REVA UNIVERSITY

BENGALURU, INDIA

Analog and Digital Electronics

B20CS0301

For Second Year B.Tech

Unit-4

Notes

SCHOOL OF COMPUTING AND INFORMATION TECHNOLOGY

CONTENTS

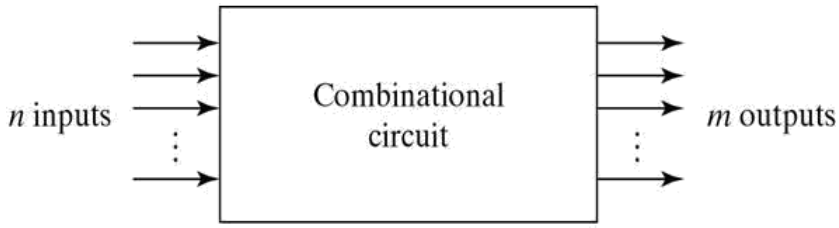
UNIT.NO	SYLLABUS	PAGE NO
4	**Analysis of Combinational and sequential Circuits:** Half adder, full Adder, Half Subtractor, full Subtractor, multiplexers and Demultiplexers. Application study2: Calculator **Introduction to Sequential circuits:** flip-flops: SR, JK, D, T Characteristic tables and equations; Application of Shift register (Ring Counter and Johnson counter) Application sudy3: Digital combinational lock.	

UNIT-4
ANALYSIS OF COMBINATIONAL AND SEQUENTIAL CIRCUITS

COMBINATIONAL CIRCUITS

Combinational Logic

- Logic circuits for digital systems may be combinational or sequential.
- A combinational circuit consists of input variables, logic gates, and output variables.



For n input variables, there are 2^n possible combinations of binary input variables .For each possible input Combination, there is one and only one possible output combination. A combinational circuit can be described by m Boolean functions one for each output variables. Usually the input s comes from flip-flops and outputs go to flip-flops.

Design Procedure:

1. The problem is stated
2. The number of available input variables and required output variables is Determined.
3. The input and output variables are assigned letter symbols.
4. The truth table that defines the required relationship between inputs and outputs is derived.
5. The simplified Boolean function for each output is obtained.

Digital computers perform a variety of information processing tasks, the one is arithmetic operations. And the most basic arithmetic operation is the addition of two binary digits. i.e, 4 basic possible operations are:

0+0=0,0+1=1,1+0=1,1+1=10

The first three operations produce a sum whose length is one digit, but when augends and addend bits are equal to 1,the binary sum consists of two digits. The most significant bit of this result is called a carry. A combinational circuit that performs the addition of two bits is called a half-adder. One that performs the addition of 3 bits (two significant bits & previous carry) is called a full adder.& 2 half adder can be employed as a full adder.

The Half Adder:

A Half Adder is a combinational circuit with two binary inputs (augends and addend bits and two binary outputs (sum and carry bits.) It adds the two inputs (A and B) and produces the sum (S) and the carry (C) bits. It is an arithmetic operation of addition of two single bit words.

Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(a) Truth table

(b) Block diagram

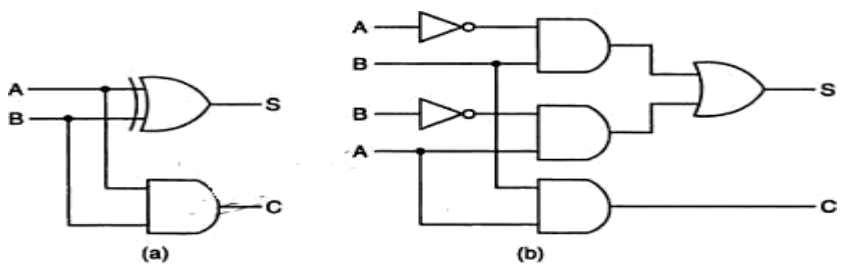
The Sum(S) bit and the carry (C) bit, according to the rules of binary addition, the sum (S) is the X-OR of A and B (It represents the LSB of the sum). Therefore,

$S = A'B+AB' = A \oplus B$

The carry (C) is the AND of A and B (it is 0 unless both the inputs are 1).Therefore,

$C = AB$

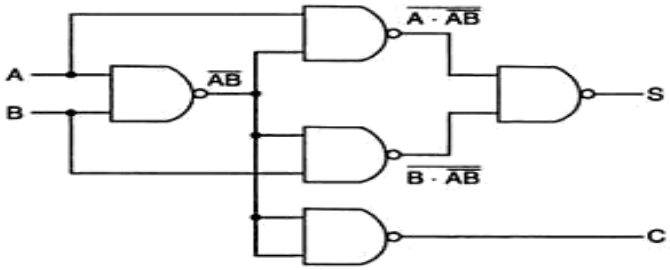
A half-adder can be realized by using one X-OR gate and one AND gate a



Logic diagrams of half-adder

NAND LOGIC:

$$S = A\bar{B} + \bar{A}B = A\bar{B} + A\bar{A} + \bar{A}B + B\bar{B}$$
$$= A(\bar{A} + \bar{B}) + B(\bar{A} + \bar{B})$$
$$= A \cdot \overline{AB} + B \cdot \overline{AB}$$
$$= \overline{A \cdot \overline{AB}} \cdot \overline{B \cdot \overline{AB}}$$
$$C = AB = \overline{\overline{AB}}$$

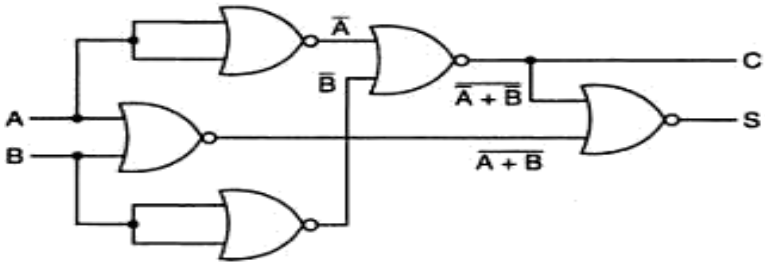


Logic diagram of a half-adder using only 2-input NAND gates.

NOR logic:

$$S = A\bar{B} + \bar{A}B = A\bar{B} + A\bar{A} + \bar{A}B + B\bar{B}$$
$$= A(\bar{A} + \bar{B}) + B(\bar{A} + \bar{B})$$

$$= (A + B)(\bar{A} + \bar{B})$$
$$= \overline{\overline{A + B} + \overline{\bar{A} + \bar{B}}}$$
$$C = AB = \overline{\overline{AB}} = \overline{\bar{A} + \bar{B}}$$



Logic diagram of a half-adder using only 2-input NOR gates.

The Full Adder:

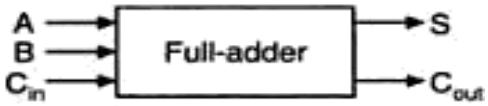
A Full-adder is a combinational circuit that adds two bits and a carry and outputs a sum bit and a carry bit. To add two binary numbers, each having two or more bits, the LSBs can be added by using a half-adder. The carry resulting from the addition of the LSBs is carried over to the next significant column and added to the two bits in that column. So, in the second and higher columns, the two data bits of that column and the carry bit generated from the addition in the previous column need to be added.

The full-adder adds the bits A and B and the carry from the previous column called the carry-in C_{in} and outputs the sum bit S and the carry bit called the carry-out C_{out} . The variable S gives the value of the least significant bit of the sum. The variable C_{out} gives the output carry. The eight rows under the input variables designate all possible combinations of 1s and 0s that these variables may have. The 1s and 0s for the output variables are determined from the arithmetic sum of the input bits.

When all the bits are 0s , the output is 0. The S output is equal to 1 when only 1 input is equal to 1 or when all the inputs are equal to 1. The C_{out} has a carry of 1 if two or three inputs are equal to 1

Inputs			Sum	Carry
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a) Truth table



(b) Block diagram

Full-adder.

$$S = A'B'C_{in} + A'BC_{in}' + AB'C_{in}' + ABC_{in}$$

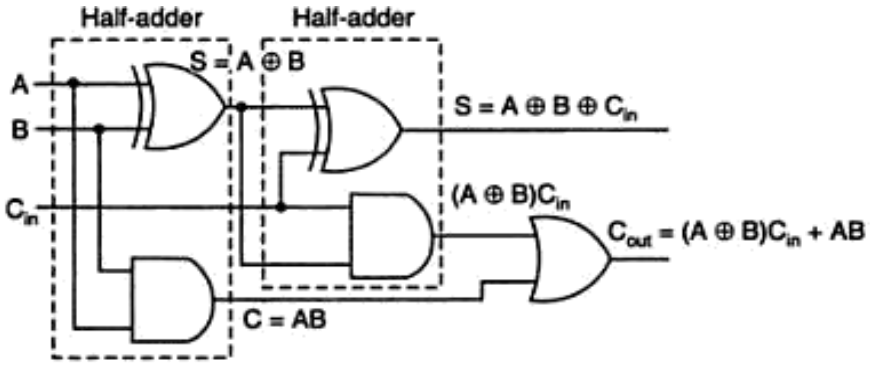
$$C_{out} = A'BC_{in}' + AB'C_{in}' + ABC_{in}' + ABC_{in}$$

and

$$S = A \; Xor \; B \; Xor \; C_{in}$$

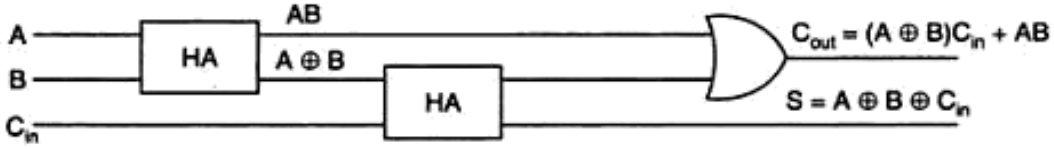
$$C_{out} = AC_{in} + BC_{in} + AB$$

The sum term of the full-adder is the X-OR of A,B, and C_{in}, i.e, the sum bit the modulo sum of the data bits in that column and the carry from the previous column. The logic diagram of the full-adder using two X-OR gates and two AND gates (i.e, Two half adders) and one OR gate is



Logic diagram of a full-adder using two half-adders.

The block diagram of a full-adder using two half-adders is



Block diagram of a full-adder using two half-adders.

Even though a full-adder can be constructed using two half-adders, the disadvantage is that the bits must propagate through several gates in accession, which makes the total propagation delay greater than that of the full-adder circuit using AOI logic

Subtractors:

The subtraction of two binary numbers may be accomplished by taking the complement of the subtrahend and adding it to the minuend. By this, the subtraction operation becomes an addition operation and instead of having a separate circuit for subtraction, the adder itself can be used to perform subtraction. This results in a reduction of hardware. In subtraction, each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a difference bit. If the minuend bit is smaller than the subtrahend bit, a 1 is borrowed from the next significant position., that has been borrowed must be conveyed to the next higher pair of bits by means of a signal coming out (output) of a given stage and going into (input) the next higher stage.

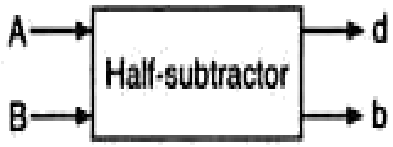
The Half-Subtractor:

A Half-subtractor is a combinational circuit that subtracts one bit from the other and produces the difference. It also has an output to specify if a 1 has been borrowed. . It is used to subtract the LSB of the subtrahend from the LSB of the minuend when one binary number is subtracted from the other.

A Half-subtractor is a combinational circuit with two inputs A and B and two outputs d and b. d indicates the difference and b is the output signal generated that informs the next stage that a 1 has been borrowed. When a bit B is subtracted from another bit A, a difference bit (d) and a borrow bit (b) result according to the rules given as

Inputs		Outputs	
A	B	d	b
0	0	0	0
1	0	1	0
1	1	0	0
0	1	1	1

(a) Truth table



(b) Block diagram

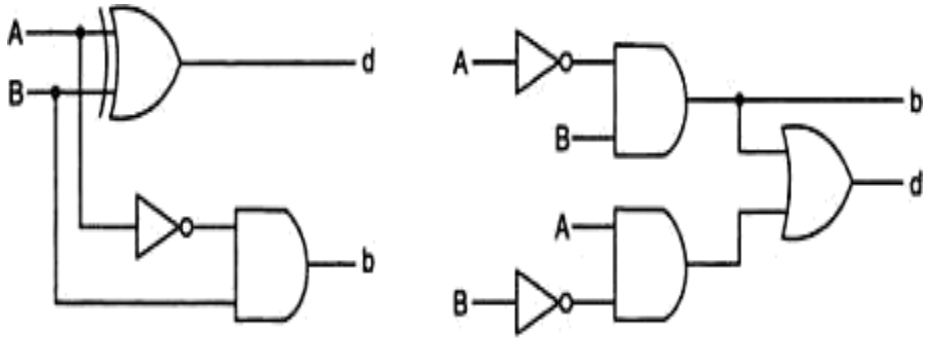
Half-subtractor.

The output borrow b is a 0 as long as A≥B. It is a 1 for A=0 and B=1. The d output is the result of the arithmetic operation2b+A-B.

A circuit that produces the correct difference and borrow bits in response to every possible combination of the two 1-bit numbers is , therefore ,

$$d=A'B+AB' = A \oplus B \text{ and } b = A'B$$

That is, the difference bit is obtained by X-OR ing the two inputs, and the borrow bit is obtained by ANDing the complement of the minuend with the subtrahend. Note that logic for this exactly the same as the logic for output S in the half-adder



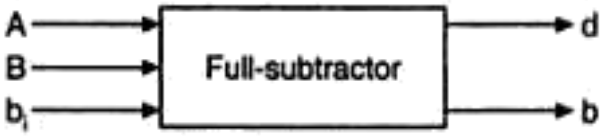
Logic diagrams of a half-subtractor.

