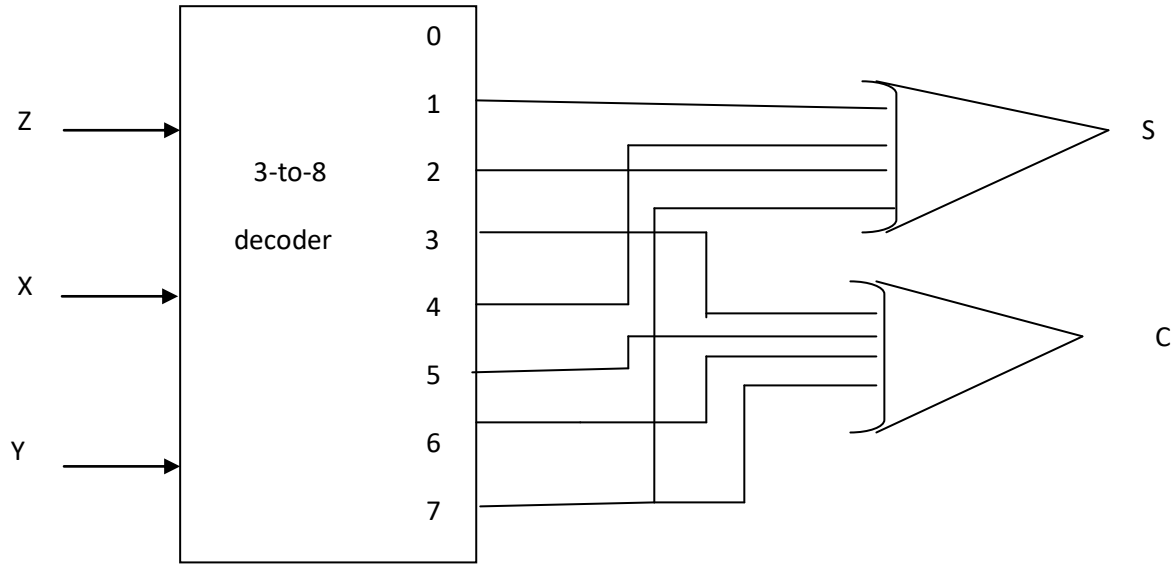
Ex : Implement a full adder function with a decoder and OR gates:

From the truth table of full adder we get:

$S(X,Y,Z) = \sum 1, 2, 4, 7$

$C(X,Y,Z) = \sum 3, 5, 6, 7$

Since there are 3- inputs and a total of 8 minterms, then we need 3-to-8 decoder.



Encoder :

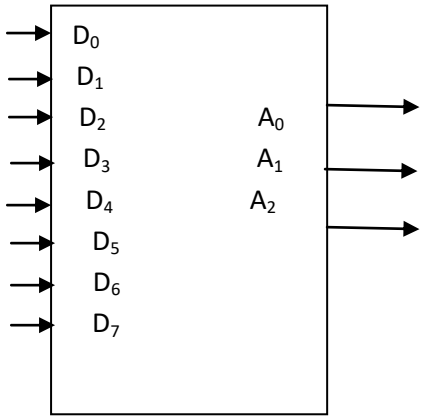
It performs the inverse operation of a decoder. It has 2^n (0r less) inputs and (n) output lines.

It is assumed that only one input has a value of (1) at any given time, otherwise the circuit has no meaning.

For example the 8-to-3 encoder has the following T.T. :

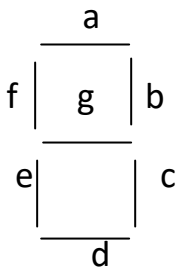
D7	D6	D5	D4	D3	D2	D1	D0	A2	A1	A0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

8-t0-3 encoder

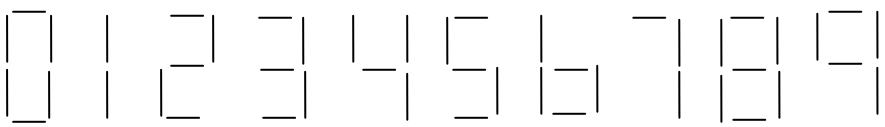


7- segment display :

It consists of seven segments, usually LEDs or liquid crystals.