The Full-Subtractor:

The half-subtractor can be only for LSB subtraction. IF there is a borrow during the subtraction of the LSBs, it affects the subtraction in the next higher column; the subtrahend bit is subtracted from the minuend bit, considering the borrow from that column used for the subtraction in the preceding column. Such a subtraction is performed by a full subtractor. It subtracts one bit (B) from another bit (A) , when already there is a borrow b_i from this column for the subtraction in the preceding column, and outputs the difference bit (d) and the borrow bit(b) required from the next d and b. The two outputs present the difference and output borrow. The 1s and 0s for the output variables are determined from the subtraction of A-B-b_i.

Inputs			Difference	Borrow
A	B	b_i	d	b
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

(a) Truth table



(b) Block diagram

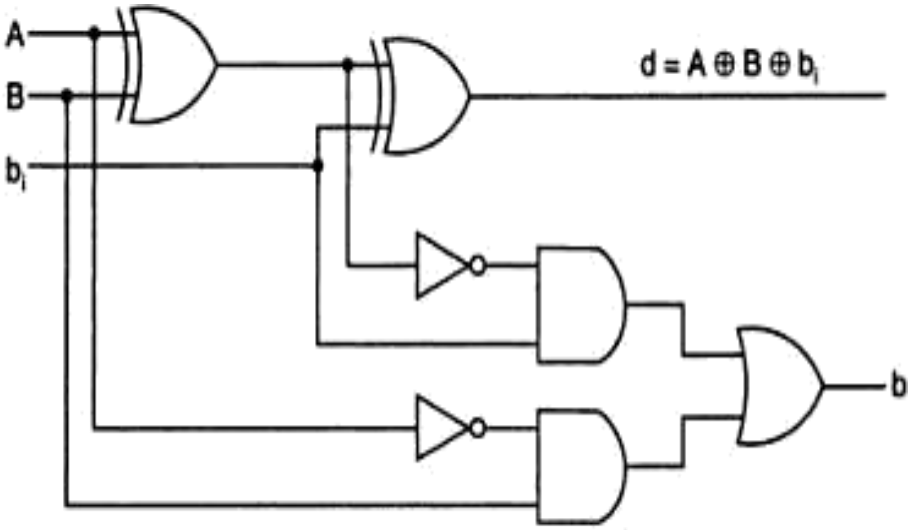
Full-subtractor.

From the truth table, a circuit that will produce the correct difference and borrow bits in response to every possible combinations of A,B and b_i is

$$d = \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + A\bar{B}\bar{b}_i + ABb_i$$
$$= b_i(AB + \bar{A}\bar{B}) + \bar{b}_i(A\bar{B} + \bar{A}B)$$
$$= b_i(\overline{A \oplus B}) + \bar{b}_i(A \oplus B) = A \oplus B \oplus b_i$$

and

$$b = \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + \bar{A}Bb_i + ABb_i = \bar{A}B(b_i + \bar{b}_i) + (AB + \bar{A}\bar{B})b_i$$
$$= \bar{A}B + (\overline{A \oplus B})b_i$$



Logic diagram of a full-subtractor.

A full-subtractor can be realized using X-OR gates and AND and NOT gates as

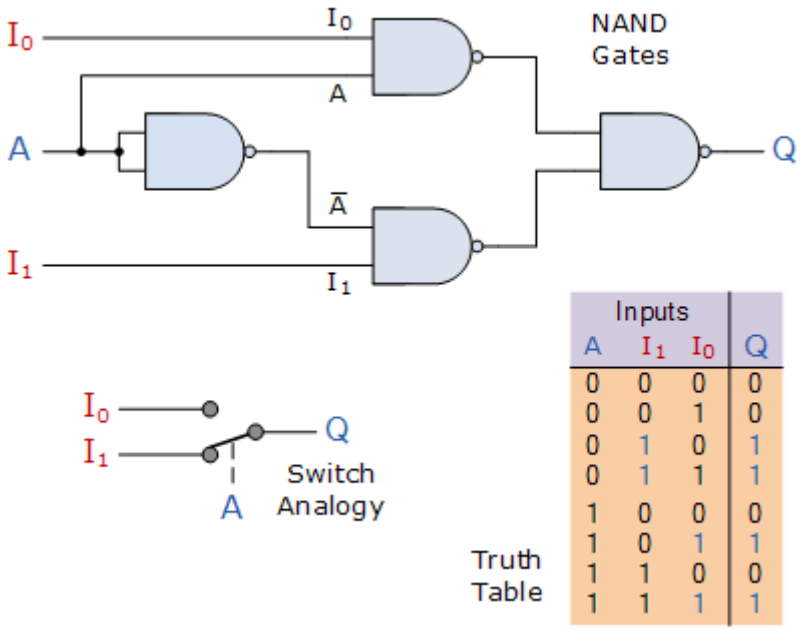
The Multiplexer:

The multiplexer is a combinational logic circuit designed to switch one of several input lines to a single common output line

Multiplexing is the generic term used to describe the operation of sending one or more analogue or digital signals over a common transmission line at different times or speeds and as such, the device we use to do just that is called a **Multiplexer**.

The *multiplexer*, shortened to “MUX” or “MPX”, is a combinational logic circuit designed to switch one of several input lines through to a single common output line by the application of a control signal. Multiplexers operate like very fast acting multiple position rotary switches connecting or controlling multiple input lines called “channels” one at a time to the output.

2-input Multiplexer Design



Inputs			
A	I_1	I_0	Q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Truth Table

The input A of this simple 2-1 line multiplexer circuit constructed from standard NAND gates acts to control which input (I_0 or I_1) gets passed to the output at Q.

From the truth table above, we can see that when the data select input, A is LOW at logic 0, input I_1 passes its data through the NAND gate multiplexer circuit to the output, while input I_0 is blocked. When the data select A is HIGH at logic 1, the reverse happens and now input I_0 passes data to the output Q while input I_1 is blocked.

So by the application of either a logic "0" or a logic "1" at A we can select the appropriate input, I_0 or I_1 with the circuit acting a bit like a single pole double throw (SPDT) switch.

As we only have one control line, (A) then we can only switch 2^1 inputs and in this simple example, the 2-input multiplexer connects one of two 1-bit sources to a common output, producing a 2-to-1-line multiplexer. We can confirm this in the following Boolean expression.

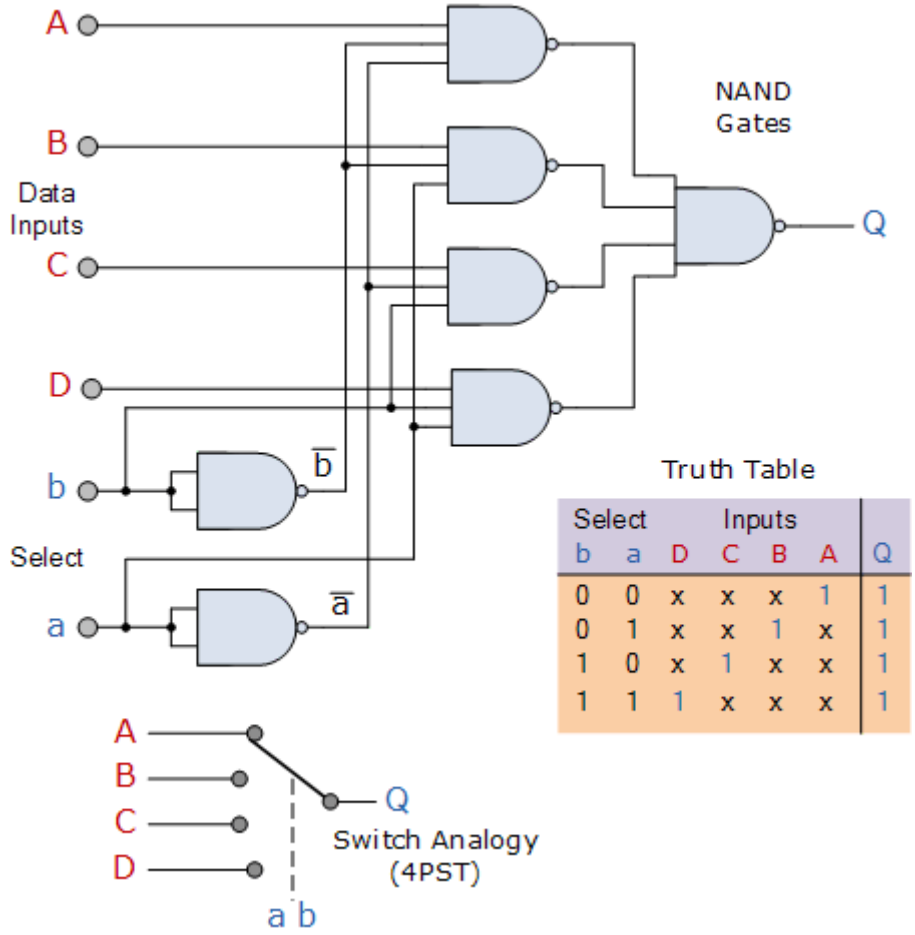
$$Q = A.I_0.I_1 + A.I_0.I_1 + A.I_0.I_1 + A.I_0.I_1$$

and for our 2-input multiplexer circuit above, this can be simplified too:

$$Q = A.I_1 + A.I_0$$

We can increase the number of data inputs to be selected further simply by following the same procedure and larger multiplexer circuits can be implemented using smaller 2-to-1 multiplexers as their basic building blocks. So for a 4-input multiplexer we would therefore require two data select lines as 4-inputs represents 2^2 data control lines give a circuit with four inputs, I_0, I_1, I_2, I_3 and two data select lines A and B as shown.

4-to-1 Channel Multiplexer



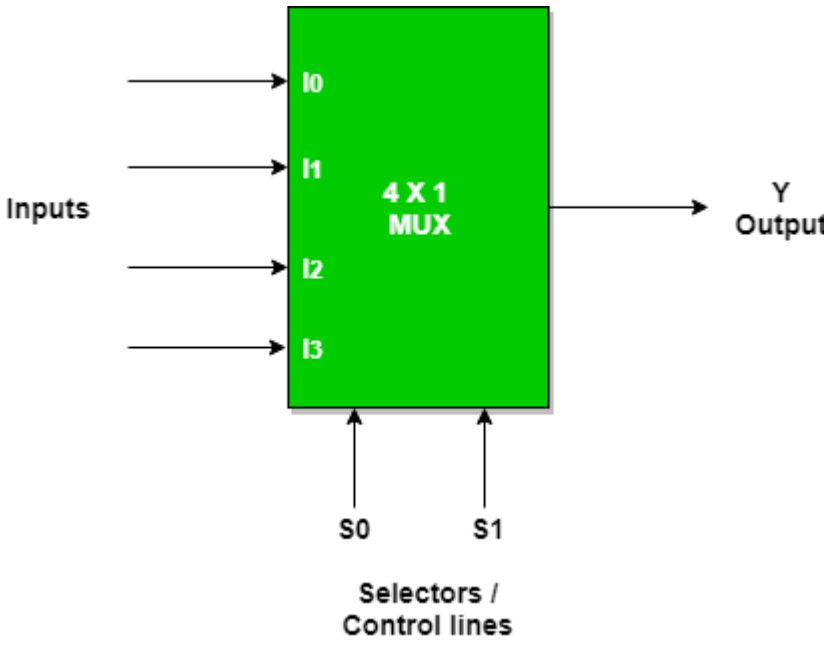
The Boolean expression for this 4-to-1 **Multiplexer** above with inputs A to D and data select lines a, b is given as:

$$Q = a'b'A + a'bB + ab'C + abD$$

In this example at any one instant in time only ONE of the four analogue switches is closed, connecting only one of the input lines A to D to the single output at Q. As to which switch is closed depends upon the addressing input code on lines "a" and "b".

So for this example to select input B to the output at Q, the binary input address would need to be "a" = logic "1" and "b" = logic "0". Thus we can show the selection of the data through the multiplexer as a function of the data select bits as shown.

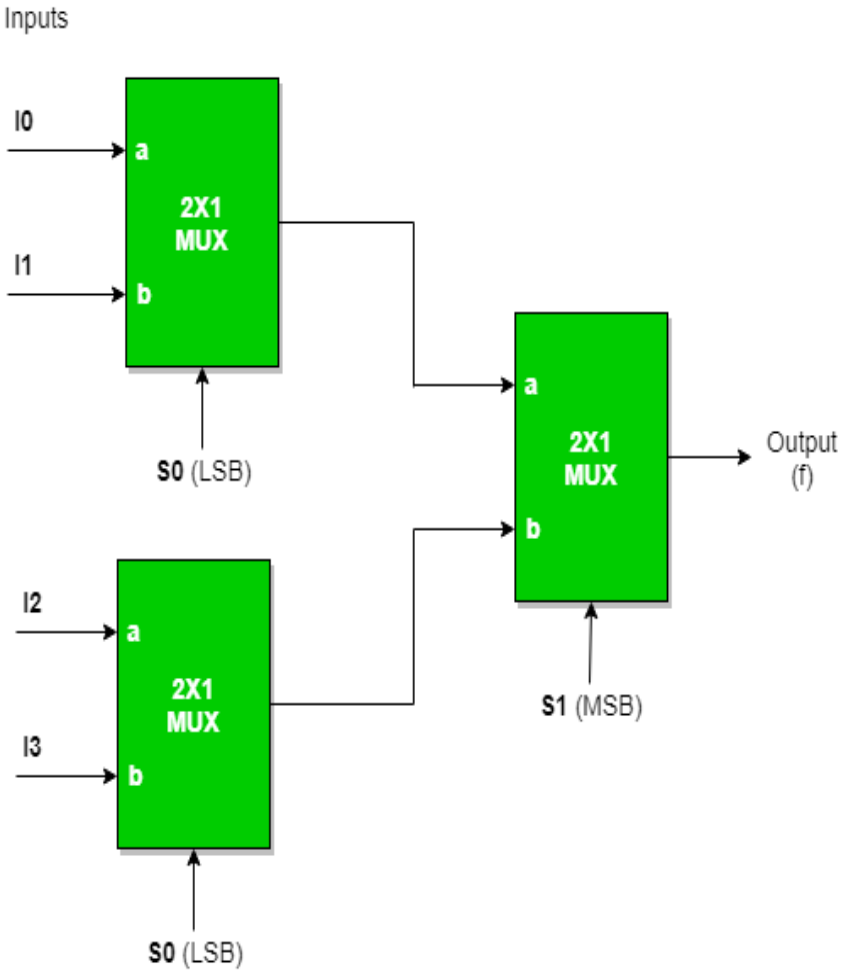
It is a combinational circuit which have many data inputs and single output depending on control or select inputs. For N input lines, log n (base2) selection lines, or we can say that for 2^n input lines, n selection lines are required. Multiplexers are also known as **"Data n selector, parallel to serial convertor, many to one circuit, universal logic circuit"**. Multiplexers are mainly used to increase amount of the data that can be sent over the network within certain amount of time and bandwidth.