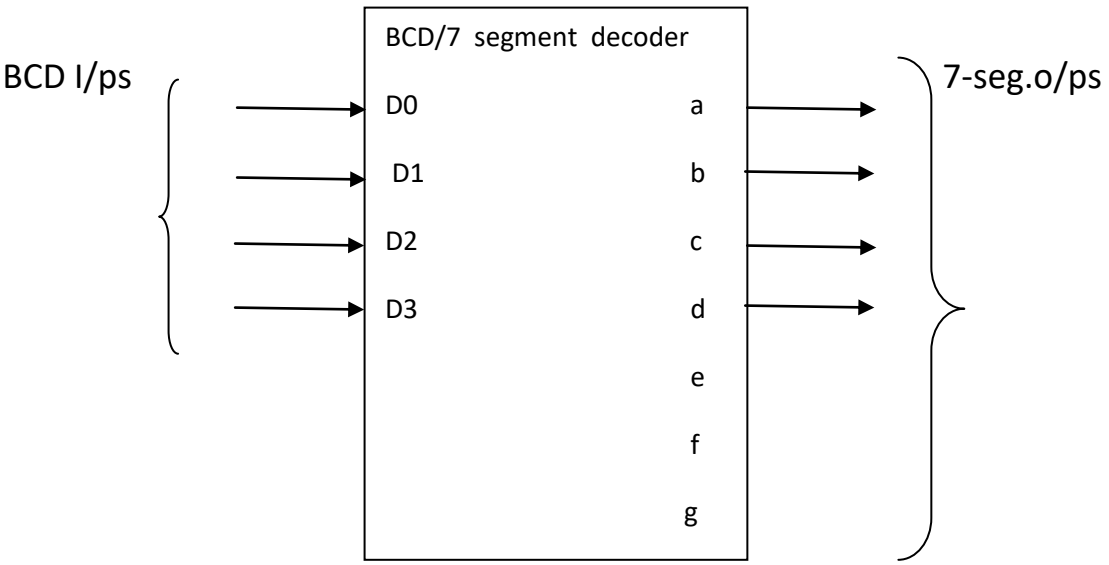
we can display any decimal digit by turning on the appropriate elements (a…..g).



BDC - T0 -7 segment decoder :

It is a circuit with 4- bit input (BCD)and 7-outputs(segments). To display a number, the decoder must translate the input bits to the required output segment:

Digit	D3 D2 D1 D0	a b c d e f g
0	0 0 0 0	1 1 1 1 1 1 0
1	0 0 0 1	0 1 1 0 0 0 0
2	0 0 1 0	1 1 0 1 1 0 1
3	0 0 1 1	1 1 1 1 0 0 1
4	0 1 0 0	0 1 1 0 0 1 1
5	0 1 0 1	1 0 1 1 0 1 1
6	0 1 1 0	0 0 1 1 1 1 1
7	0 1 1 1	1 1 1 0 0 0 0
8	1 0 0 0	1 1 1 1 1 1 1
9	1 0 0 1	1 1 1 0 0 1 1

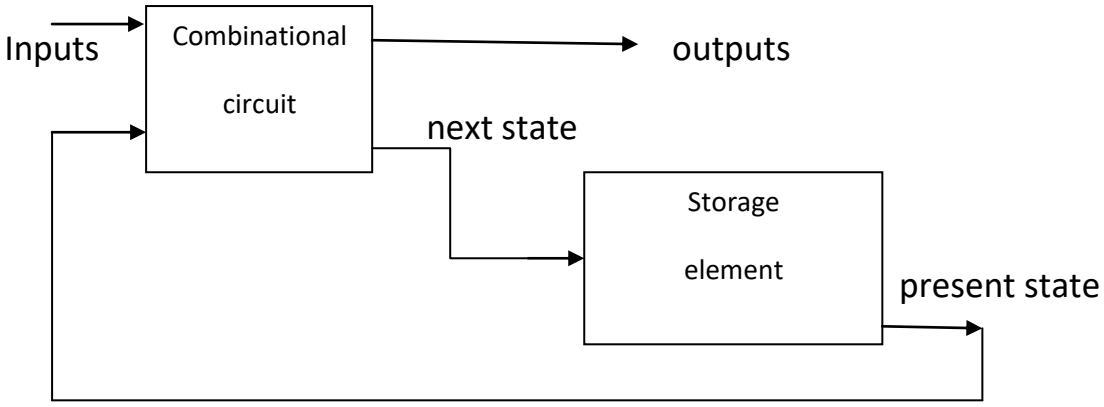


Chapter five

Sequential circuits

Sequential circuits :

The sequential circuits consists of a combinational circuit and storage elements as shown below:



The next state of the storage elements is a function of the inputs and the present state.

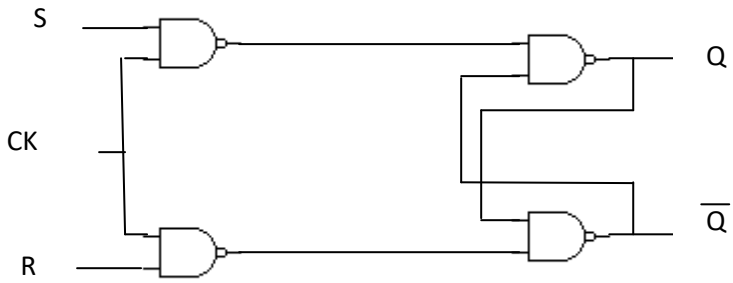
There are two types of sequential circuit:

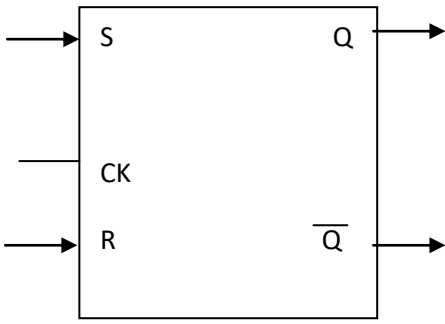
1-Asynchronous: The stored information in the stored element depends on the input signal only.

2-Synchronous: The stored information can change only during the occurrence of a clock pulse.

The storage elements are called flip-flop (f.f.) . There are many types of f.f. :

Set – Reset f.f. (S-R f.f.) :





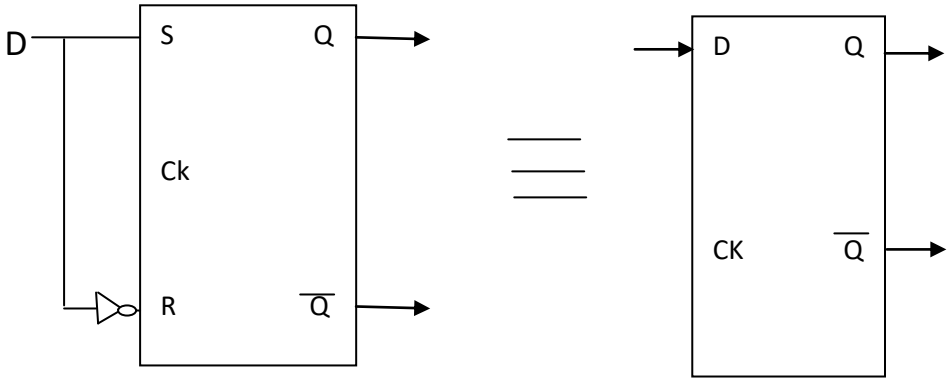
CK	S	R	Q	$\overline{Q}$
0	X	X	X	
1	0	0	no change	
1	0	1	0 (reset)	1
1	1	0	1 (set)	0
1	1	1	not allowed	

Note: Only when ck =1, the information from S and R is allowed to reach the output Q.

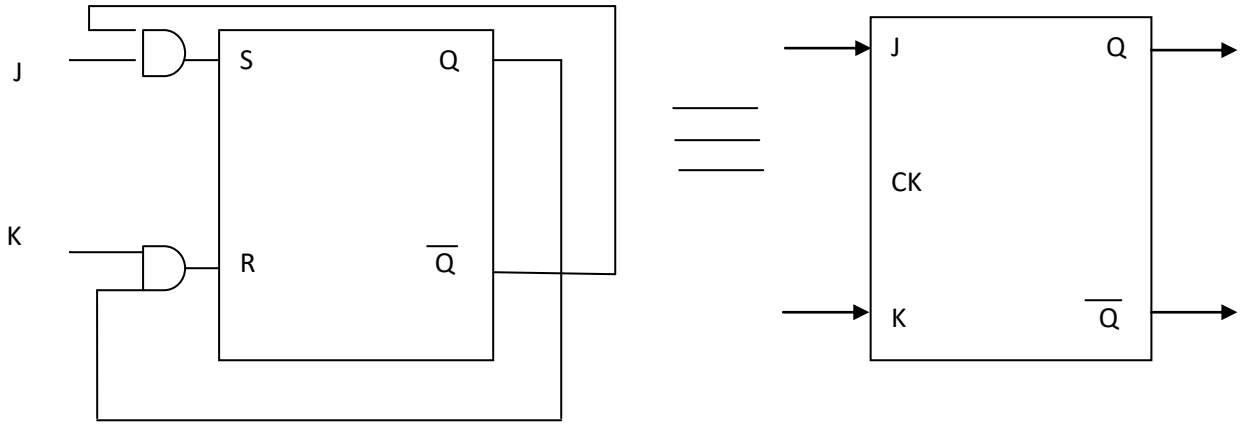
D f.f. :

The only way to eliminate the not *allowed state* in S-R ff is to ensure that both S & R will never be 1 at the same time. This is done in D-ff :

CK	D	Q	$\overline{Q}$
0	X	X	X
1	0	0	1
1	1	1	0



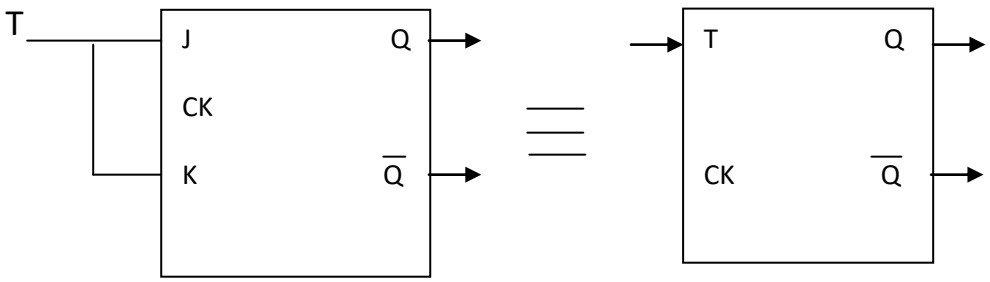
J – K f.f. :