Implementation of Higher order MUX using lower order MUX

a) 4 : 1 MUX using 2 : 1 MUX

Three(3) 2 : 1 MUX are required to implement 4 : 1 MUX.



Truth Table

S0	S1	f
0	0	I0
0	1	I1
1	0	I2
1	1	I3

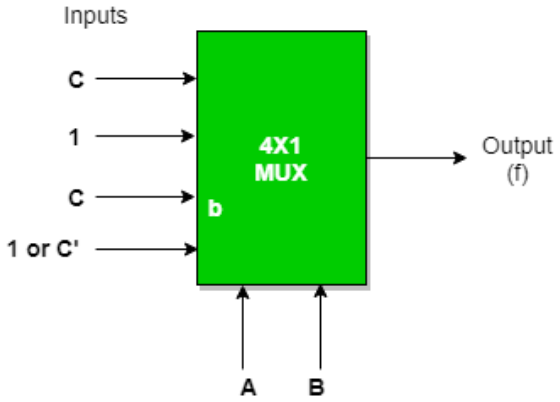
An example to implement a boolean function if minimal and don't care terms are given using MUX.

f (A, B, C) = Σ (1, 2, 3, 5, 6) with don't care (7) using 4 : 1 MUX using as

a) AB as select : Expanding the minterms to its boolean form and will see its 0 or 1 value in Cth place so that they can be placed in that manner.

A	B	C	f
0	0	0	C′
0	0	1	C
0	1	0	C′
0	1	1	C
1	0	0	C′
1	0	1	C
1	1	0	C′
1	1	1	C

	I0	I1	I2	I3
C′	0	2	4	6
C	1	3	5	7 is don't care (can consider or not)
	C	1	C	1 (in case 7 is considered or C′)



Implementation of Higher-order Multiplexers.

Now, let us implement the following two higher-order Multiplexers using lower-order Multiplexers.

- 8x1 Multiplexer
- 16x1 Multiplexer

8x1 Multiplexer:

In this section, let us implement 8x1 Multiplexer using 4x1 Multiplexers and 2x1 Multiplexer. We know that 4x1 Multiplexer has 4 data inputs, 2 selection lines and one output. Whereas 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output.

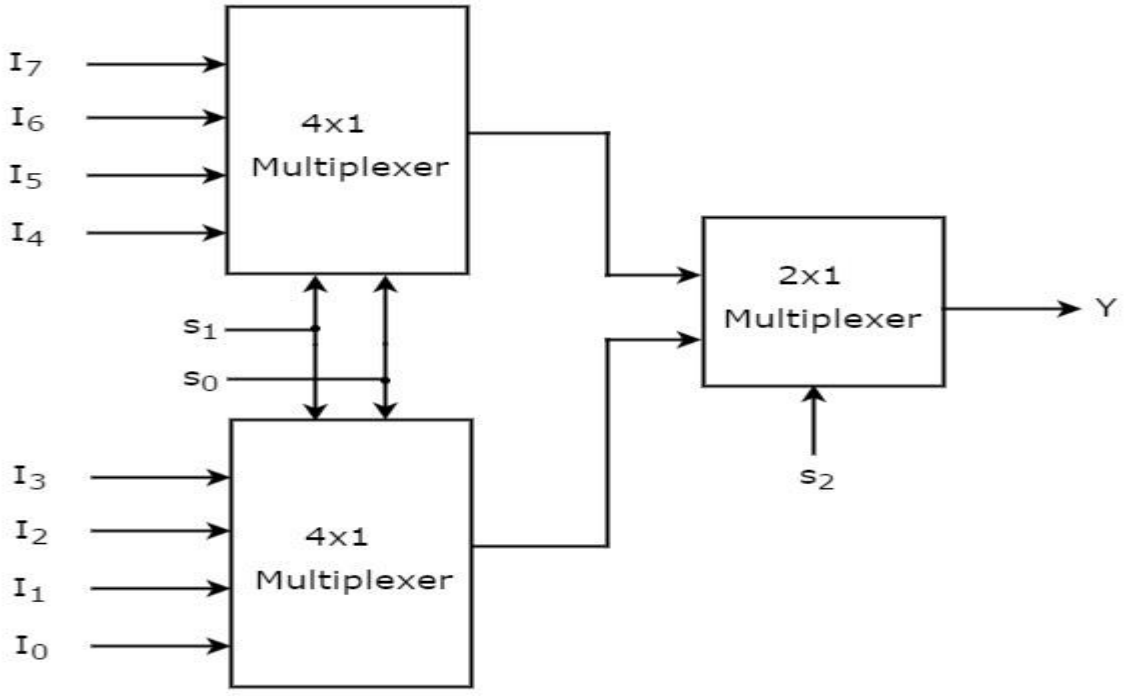
So, we require two **4x1 Multiplexers** in first stage to get the 8 data inputs. Since, each 4x1 Multiplexer produces one output, we require a **2x1 Multiplexer** in second

stage by considering the outputs of first stage as inputs and to produce the final output.

Let the 8x1 Multiplexer has eight data inputs I_7 to I_0, three selection lines s_2, s_1 & s0 and one output Y. The **Truth table** of 8x1 Multiplexer is shown below.

Selection Inputs			Output
S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

We can implement 8x1 Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 8x1 Multiplexer is shown in the following figure.



The same **selection lines, s_1 & s_0** are applied to both 4x1 Multiplexers. The data inputs of upper 4x1 Multiplexer are I_7 to I_4 and the data inputs of lower 4x1 Multiplexer are I_3 to I_0. Therefore, each 4x1 Multiplexer produces an output based on the values of selection lines, s_1 & s_0.

The outputs of first stage 4x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, s_2** is applied to 2x1 Multiplexer.

- If s_2 is zero, then the output of 2x1 Multiplexer will be one of the 4 inputs I_3 to I_0 based on the values of selection lines s_1 & s_0.
- If s_2 is one, then the output of 2x1 Multiplexer will be one of the 4 inputs I_7 to I_4 based on the values of selection lines s_1 & s_0.

Therefore, the overall combination of two 4x1 Multiplexers and one 2x1 Multiplexer performs as one 8x1 Multiplexer.

16x1 Multiplexer:

In this section, let us implement 16x1 Multiplexer using 8x1 Multiplexers and 2x1 Multiplexer. We know that 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output. Whereas 16x1 Multiplexer has 16 data inputs, 4 selection lines and one output.

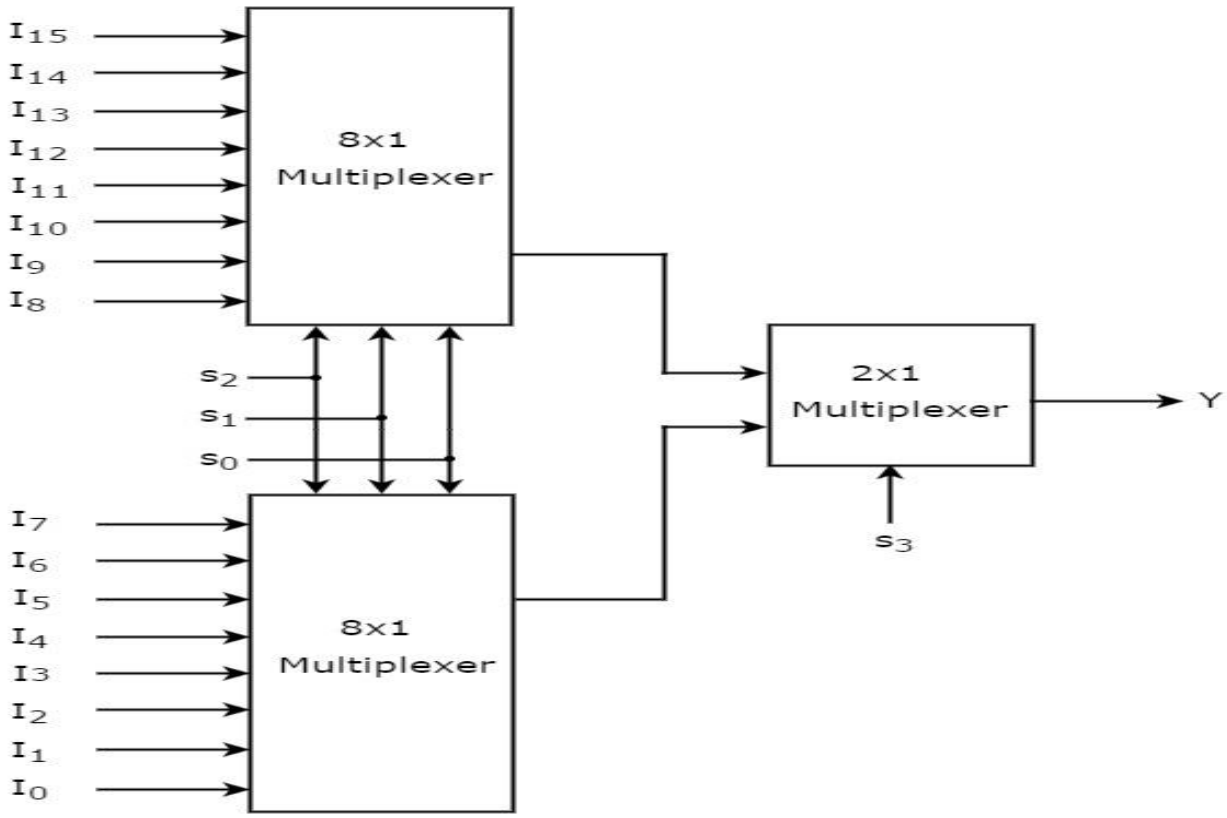
So, we require two **8x1 Multiplexers** in first stage to get the 16 data inputs. Since, each 8x1 Multiplexer produces one output, we require a 2x1 Multiplexer in second stage by considering the outputs of first stage as inputs and to produce the final output.

Let the 16x1 Multiplexer has sixteen data inputs I_{15} to I_0, four selection lines s_3 to s_0 and one output Y. The **Truth table** of 16x1 Multiplexer is shown below.

Selection Inputs				Output
S_3	S_2	S_1	S_0	Y
0	0	0	0	I_0
0	0	0	1	I_1
0	0	1	0	I_2
0	0	1	1	I_3
0	1	0	0	I_4

0	1	0	1	I_5
0	1	1	0	I_6
0	1	1	1	I_7
1	0	0	0	I_8
1	0	0	1	I_9
1	0	1	0	I_{10}
1	0	1	1	I_{11}
1	1	0	0	I_{12}
1	1	0	1	I_{13}
1	1	1	0	I_{14}
1	1	1	1	I_{15}

We can implement 16x1 Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 16x1 Multiplexer is shown in the following figure.



The **same selection lines, s_2, s_1 & s_0** are applied to both 8x1 Multiplexers. The data inputs of upper 8x1 Multiplexer are I_{15} to I_8 and the data inputs of lower 8x1

Multiplexer are I_7 to I_0. Therefore, each 8x1 Multiplexer produces an output based on the values of selection lines, s_2, s_1 & s_0.

The outputs of first stage 8x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, s_3** is applied to 2x1 Multiplexer.

- If s_3 is zero, then the output of 2x1 Multiplexer will be one of the 8 inputs Is_7 to I_0 based on the values of selection lines s_2, s_1 & s_0.
- If s_3 is one, then the output of 2x1 Multiplexer will be one of the 8 inputs I_{15} to I_8 based on the values of selection lines s_2, s_1 & s_0.

Therefore, the overall combination of two 8x1 Multiplexers and one 2x1 Multiplexer performs as one 16x1 Multiplexer.

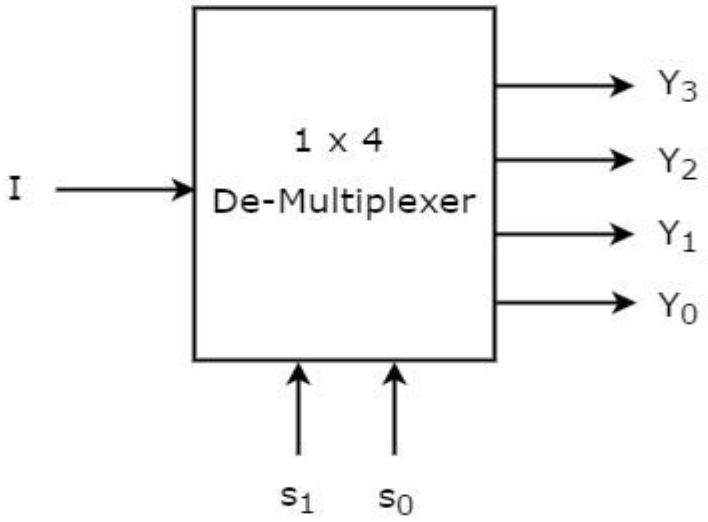
De-Multiplexer :

De-Multiplexer is a combinational circuit that performs the reverse operation of Multiplexer. It has single input, 'n' selection lines and maximum of 2^noutputs. The input will be connected to one of these outputs based on the values of selection lines.

Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination can select only one output. De-Multiplexer is also called as **De-Mux**.

1x4 De-Multiplexer:

1x4 De-Multiplexer has one input I, two selection lines, s_1 & s_0 and four outputs Y_3, Y_2, Y_1 &Y_0. The **block diagram** of 1x4 De-Multiplexer is shown in the following figure.