CK	J	K	Q
0	X	X	X
1	0	0	no change
1	0	1	0 (reset)
1	1	0	1(set)
1	1	1	toggle

T – f.f. :

CK	T	Q
0	x	x
1	0	no change
1	1	toggle



EX : Implement the following states on S-R ff. (initial state of $Q\overline{Q}$=10)

S	**R**	Q	Q
0	**1**	0	1
1	**0**	1	0
1	**1**	not allowed	
0	**0**	1	0
1	**0**	1	0

EX : Implement the following states on J-K ff (initial state of QQ=01)

J: 1 0 1 1 0 0

K: 1 0 0 1 1 0

J	k	Q	Q	
1	1	1	0	toggle
0	0	1	0	no change
1	0	1	0	set
1	1	0	1	toggle
0	1	0	1	reset
0	0	0	1	no change

Shift Register :

It is a group of flip-flops that are capable of shifting binary information in one or both directions.

Shift registers are useful in:

1- Storage of serial data.

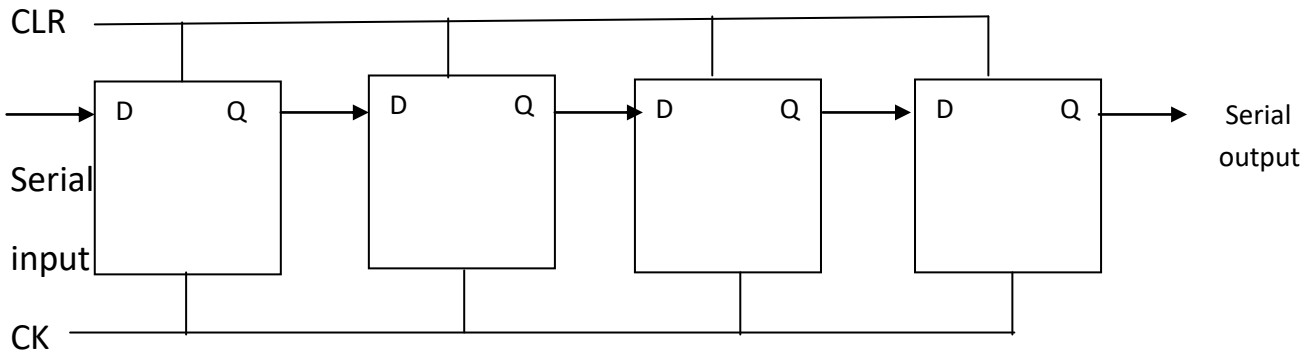
2- Serial to parallel or parallel to serial data conversion.

3- performing arithmetic operations

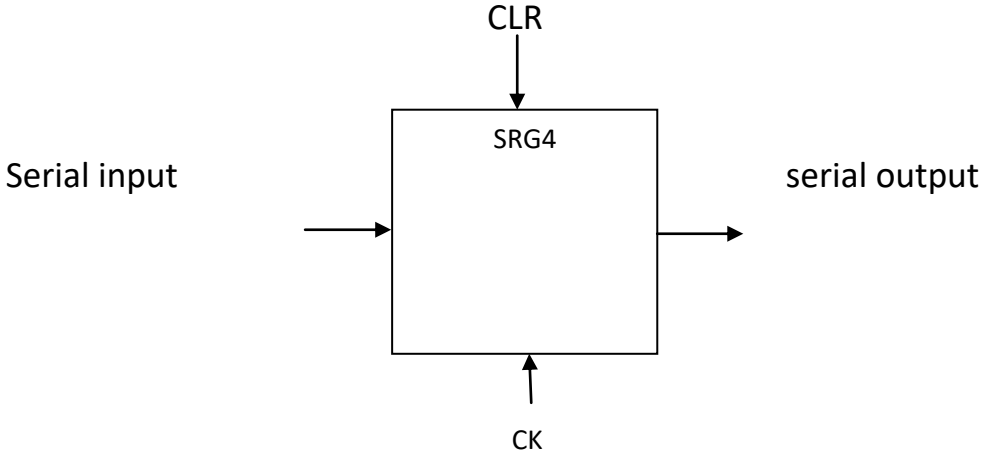
There are two types of data transfer:

1- Serial transfer :

Where data is transferred one bit at a time by shifting the bits of one ff into the next ff and so on.