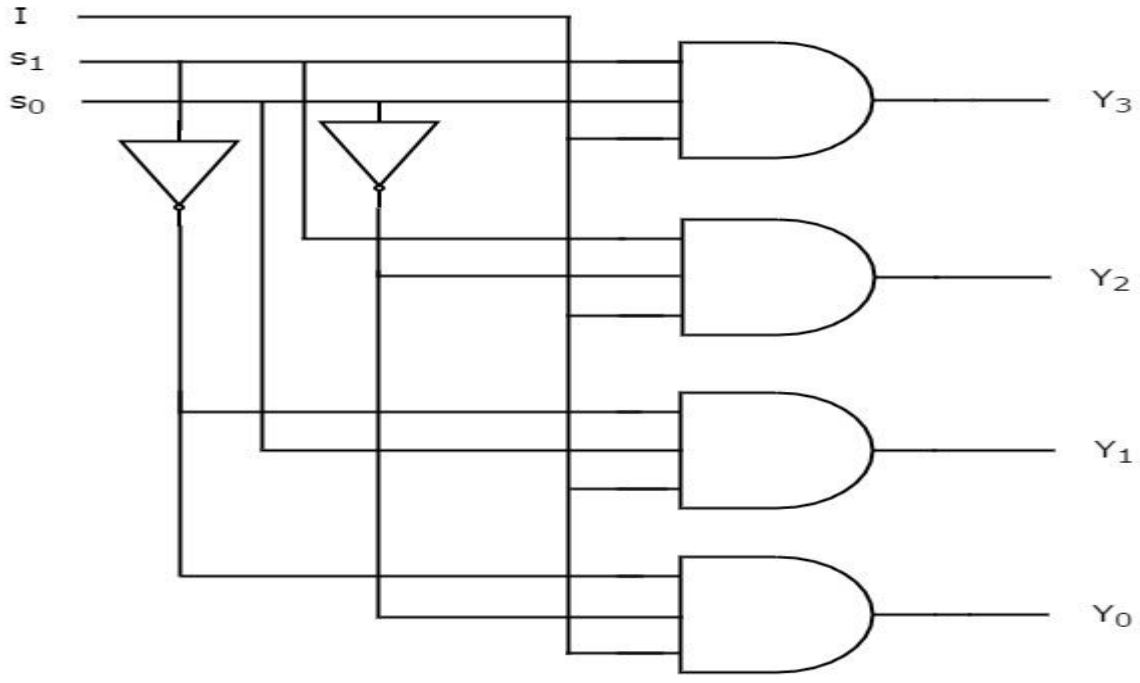
The single input 'I' will be connected to one of the four outputs, Y_3 to Y_0 based on the values of selection lines s_1 & s0. The **Truth table** of 1x4 De-Multiplexer is shown below.

Selection Inputs		**Outputs**			
S_1	**S_0**	**Y_3**	**Y_2**	**Y_1**	**Y_0**
0	0	0	0	0	**I**
0	1	0	0	**I**	0
1	0	0	**I**	0	0
1	1	**I**	0	0	0

From the above Truth table, we can directly write the **Boolean functions** for each output as

Y3=s1s0IY3=s1s0I
Y2=s1s0′IY2=s1s0′I
Y1=s1′s0IY1=s1′s0I
Y0=s1′s0′IY0=s1′s0′I

We can implement these Boolean functions using Inverters & 3-input AND gates. The **circuit diagram** of 1x4 De-Multiplexer is shown in the following figure.



We can easily understand the operation of the above circuit. Similarly, you can implement 1x8 De-Multiplexer and 1x16 De-Multiplexer by following the same procedure.

Implementation of Higher-order De-Multiplexers:

Now, let us implement the following two higher-order De-Multiplexers using lower-order De-Multiplexers.

- 1x8 De-Multiplexer
- 1x16 De-Multiplexer

1x8 De-Multiplexer:

In this section, let us implement 1x8 De-Multiplexer using 1x4 De-Multiplexers and 1x2 De-Multiplexer. We know that 1x4 De-Multiplexer has single input, two selection lines and four outputs. Whereas 1x8 De-Multiplexer has single input, three selection lines and eight outputs.

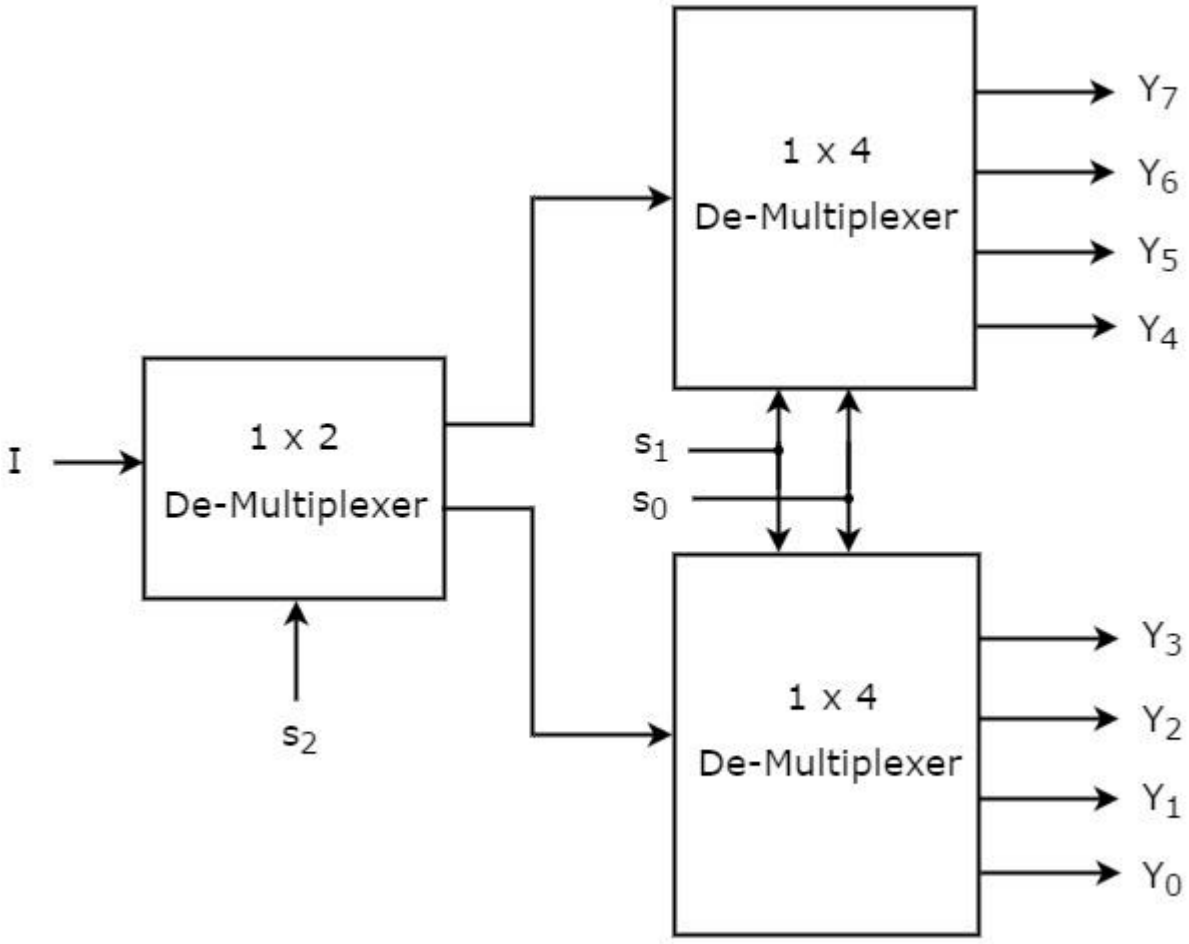
So, we require two **1x4 De-Multiplexers** in second stage to get the final eight outputs. Since, the number of inputs in second stage is two, we require **1x2 Demultiplexer** in first stage so that the outputs of first stage will be the inputs of second stage. Input of this 1x2 De-Multiplexer will be the overall input of 1x8 De-Multiplexer.

Let the 1x8 De-Multiplexer has one input I, three selection lines s_2, s_1 & $s_0$and outputs Y_7 to Y_0. The **Truth table** of 1x8 De-Multiplexer is shown below.

Selection Inputs			**Outputs**							
s_2	**s_1**	**s_0**	**Y_7**	**Y_6**	**Y_5**	**Y_4**	**Y_3**	**Y_2**	**Y_1**	**Y_0**
0	0	0	0	0	0	0	0	0	0	**I**
0	0	1	0	0	0	0	0	0	**I**	0
0	1	0	0	0	0	0	0	**I**	0	0
0	1	1	0	0	0	0	**I**	0	0	0
1	0	0	0	0	0	**I**	0	0	0	0

1	0	1	0	0	**I**	0	0	0	0	0
1	1	0	0	**I**	0	0	0	0	0	0
1	1	1	**I**	0	0	0	0	0	0	0

We can implement 1x8 De-Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 1x8 De-Multiplexer is shown in the following figure.



The common **selection lines, s_1 & s_0** are applied to both 1x4 De-Multiplexers. The outputs of upper 1x4 De-Multiplexer are Y_7 to Y_4 and the outputs of lower 1x4 De-Multiplexer are Y_3 to Y_0.

The other **selection line, s_2** is applied to 1x2 De-Multiplexer. If s_2 is zero, then one of the four outputs of lower 1x4 De-Multiplexer will be equal to input, I based on the values of selection lines s_1 & s_0. Similarly, if s_2 is one, then one of the four outputs of upper 1x4 Demultiplexer will be equal to input, I based on the values of selection lines s_1 & s_0.

1x16 De-Multiplexer:

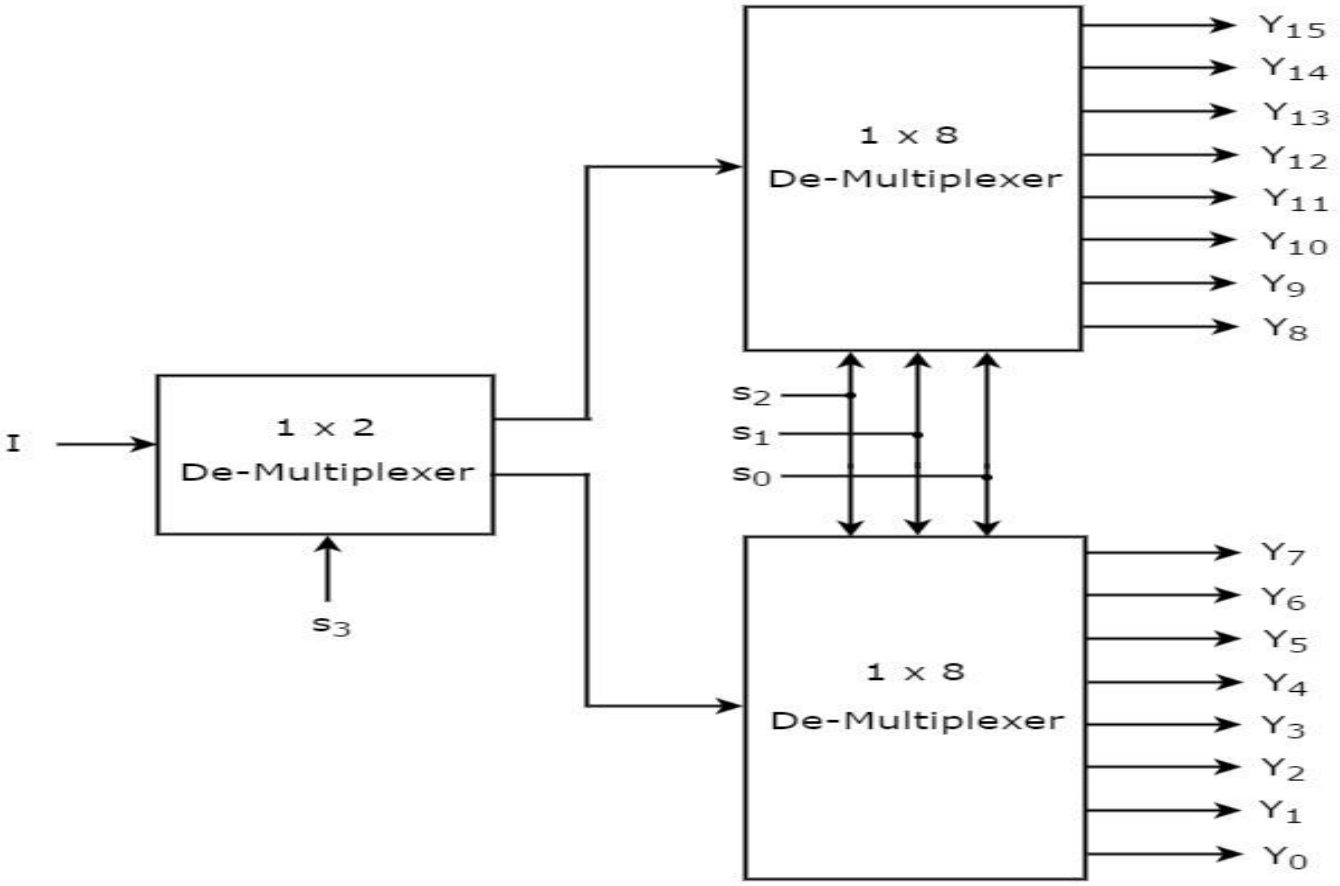
In this section, let us implement 1x16 De-Multiplexer using 1x8 De-Multiplexers and 1x2 De-Multiplexer. We know that 1x8 De-Multiplexer has single input, three selection lines and eight outputs. Whereas 1x16 De-Multiplexer has single input, four selection lines and sixteen outputs.

So, we require two **1x8 De-Multiplexers** in second stage to get the final sixteen outputs. Since, the number of inputs in second stage is two, we require **1x2 Demultiplexer** in first stage so that the outputs of first stage will be the inputs of second stage. Input of this 1x2 De-Multiplexer will be the overall input of 1x16 De-Multiplexer.

Let the 1x16 De-Multiplexer has one input I, four selection lines s_3, s_2, s_1 & $s_0$and outputs Y_{15} to Y_0. The **block diagram** of 1x16 De-Multiplexer using lower order Multiplexers is shown in the following figure.