The serial input determines what goes into left most position during the shift. The serial output is taken from the output of the right most ff. the standard graphic symbol is:



***EX*:** Shift the following data five pulses to the right:(**10111001**)

1	0	1	1	1	0	0	1

0	1	0	1	1	1	0	0

0	0	1	0	1	1	1	0

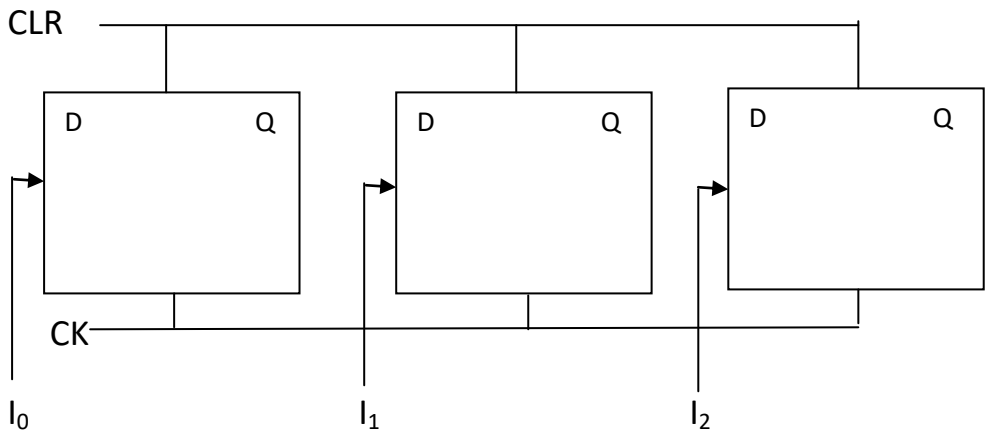
0	0	0	1	0	1	1	1

0	0	0	0	1	0	1	1

0	0	0	0	0	1	0	1

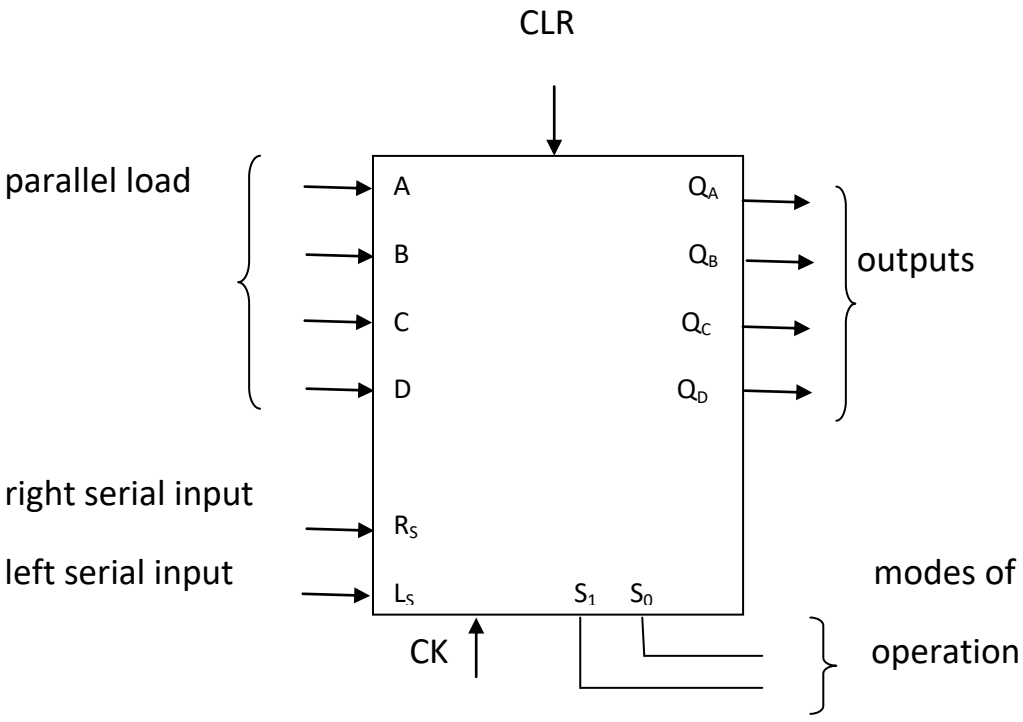
Parallel transfer :

Data can be transferred to or from all ffs at the same time:



A universal register may perform different methods of moving data into or out of register like:

1- Parallel in – parallel out.
2- Parallel in – serial out.
3- Serial in – parallel out.
4- Serial in – serial out.



Counter :

Flip-flops can be connected together to perform counting operations. The number of ffs used and the way in which they are connected determine the number of states (called modulus).