The common **selection lines s_2, s_1 & s_0** are applied to both 1x8 De-Multiplexers. The outputs of upper 1x8 De-Multiplexer are Y_{15} to Y_8 and the outputs of lower 1x8 Demultiplexer are Y_7 to Y_0.

The other **selection line, s_3** is applied to 1x2 De-Multiplexer. If s_3 is zero, then one of the eight outputs of lower 1x8 De-Multiplexer will be equal to input, I based on the values of selection lines s_2, s_1 & s_0. Similarly, if s3 is one, then one of the 8 outputs of upper 1x8 De-Multiplexer will be equal to input, I based on the values of selection lines s_2, s_1 & s_0.

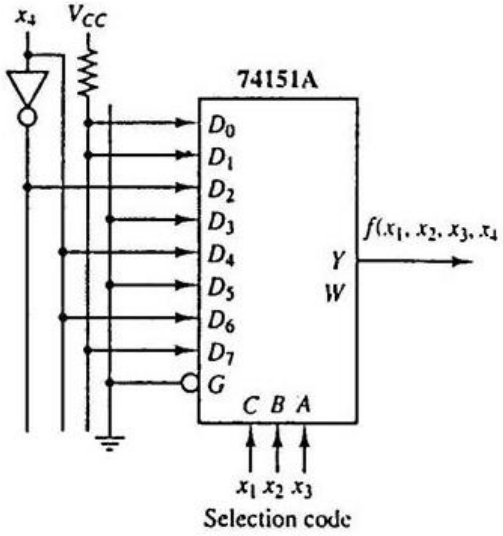


Implement the function $F(x_1,x_2,x_3,x_4) = \sum(0,1,2,3,4,9,13,14,15)$ using a single 74151A 8-to-1 mux and an inverter.

- **We choose the three most significant inputs x_1,x_2,x_3 as mux select lines.**
- **Construct truth table.**
- **Determine multiplexer Data input line D_i values.**

	C	B	A				Y
i	X_1	X_2	X_3	X_4	f	f	
0	0	0	0	0	1		
	0	0	0	1	1	1	$D_0 = 1$
1	0	0	1	0	1		
	0	0	1	1	1	1	$D_1 = 1$
2	0	1	0	0	1		
	0	1	0	1	0	$\bar{X}_4$	$D_2 = \bar{X}_4$
3	0	1	1	0	0		
	0	1	1	1	0	0	$D_3 = 0$
4	1	0	0	0	0		
	1	0	0	1	1	X_4	$D_4 = X_4$
5	1	0	1	0	0		
	1	0	1	1	0	0	$D_5 = 0$
6	1	1	0	0	0		
	1	1	0	1	1	X_4	$D_6 = X_4$
7	1	1	1	0	1		
	1	1	1	1	1	1	$D_7 = 1$

(a)



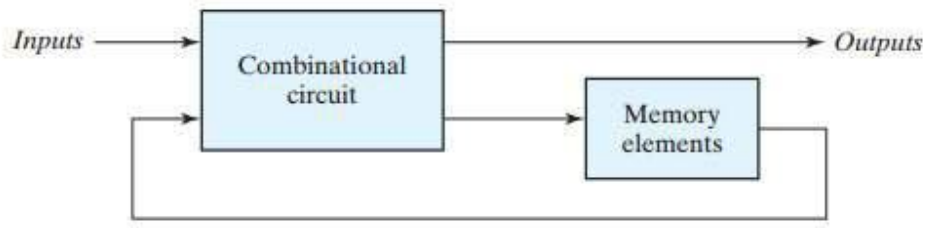
(b)

Introduction to Sequential circuits

SEQUENTIAL LOGIC CIRCUITS

Till now we studied the logic circuits whose outputs at any instant of time depend only on the input signals present at that time are known as combinational circuits. Moreover, in a combinational circuit, the output appears immediately for a change in input, except for the propagation delay through circuit gates.

On the other hand, the logic circuits whose outputs at any instant of time depend on the present inputs as well as on the past outputs are called sequential circuits. In sequential circuits, the output signals are fed back to the input side. A block diagram of a sequential circuit is shown in Figure below:-



It consists of a combinational circuit to which storage elements are connected to form a feedback path. The storage elements are devices capable of storing binary information. The binary information stored in these elements at any given time defines the ***state*** of the sequential circuit at that time. The sequential circuit receives binary information from external inputs that, together with the present state of the storage elements, determine the binary value of the outputs. These external inputs also determine the condition for changing the state in the

storage elements. The block diagram demonstrates that the outputs in a sequential circuit are a function not only of the inputs, but also of the present state of the storage elements. The next state of the storage elements is also a function of external inputs and the present state. Thus, **a sequential circuit is specified by a time sequence of inputs, outputs, and internal states.**

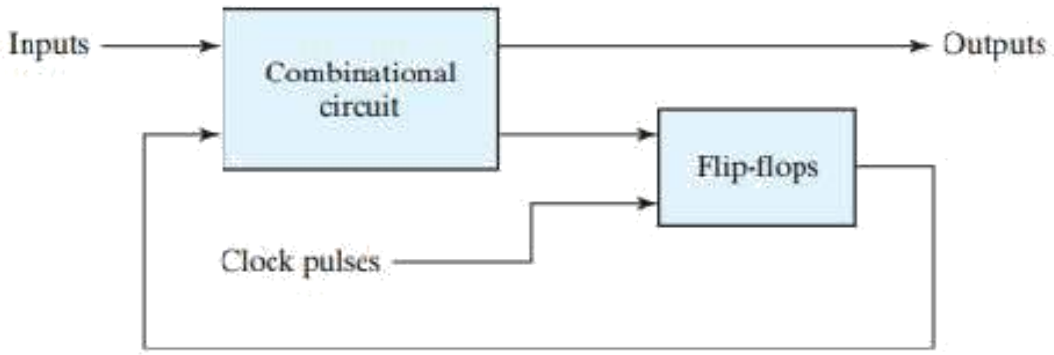
There are two types of sequential circuits, and their classification is a function of the timing of their signals.

Asynchronous sequential circuit:

A sequential circuit whose behavior depends upon the sequence in which the input signals change is referred to as an ***asynchronous sequential circuit***. The output will be affected whenever the input changes. The commonly used memory elements in these circuits are time-delay devices. There is no need to wait for a clock pulse. Therefore, in general, asynchronous circuits are faster than synchronous sequential circuits. However, in an asynchronous circuit, events are allowed to occur without any synchronization. And in such a case, the system becomes unstable. Since the designs of asynchronous circuits are more tedious and difficult, their uses are rather limited. The memory elements used in sequential circuits are flip-flops which are capable of storing binary information.

Synchronous sequential circuit:

A sequential circuit whose behavior can be defined from the knowledge of its signal at discrete instants of time is referred to as a ***synchronous sequential circuit***. In these systems, the memory elements are affected only at discrete instants of time. The synchronization is achieved by a timing device known as a system clock, which generates a periodic train of lock pulses. The outputs are affected only with the application of a clock pulse.



(a) Block diagram

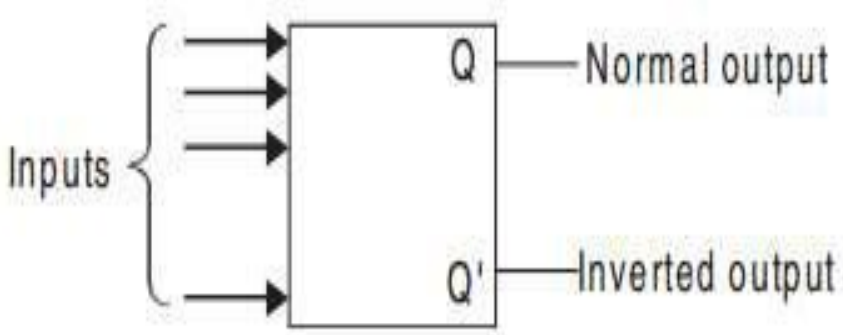
(b) Timing diagram of clock pulses

Synchronous clocked sequential circuit

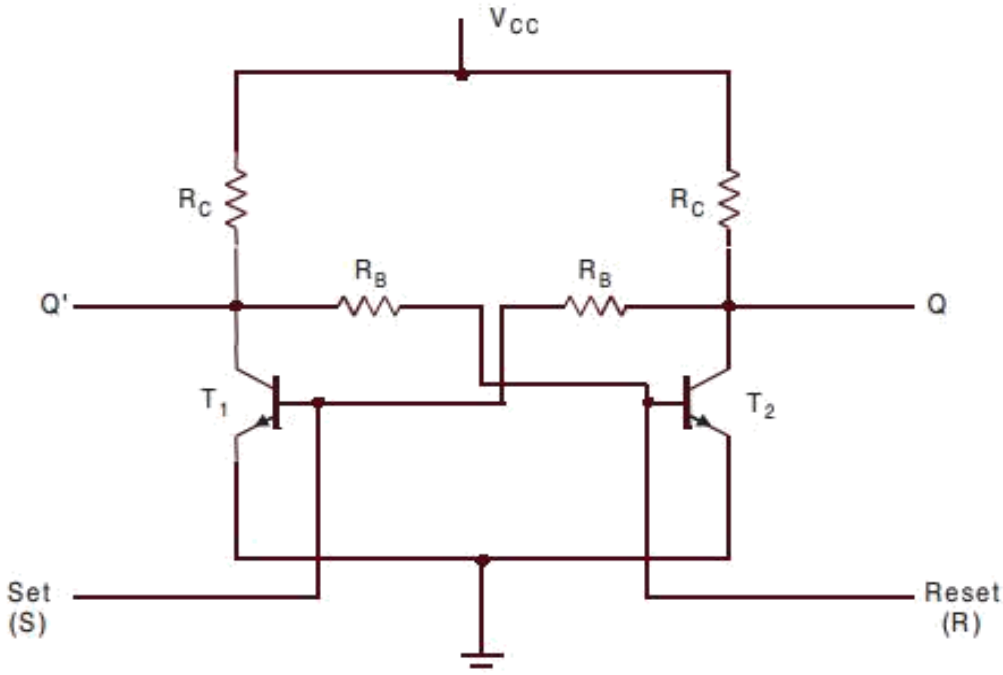
The storage elements (memory) used in clocked sequential circuits are called ***flipflops***

FLIPFLOPS

The basic 1-bit digital memory circuit is known as a flip-flop. It can have only two states, either the 1 state or the 0 state. A flip-flop is also known as a bistable multivibrator. Flip-flops can be obtained by using NAND or NOR gates. The general block diagram representation of a flip-flop is shown in Figure below. It has one or more inputs and two outputs. The two outputs are complementary to each other. If Q is 1 *i.e.*, Set, then Q' is 0; if Q is 0 *i.e.*, Reset, then Q' is 1. That means Q and Q' cannot be at the same state simultaneously. If it happens by any chance, it violates the definition of a flip-flop and hence is called an *undefined* condition. Normally, the state of Q is called the *state* of the flip-flop, whereas the state of Q' is called the *complementary state* of the flip-flop. When the output Q is either 1 or 0, it remains in that state unless one or more inputs are excited to effect a change in the output. Since the output of the flip-flop remains in the same state until the trigger pulse is applied to change the state, it can be regarded as a memory device to store one binary bit. The block diagram of a flip-flop is given below:-



The Bistable multivibrator circuit of a flip-flop is given below:-

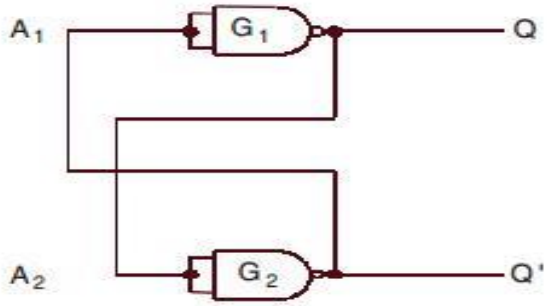


From the circuit shown above, the multivibrator is basically two cross-coupled inverting amplifiers, consist of two transistors and four resistors. Obviously, if transistor T_1 is initially turned ON (saturated) by applying a positive signal through the Set input at its base, its collector will be at $V_{CE(sat)}$ (0.2 to 0.4 V). The collector of T_1 is connected to the base of T_2, which cannot turn T_2 On. Hence, T_2 remains OFF (cut off). Therefore, the voltage at the collector of T_2 tries to reach V_{CC}. This action only enhances the initial positive signal applied to the base of T_1. Now if the initial signal at the Set input is removed, the circuit will maintain T_1 in the ON state and T_2 in the OFF state indefinitely, *i.e.*, Q = 1 & Q' = 0. In this condition the bistable multivibrator is said to be in the **Set state**. A positive signal applied to the Reset input at the base of T_2 turns it ON. As we have discussed earlier, in the same sequence T_2 turns ON & T_1 turns OFF, resulting in a second stable state *i.e.*, Q = 0 & Q' = 1. In this condition the bistable multivibrator is said to be in the ***Reset* state.**

LATCHES

The basic difference between a latch & flip-flop is, Storage elements that operate with signal levels (rather than signal transitions) are referred to as **latches**; those controlled by a clock transition are **flip-flops**. Latches are said to be level sensitive devices; flip-flops are edge-sensitive devices.

The two types of storage elements are related because latches are the basic circuits from which all flip-flops are constructed.



We consider the fundamental circuit shown in Fig.(last page). It consists of two inverters G_1 and G_2 (NAND gates are used as inverters). The output of G_1 is connected to the input of G_2 (A_2) and the output of G_2 is connected to the input of G_1 (A_1).

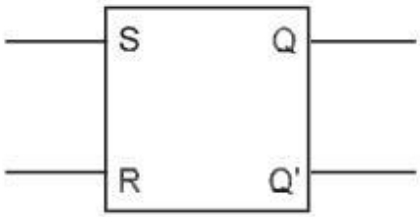
Let us assume the output of G_1 to be Q = 0, which is also the input of G_2 (A_2 = 0). So, the output of G_2 will be Q'= 1, which makes A_1 = 1 and consequently Q = 0 which is according to our assumption. Similarly, we can demonstrate that if Q = 1, then Q' = 0 and this is also consistent with the circuit connections. Hence, we see that Q and Q' are always complementary. And if the circuit is in 1 state, it continues to remain in this state and vice versa is also true. Since this information is locked or latched in this circuit, therefore, this circuit is also referred to as a ***latch***. In this circuit there is no way to enter the desired digital information to be stored in it. To make that possible we have to modify the circuit by replacing the inverters with NAND gates and then it becomes a flip-flop.

TYPES OF FLIP-FLOPS

There are different types of flip-flops depending on how their inputs and clock pulses cause transition between two states. We will discuss four different types of flip-flops in this chapter, *viz*., S-R, D, J-K, and T. Basically D, J-K, and T are three different modifications of the S-R flip-flop.

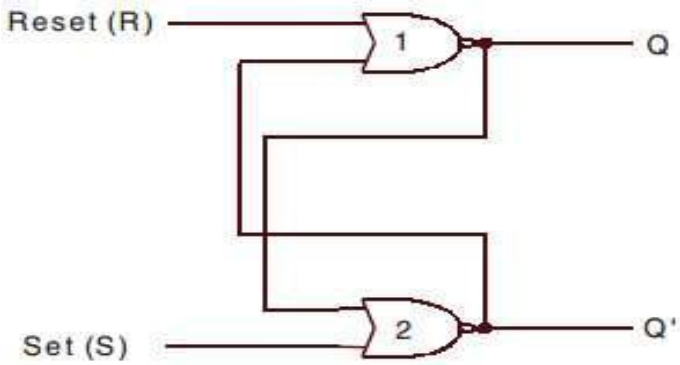
S-R (Set-Reset) Flip-flop

An S-R flip-flop has two inputs named Set (S) and Reset (R), and two outputs Q and Q'. The outputs are complemented of each other, *i.e.*, if one of the outputs is 0 then the other should be 1. This can be implemented using NAND or NOR gates. The block diagram of an S-R flip-flop is shown in Figure below:-



S-R Flip-flop Based on NOR Gates

An S-R flip-flop can be constructed with NOR gates at ease by connecting the NOR gates back-to-back as shown in Figure below. The cross-coupled connections from the output of gate 1 to the input of gate 2 constitute a feedback path. This circuit is not clocked and is classified as an asynchronous sequential circuit. The truth table for the S-R flip-flop based on a NOR gate is shown in the table below



Inputs		*Outputs*		*Action*
S	*R*	Q_{n+1}	Q'_{n+1}	
0	0	Q_n	Q'_n	No change
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	Forbidden (Undefined)
0	0	–	–	Indeterminate

To analyze the circuit of S-R Flip-flop Based on NOR Gates, we have to consider the fact that the output of a NOR gate is 0 if any of the inputs are 1, irrespective of the other input. The output is 1 only if all of the inputs are 0. The outputs for all the possible conditions as shown in the above table are described as follows.

Case 1. For S = 0 and R = 0, the flip-flop remains in its present state (Q_n). It means that the next state of the flip-flop does not change, *i.e.*, $Q_{n+1} = 0$ if $Q_n = 0$ and vice versa. First let us assume that $Qn= 1$ and $Q'_n= 0$.Thus the inputs of NOR gate 2 are 1 and 0, and therefore its output $Q'n+1 = 0$. This output $Q'_{n+1} = 0$ is fed back as the input of NOR gate1, thereby producing a 1 at the output, as both of the inputs of NOR gate 1 are 0 and 0; so $Q_{n+1} = 1$ as originally assumed. Now let us assume the opposite case, *i.e.*, $Q_n = 0$ and $Q'_n = 1$. Thus, the inputs of NOR gate 1 are 1 and 0, and therefore its output $Q'_{n+1} = 0$. This output $Q_{n+1} = 0 =0$ is fed back as the input of NOR gate 2, thereby producing a 1 at the output, as both of the inputs of NOR gate 2 are 0 and 0; so $Q'_{n+1} = 1$ as originally assumed. Thus, we find that the condition S = 0 and R = 0 do not affect the outputs of the flip-flop, which means this is the memory condition of the S-R flip-flop.

Case 2. The second input condition is S = 0 and R = 1. The 1 at R input forces the output of NOR gate 1 to be 0 (*i.e.*, $Q_{n+1} = 0$). Hence both the inputs of NOR gate 2 are 0 and 0 and so its output $Q'_{n+1} = 1$. Thus, the condition S = 0 and R = 1 will always reset the flip-flop to 0. Now if the R returns to 0 with S = 0, the flip-flop will remain in the same state.

Case 3. The third input condition is S = 1 and R = 0. The 1 at S input forces the output of NOR gate 2 to be 0 (*i.e.*, $Q'_{n+1} = 0$). Hence both the inputs of NOR gate 1 are 0 and 0 and so its output $Q_{n+1} = 1$. Thus, the condition S = 1 and R = 0 will always set the flip-flop to 1. Now if the S returns to 0 with R = 0, the flip-flop will remain in the same state.

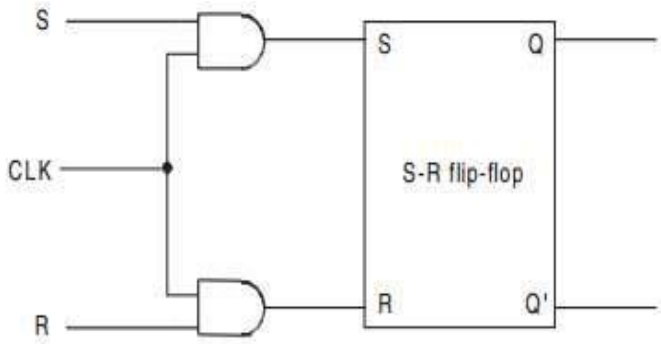
Case 4. The fourth input condition is S = 1 and R = 1. The 1 at R input and 1 at S input forces the output of both NOR gate 1 and NOR gate 2 to be 0. Hence both the outputs of NOR gate 1 and NOR gate 2 are 0 and 0; *i.e.*, $Q_{n+1} = 0$ and $Q'_{n+1} = 0$. Hence this condition S = 1 and R = 1 violates the fact that the outputs of a flip-flop will always be the complement of each other. Since the condition violates the basic definition of flip-flop, it is called the ***undefined*** condition. Generally, this condition must be avoided by making sure that 1s are not applied simultaneously to both of the inputs.

Case 5. If case 4 arises at all, then S and R both return to 0 and 0 simultaneously, and then any one of the NOR gates acts faster than the other and assumes the state. For example, if NOR gate 1 is faster than NOR gate 2, then Q_{n+1} will become 1 and this will make $Q'_{n+1} = 0$. Similarly, if NOR gate 2 is faster than NOR gate 1, then Q'_{n+1} will become 1 and this will make $Q_{n+1} = 0$. Hence, this condition is determined by the flip-flop itself. Since this condition cannot be controlled and predicted it is called the ***indeterminate*** condition.

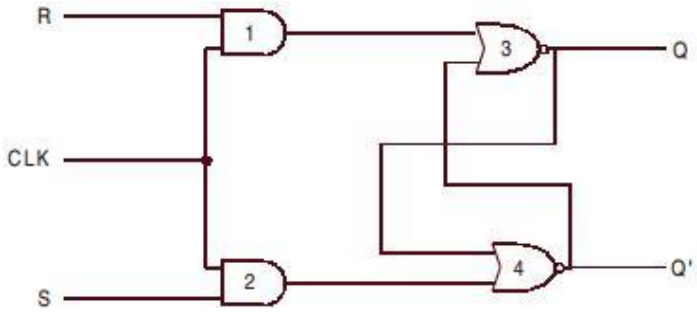
Similarly, we can analyze the case of S'-R' Flip-flop Based on NAND Gates (assignment for the students).

CLOCKED S-R FLIP-FLOP

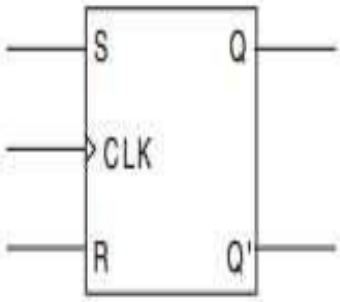
Generally, synchronous circuits change their states only when clock pulses are present. The operation of the basic flip-flop can be modified by including an additional input to control the behavior of the circuit. Such a circuit is shown below:-



The circuit shown above consists of two AND gates. The clock input is connected to both of the AND gates, resulting in LOW outputs when the clock input is LOW. In this situation the changes in S and R inputs will not affect the state (Q) of the flip-flop. On the other hand, if the clock input is HIGH, the changes in S and R will be passed over by the AND gates and they will cause changes in the output (Q) of the flip-flop. This way, any information, either 1 or 0, can be stored in the flip-flop by applying a HIGH clock input and be retained for any desired period of time by applying a LOW at the clock input. This type of flip-flop is called a ***clocked S-R flip-flop***. Such a clocked S-R flip-flop made up of two AND gates and two NOR gates is shown in Figure below:-



The logic symbol of the S-R flip-flop is shown below. It has three inputs: S, R, and CLK. The CLK input is marked with a small triangle. The triangle is a symbol that denotes the fact that the circuit responds to an edge or transition at CLK input.



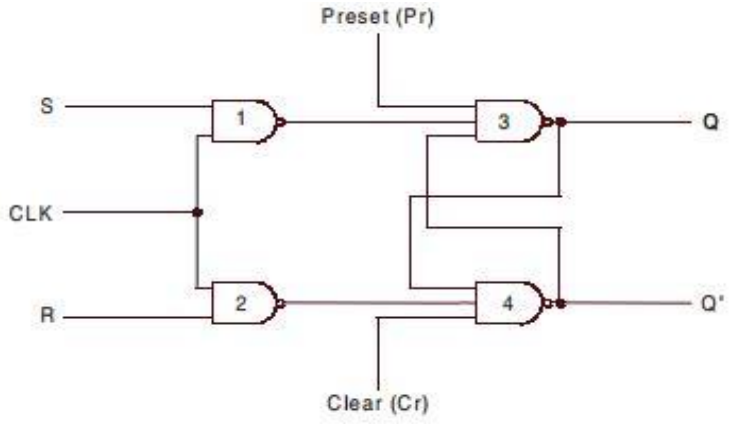
Inputs		*Output*
S_n	R_n	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	–

Assuming that the inputs do not change during the presence of the clock pulse, we can express the working of the S-R flip-flop in the form of the truth table shown here. Here, S_n and R_n denote the inputs and Q_n denotes the output during the bit time n. Q_{n+1} denotes the output after the pulse passes *i.e.*, in the bit time $n + 1$.

Case 1. If $S_n = R_n = 0$, and the clock pulse is not applied, the output of the flip-flop remains in the present state. Even if $S_n = R_n = 0$, and the clock pulse is applied, the output at the end of the clock pulse is the same as the output before the clock pulse, *i.e.*, $Q_{n+1} = Q_n$. The first row of the table indicates that situation. **Case 2.** For $S_n = 0$ and $R_n = 1$, if the clock pulse is applied (*i.e.*, CLK = 1), the output of NAND gate 1 becomes 1; whereas the output of NAND gate 2 will be 0. Now a 0 at the input of NAND gate 4 forces the output to be 1 *i.e.*, Q' = 1. This 1 goes to the input of NAND gate 3 to make both the inputs of NAND gate 3 as 1, which forces the output of NAND gate 3 to be 0, *i.e.*, Q = 0. **Case 3.** For $S_n = 1$ and $R_n = 0$, if the clock pulse is applied (*i.e.*, CLK = 1), the output of NAND gate 2 becomes 1; whereas the output of NAND gate 1 will be 0. Now a 0 at the input of NAND gate 3 forces the output to be 1, *i.e.*, Q = 1. This 1 goes to the input of NAND gate 4 to make both the inputs of NAND gate 4 as 1, which forces the output of NAND gate 4 to be 0, *i.e.*, Q' = 0. **Case 4.** For $S_n = 1$ and $R_n = 1$, if the clock pulse is applied (*i.e.*, CLK = 1), the outputs of both NAND gate 2 and NAND gate 1 becomes 0. Now a 0 at the input of both NAND gate 3 and NAND gate 4 forces the outputs of both the gates to be 1, *i.e.*, Q = 1 and Q' = 1. When the CLK input goes back to 0 (while S and R remain at 1), it is not possible to determine the next state, as it depends on whether the output of gate 1 or gate 2 goes to 1 first.

Preset and Clear

Till now the flip-flops we discussed there when the power is switched on, the state of the circuit is uncertain. It may come to reset (Q = 0) or set (Q = 1) state. But in many applications, it is required to initially set or reset the flip-flop., *i.e.*, the initial state of the flip-flop is to be assigned. This is done by using the direct or asynchronous inputs. These inputs are referred to as ***preset*** **(Pr)** and ***clear*** **(Cr)** inputs. These inputs may be applied at any time between clock pulses and is not in synchronism with the clock. Such an S-R flip-flop containing preset and clear inputs is shown in Figure below.



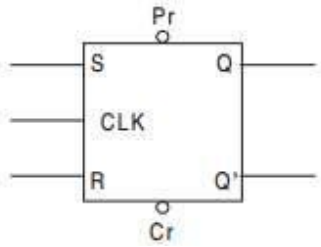
From the above Figure, we see that if Pr = Cr = 1, the circuit operates according to the table of clocked S-R flip-flop as we discussed just before.

If Pr = 1 and Cr = 0, the output of NAND gate 4 is forced to be 1, *i.e.*, Q' = 1 and the flip-flop is reset, overwriting the previous state of the flip-flop.

If Pr = 0 and Cr = 1, the output of NAND gate 3 is forced to be 1, *i.e.*, Q = 1 and the flip-flop is set, overwriting the previous state of the flip-flop. Once the state of the flip-flop is established asynchronously, the inputs Pr and Cr must be connected to logic 1 before the next clock is applied.

The condition Pr = Cr = 0 must not be applied since this leads to an uncertain state.

The logic symbol of an S-R flip-flop with Pr and Cr inputs is shown in the side. Here, bubbles are used for Pr and Cr inputs, which indicate these are active low inputs, which means that the intended function is performed if the signal applied to Pr and Cr is LOW. The operation of the clocked S-R flip-flop is shown in the table below. The circuit can be designed such that the asynchronous inputs override the clock, *i.e.*, the circuit can be set or reset even in the presence of the clock pulse.