Basically counters are of two types:

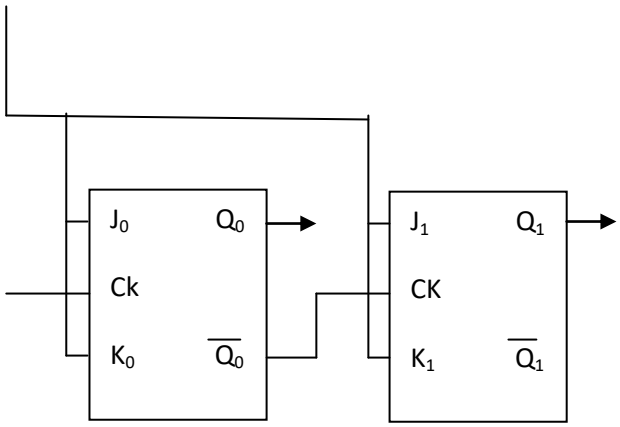
1- **Asynchronous counter (Ripple counter):**

An external clock signal is applied to the first ff and then the output of the preceding ff is connected to the clock of the next ff.

EX: Design 2- bit asynchronous binary counter:

Clk	Q1	Q0
Initially	0	0
1	0	1
2	1	0
3	1	1
4	0	0

logic (1)



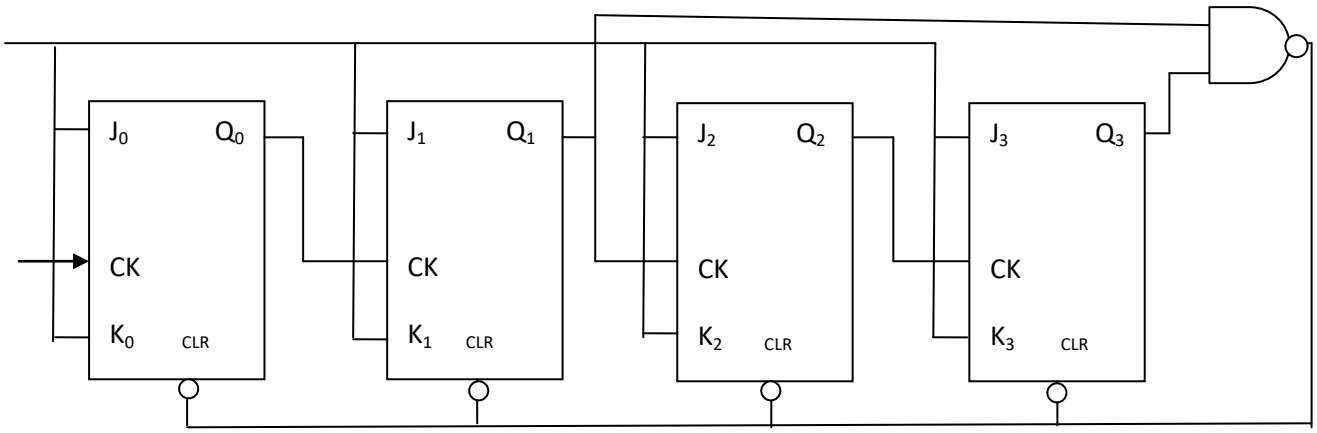
H.W. : Design 3- bit asynchronous counter .

BCD decade counters (MOD 10) :

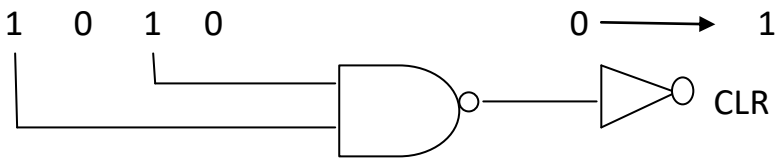
The number of states are ten (0000 – 1001). In this type the counter should be count back to (0000) after (1001).

0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001

Logic(1)



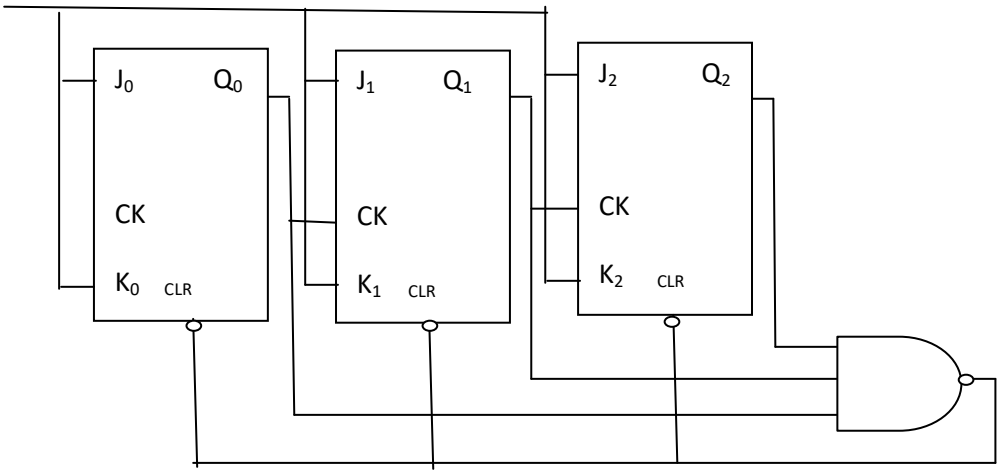
Skipped Q_3 Q_2 Q_1 Q_0



***EX*:** Design MOD 7 counter:

The counter should count from 000- 011 so there are 3 ffs. But state 111 should be skipped:

logic(1)