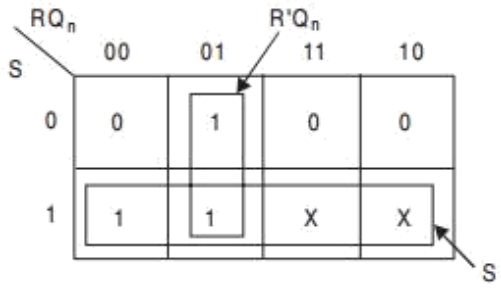
Inputs			*Output*	*Operation*
CLK	*Cr*	*Pr*	*Q*	*performed*
1	1	1	Q_{n+1} (Figure 7.3)	Normal flip-flop
0	1	0	1	Preset
0	0	1	0	Clear
0	0	0	–	Uncertain

Characteristic Table of an S-R Flip-flop

From the name itself it is very clear that the *characteristic table* of a flip-flop actually gives us an idea about the character, *i.e.*, the working of the flip-flop. Now, from all our above discussions, we know that the next state flip-flop output (Q_{n+1}) depends on the present inputs as well as the present output (Q_n). So, in order to know the next state output of a flip-flop, we have to consider the present state output also. The characteristic table of an S-R fl ip-flop is given in the table below. From the characteristic table we have to find out the characteristic equation of the S-R flip-flop.

Flip-flop inputs		*Present output*	*Next output*
S	*R*	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

Now we will find out the characteristic equation of the S-R flip-flop from the characteristic table with the help of the Karnaugh map:-



From the Karnaugh map above we find the expression for Q_{n+1} as

$$Q_{n+1} = S + R'Q_n$$

Along with the above equation we have to consider the fact that S and R cannot be simultaneously 0. In order to take that fact into account we have to incorporate another equation for the S-R flip-flop. The equation is given below.

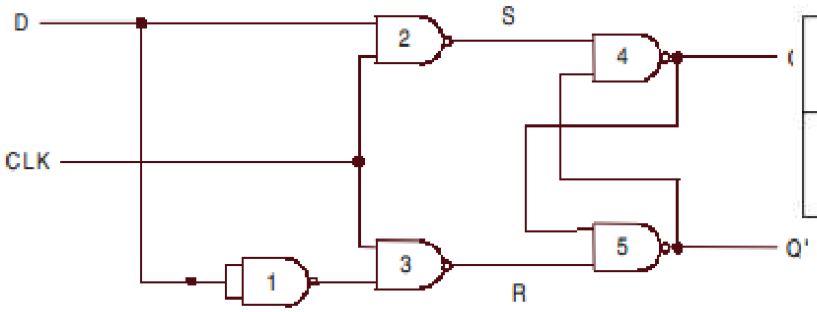
$$SR = 0$$

Hence the characteristic equations of an S-R flip-flop are $Q_{n+1} = S + R' Q_n$

$SR = 0$

CLOCKED D FLIP-FLOP

One way to eliminate the undesirable condition of the indeterminate state in the *SR* latch is to ensure that inputs *S* and *R* are never equal to 1 at the same time. This is done in the *D* latch. The D flip-flop has only one input referred to as the D (**data**) input & two outputs as usual Q and Q'. It transfers the data at the input after the delay of one clock pulse at the output Q. So, in some cases the input is referred to as a delay input and the flip-flop gets the name ***delay*** (D) flip-flop. It can be easily constructed from an S-R flip-flop by simply incorporating an inverter between S and R such that the input of the inverter is at the S end & the output of the inverter is at the R end. We can get rid of the undefined condition, *i.e.*, S = R = 1 condition, of the S-R flip-flop in the D flip flop. The D flip-flop is either used as a delay device or as a latch to store one bit of binary information. The truth table of D flip-flop is given in the table below. The structure of the D flip-flop is shown in Figure below, which is being constructed using NAND gates. The same structure can be constructed using only NOR gates.



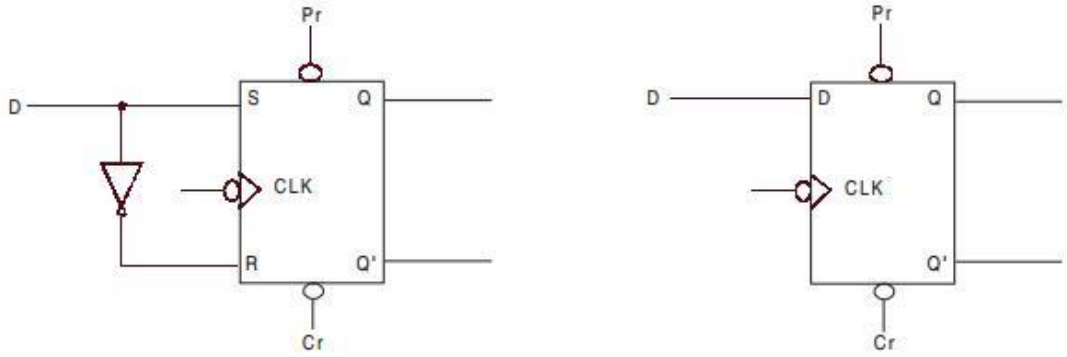
Input D_n	*Output* Q_{n+1}
0	0
1	1

Case 1. If the CLK input is low, the value of the D input has no effect since the S and R inputs of the basic NAND flip-flop are kept as 1.

Case 2. If the CLK = 1 and D = 1, the NAND gate 1 produces 0, which forces the output of NAND gate 3 as 1. On the other hand, both the inputs of NAND gate 2 are 1, which gives the output of gate 2 as 0. Hence, output

of NAND gate 4 is forced to be 1, *i.e.*, Q = 1, whereas both the inputs of gate 5 are 1 and the output is 0, *i.e.*, Q' = 0. Hence, we find that when D = 1, after one clock pulse passes Q = 1, which means the output follows D.
Case 3. If the CLK = 1, and D = 0, the NAND gate 1 produces 1. Hence both the inputs of NAND gate 3 are 1, which gives the output of gate 3 as 0. On the other hand, D = 0 forces the output of NAND gate 2 to be 1. Hence the output of NAND gate 5 is forced to be 1, *i.e.*, Q' = 1, whereas both the inputs of gate 4 are 1 and the output is 0, *i.e.*, Q = 0. Hence, we find that when D = 0, after one clock pulse passes Q = 0, which means the output again follows D.

A simple way to construct a D flip-flop using an S-R flip-flop is shown in Figure below. The logic symbol of a D flip-flop is shown in Figure below. A D flip-flop is most often used in the construction of sequential circuits like registers.



Characteristic Table of a D Flip-flop

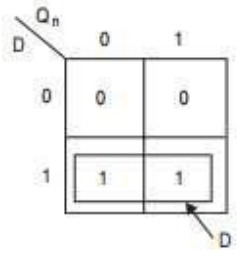
As we have already discussed the characteristic equation of an S-R flip-flop, we can similarly find out the characteristic equation of a D flip-flop. The characteristic table of a D flip-flop is given in the table below. From the characteristic table we have to find out the characteristic equation of the D flip-flop.

Flip-flop inputs	*Present output*	*Next output*
D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

Now we will find out the characteristic equation of the D flip-flop from the characteristic table with the help of the Karnaugh map:-

Hence, the characteristic equation of a D flip-flop is

Q_{n+1} = D

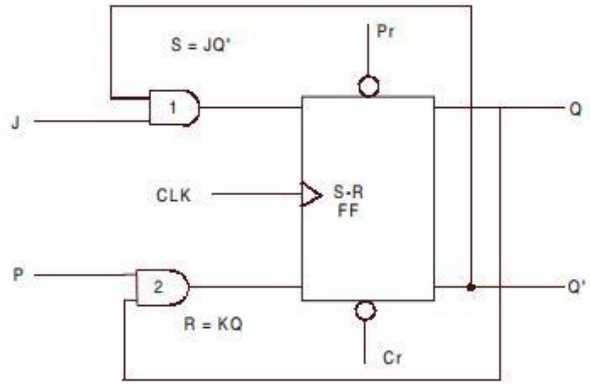


J-K FLIP-FLOP

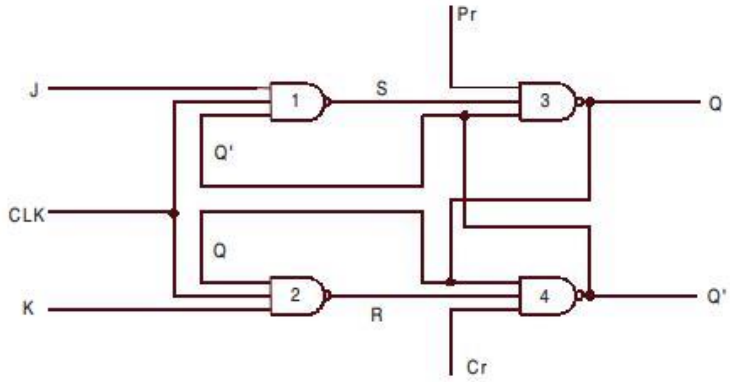
A J-K flip-flop has very similar characteristics to an S-R flip-flop. The only difference is that the undefined condition for an S-R flip-flop, *i.e.*, $S_n = R_n = 1$ condition, is also included in this case. Inputs J and K behave like inputs S and R to set and reset the flip-flop respectively. When J = K = 1, the flip-flop is said to be in a ***toggle state***, which means the output switches to its complementary state every time a clock passes.

The data inputs are J and K, which are ANDed with Q' and Q respectively to obtain the inputs for S and R respectively. A J-K flip-flop thus obtained is shown in Figure below.

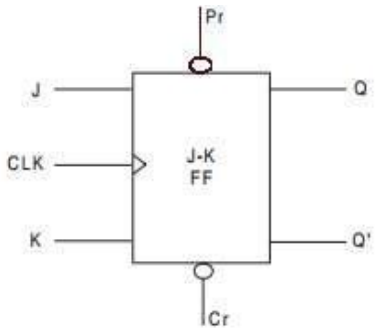
An S-R flip-flop converted into a J-K flip-flop:-



A J-K flip-flop using NAND gates:-

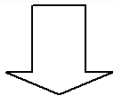


Logic symbol of a J-K flip-flop:-



The TRUTH table for JK flip-flop is:-

Data inputs		*Outputs*		*Inputs to S-R FF*		*Output*
J_n	K_n	Q_n	Q'_n	S_n	R_n	Q_{n+1}
0	0	0	1	0	0	0
0	0	1	0	0	0	1
0	1	0	1	0	0	0
0	1	1	0	0	1	0
1	0	0	1	1	0	1
1	0	1	0	0	0	1
1	1	0	1	1	0	1
1	1	1	0	0	1	0



Inputs		*Output*
J_n	K_n	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	Q'_n

Case 1. When the clock is applied and J = 0, whatever the value of Q'$_n$ (0 or 1), the output of NAND gate 1 is 1. Similarly, when K = 0, whatever the value of Q$_n$ (0 or 1), the output of gate 2 is also 1. Therefore, when J = 0 and K = 0, the inputs to the basic flip-flop are S = 1 and R = 1. This condition forces the flip-flop to remain in the same state.

Case 2. When the clock is applied and J = 0 and K = 1 & the previous state of the flip-flop is reset (*i.e.*, Q$_n$ = 0 and Q'$_n$ = 1), then S = 1 and R = 1. Since S = 1 and R = 1, the basic flip-flop does not alter the state and remains in the reset state. But if the flip-flop is in set condition (*i.e.*, Q$_n$ = 1 & Q'$_n$ = 0), then S = 1 and R = 0. Since S = 1 and R = 0, the basic flip-flop changes its state and resets.

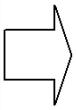
Case 3. When the clock is applied and J = 1 and K = 0 and the previous state of the flip-flop is reset (*i.e.*, Q$_n$ = 0 and Q'$_n$ = 1), then S = 0 and R = 1. Since S = 0 and R = 1, the basic flip-flop changes its state and goes to the set state. But if the flip-flop is already in set condition (*i.e.*, Q$_n$ = 1 and Q'$_n$ = 0), then S = 1 and R = 1. Since S = 1 and R = 1, the basic flip-flop does not alter its state and remains in the set state.

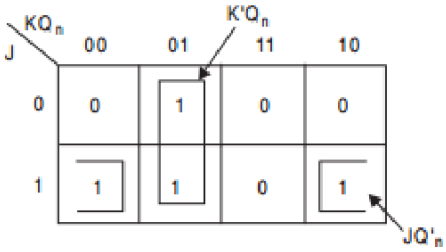
Case 4. When the clock is applied and J = 1 and K = 1 and the previous state of the flip-flop is reset (*i.e.*, Q$_n$ = 0 and Q'$_n$ = 1), then S = 0 and R = 1. Since S = 0 and R = 1, the basic flip-flop changes its state and goes to the set state. But if the flip-flop is already in set condition (*i.e.*, Q$_n$ = 1 and Q'$_n$ = 0), then S = 1 and R = 0. Since S = 1 and R = 0, the basic flip-flop changes its state and goes to the reset state. So, we find that for J = 1 and K = 1, the flip-flop toggles its state from *set* to *reset* and vice versa. Toggle means to switch to the opposite state.

Characteristic Table of a J-K Flip-flop

As we have already discussed the characteristic equation of an S-R flip-flop, we can similarly find out the characteristic equation of a J-K flip-flop. The characteristic table of a J-K flip-flop is given in the table below. From the characteristic table we have to find out the characteristic equation of the J-K flip-flop.