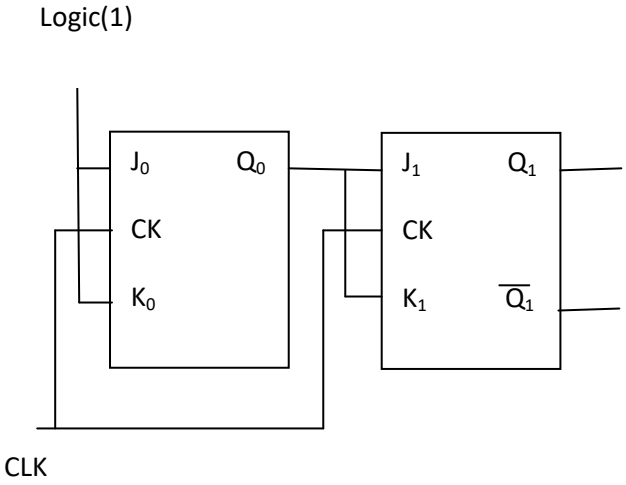
H.W. : Design MOD 12 counter (0000 – 1011)

2- **Synchronous counter (parallel counter) :**

All the ffs in the counter are clocked at the same time by a common clock pulse.

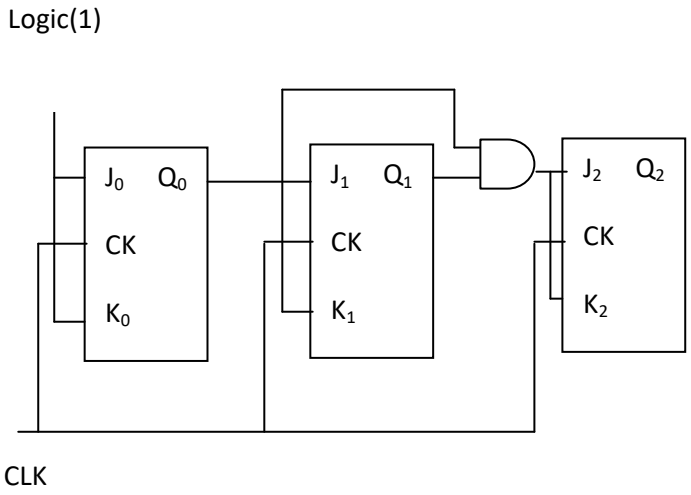
***EX*:** Design 2-bit counter

Clk	Q1	Q0
Initially	0	0
1	0	1
2	1	0
3	1	1
4	0	0



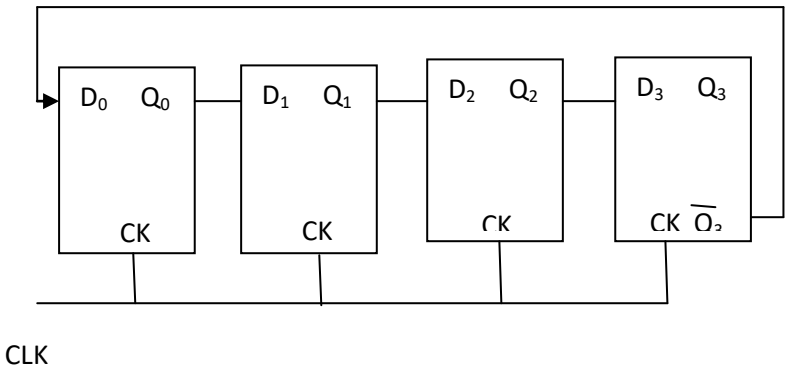
***EX*:** Design 3-bit synchronous counter:

CLK	Q2	Q1	Q0
Initially	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1



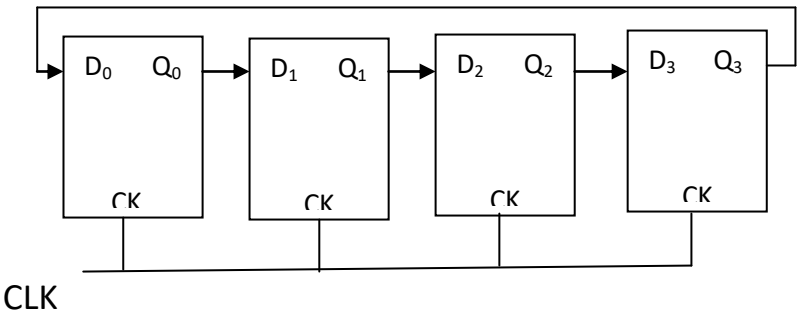
EX: Design 4-bit Johnson counter:

CLK	Q0	Q1	Q2	Q3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1



EX: Design 4- bit Ring counter:

0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1

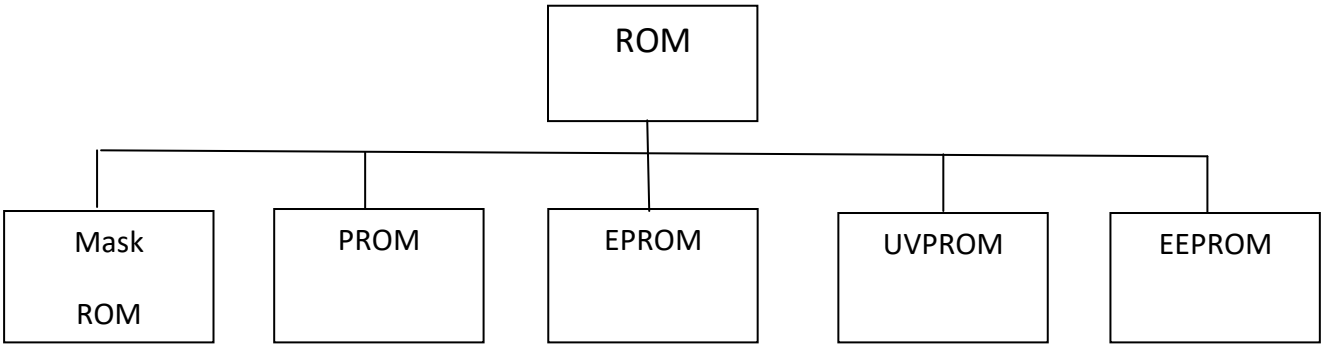


Note: initially a (1) present into first ff and the rest are cleared.

ROM(read only memory) :

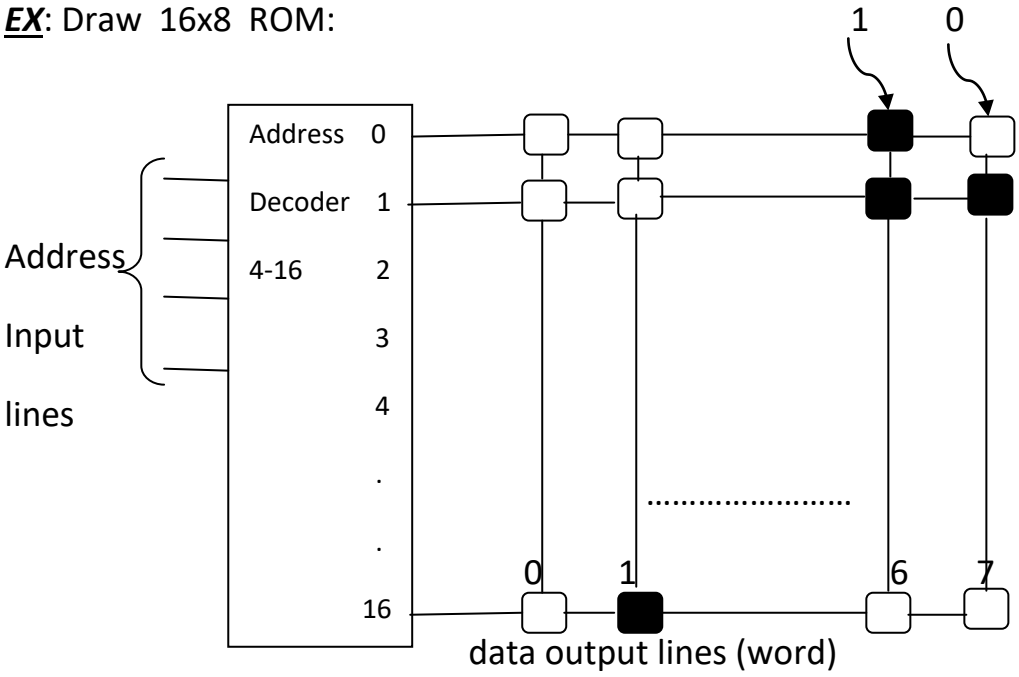
A ROM contains permanently or semi permanently stored data, which can be read from the memory but either cannot be changed at all or cannot be changed without specialized equipment. ROMs retain stored data when the power is OFF.

ROM family:



A simple ROM :

EX: Draw 16x8 ROM:



ROMs can be used as look-up-tables (LUT) for code conversion and logic function.

EX: Show a ROM programmed for 4-bit binary to gray conversion:

Binary				Gray			
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
	.				.		
	.				.		
1	1	1	1	1	0	0	0