Flip-flop inputs		*Present output*	*Next output*
J	*K*	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0



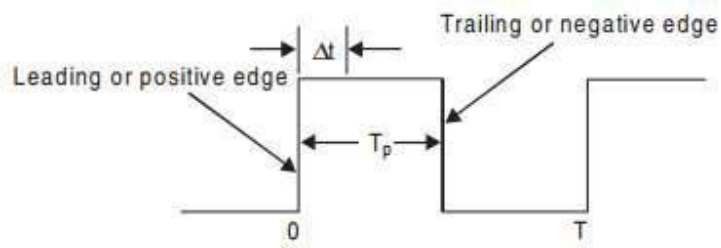


From the Karnaugh map, we obtain $\mathbf{Q_{n+1} = JQ'_n + K'Q_n}$.
Hence, the characteristic equation of a J-K flip-flop is

$$\mathbf{Q_{n+1} = JQ'_n + K'Q_n}$$

Race-around Condition of a J-K Flip-flop

The inherent difficulty of an S-R flip-flop (*i.e.*, S = R = 1) is eliminated by using the feedback connections from the outputs to the inputs of gate 1 and gate 2 as discussed in JK flip-flop. Truth tables JK flip-flop were formed with the assumption that the inputs do not change during the clock pulse (CLK = 1). But the consideration is not true because of the feedback connections. Consider, for example, that the inputs are J = K = 1 and Q = 1, and a pulse as shown in Figure below is applied at the clock input.



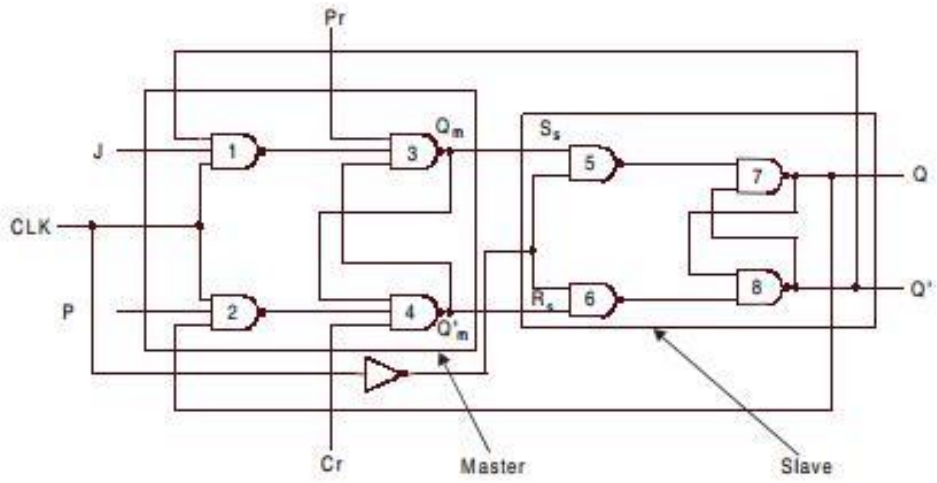
Consider, for example, that the inputs are J = K = 1 and Q = 1, and a pulse as shown above is applied at the clock input. After a time, interval Δt equal to the propagation delay through two NAND gates in series, the outputs will change to Q = 0. So now we have J = K = 1 and Q = 0. After another time interval of Δt the output will change back to Q = 1. Hence, we conclude that for the time duration of t_p of the clock pulse, the output will oscillate between 0 and 1. Hence, at the end of the clock pulse, the value of the output is not certain. This situation is referred to as a ***race-around condition***.

Generally, the propagation delay of TTL gates is of the order of nanoseconds. So, if the clock pulse is of the order of microseconds, then the output will change thousands of times within the clock pulse. This race-around condition can be avoided if $t_p < \Delta t < T$. Due to the small propagation delay of the ICs it may be difficult to satisfy the above condition. A more practical way to avoid the problem is to use the master-slave (M-S) configuration as discussed below.

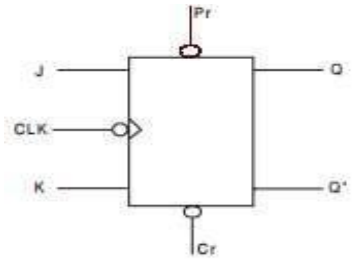
Master-Slave J-K Flip-flop

A master-slave (M-S) flip-flop is shown in Figure below. Basically, a master-slave flip-flop is a system of two flip-flops—one being designated as *master* and the other is the *slave*. From the figure below we see that a clock pulse is applied to the master and the inverted form of the same clock pulse is applied to the slave.

When CLK = 1, the first flip-flop (*i.e.*, the master) is enabled and the outputs Q_m and Q'_m respond to the inputs J and K according to the table shown in Figure 7.13. At this time the second flip-flop (*i.e.*, the slave) is disabled because the CLK is LOW to the second flip-flop. Similarly, when CLK becomes LOW, the master becomes disabled and the slave becomes active, since now the CLK to it is HIGH. Therefore, the outputs Q and Q' follow the outputs Q_m and Q'_m respectively. Since the second flip-flop just follows the first one, it is referred to as a slave and the first one is called the master. Hence, the configuration is referred to as a master-slave (M-S) flip-flop.



In this type of circuit configuration, the inputs to the gates 5 and 6 do not change at the time of application of the clock pulse. Hence the race-around condition does not exist. The state of the master-slave flip-flop, shown in above Figure, changes at the negative transition (trailing edge) of the clock pulse. Hence, it becomes negative triggering a master-slave flip-flop. This can be changed to a positive edge triggering flip-flop by adding two inverters to the system—one before the clock pulse is applied to the master and an additional one in between the master and the slave. The logic symbol of a negative edge master-slave is shown in Figure below.



The system of master-slave flip-flops is not restricted to J-K master-slave only. There may be an S-R master-slave or a D master-slave, etc., in all of them the slave is an S-R flip-flop, whereas the master changes to J-K or S-R or D flip-flops.

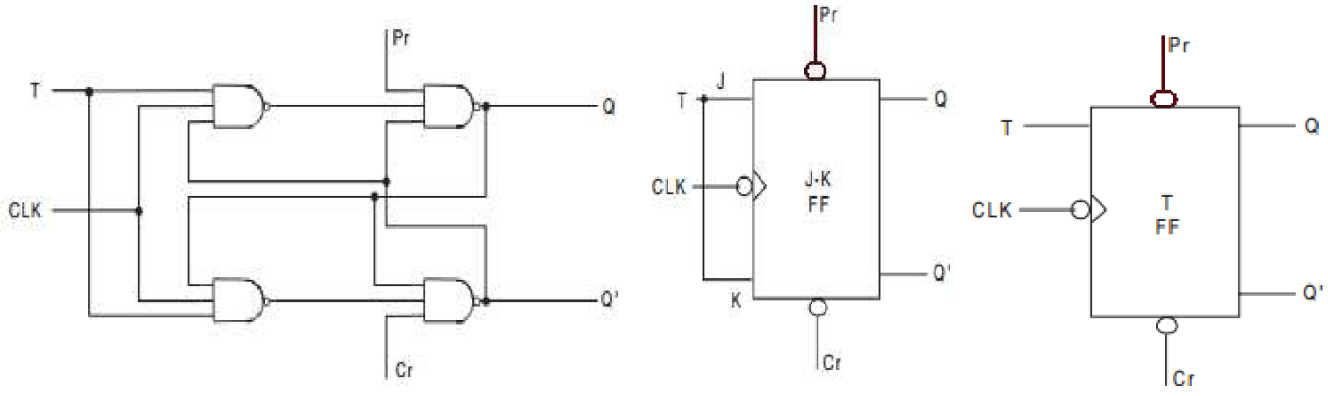
T Flip-flop

With a slight modification of a J-K flip-flop, we can construct a new flip-flop called a T flip-flop. If the two inputs J and K of a J-K flip-flop are tied together it is referred to as a T flip-flop. Hence, a T flip-flop has only one input T and two outputs Q and Q'. The name T flip-flop actually indicates the fact that the flip-flop has the ability to toggle. It has actually only two states—***toggle state*** **and** ***memory state***. Since there are only two states, a T flip-flop is a very good option to use in counter design and in sequential circuits design where switching an operation is required. The truth table of a T flip-flop is given below:-

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

If the T input is in 0 state (*i.e.*, J = K = 0) prior to a clock pulse, the Q output will not change with the clock pulse. On the other hand, if the T input is in 1 state (*i.e.*, J = K = 1) prior to a clock pulse, the Q output will change to Q' with the clock pulse. In other words, we may say that, if T = 1 and the device is clocked, then the output toggles its state.

The truth table shows that when T = 0, then $Q_{n+1} = Q_n$, *i.e.*, the next state is the same as the present state and no change occurs. When T = 1, then $Q_{n+1} = Q'_n$, *i.e.*, the state of the flip-flop is complemented. The circuit diagram of a T flip-flop and the block diagram of the T flip-flop is shown below:-

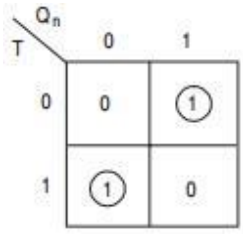


Characteristic Table of a T Flip-flop

As we have already discussed the characteristic equation of a J-K flip-flop, we can similarly find out the characteristic equation of a T flip-flop. The characteristic table of a T flip-flop is given below. From the characteristic table we have to find out the characteristic equation of the T flip-flop.

Flip-flop inputs	*Present output*	*Next output*
T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Now we will find out the characteristic equation of the T flip-flop from the characteristic table with the help of the Karnaugh map below:-



From the Karnaugh map, the Boolean expression of Q_{n+1} is derived as $Q_{n+1} = TQ'_n + T'Q_n$. Hence, the characteristic equation of a T flip-flop is

$$Q_{n+1} = TQ'_n + T'Q_n$$

TRIGGERING OF FLIP-FLOPS

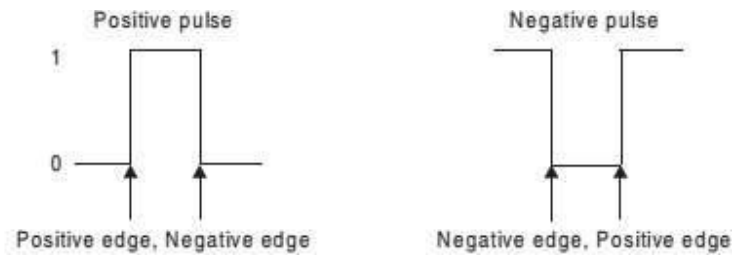
Flip-flops are synchronous sequential circuits. This type of circuit works with the application of a synchronization mechanism, which is termed a *clock*. Based on the specific interval or point in the clock during or at which triggering of the flip-flop takes place, it can be classified into two different types—***level triggering*** **and** ***edge triggering***. A clock pulse starts from an initial value of 0, goes momentarily to 1, and after a short interval, returns to the initial value.

Level Triggering of Flip-flops

If a flip-flop gets enabled when a clock pulse goes HIGH and remains enabled throughout the duration of the clock pulse remaining HIGH, the flip-flop is said to be a *level triggered flip-flop*. If the flip-flop changes its state when the clock pulse is positive, it is termed as a *positive level triggered flip-flop*. On the other hand, if a NOT gate is introduced in the clock input terminal of the flip-flop, then the flip-flop changes its state when the clock pulse is negative, it is termed as a *negative level triggered flip-flop.* The main drawback of level triggering is that, as long as the clock pulse is active, the flip-flop changes its state more than once or many times for the change in inputs. If the inputs do not change during one clock pulse, then the output remains stable. On the other hand, if the frequency of the input change is higher than the input clock frequency, the output of the flip-flop undergoes multiple changes as long as the clock remains active. This can be overcome by using either master-slave flip-flops or the edge-triggered flip-flop.

Edge-triggering of Flip-flops

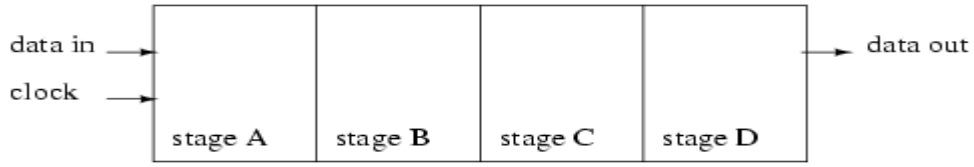
A clock pulse goes from 0 to 1 and then returns from 1 to 0. The Figure below shows the two transitions, and they are defined as the *positive edge* (0 to 1 transition) and the *negative edge* (1 to 0 transition). The term *edge-triggered* means that the flip-flop changes its state only at either the positive or negative edge of the clock pulse.



Shift registers produce a discrete delay of a digital signal or waveform. A waveform synchronized to a *clock*, a repeating square wave, is delayed by **"n"** discrete clock times, and where **"n"** is the number of shift register stages. Thus, a four-stage shift register delays "data in" by four clocks to "data out". The stages in a shift register are *delay stages*, typically type **"D"** Flip-Flops or type **"JK"** Flip-flops.

Basic shift registers are classified by structure according to the following types:

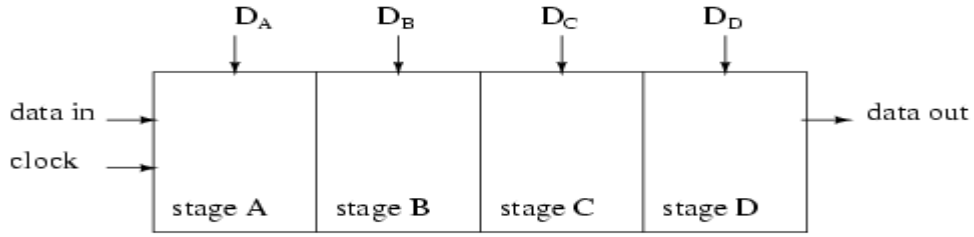
- Serial-in/serial-out
- Parallel-in/serial-out
- Serial-in/parallel-out
- Universal parallel-in/parallel-out
- Ring counter



Serial-in, serial-out shift register with 4-stages

Above we show a block diagram of a serial-in/serial-out shift register, which is 4-stages long. Data at the input will be delayed by four clock periods from the input to the output of the shift register.

Data at "data in", above, will be present at the Stage **A** output after the first clock pulse. After the second pulse stage **A** data is transferred to stage **B** output, and "data in" is transferred to stage **A** output. After the third clock, stage **C** is replaced by stage **B**; stage **B** is replaced by stage **A**; and stage A is replaced by "data in". After the fourth clock, the data originally present at "data in" is at stage **D**, "output". The "first in" data is "first out" as it is shifted from "data in" to "data out".

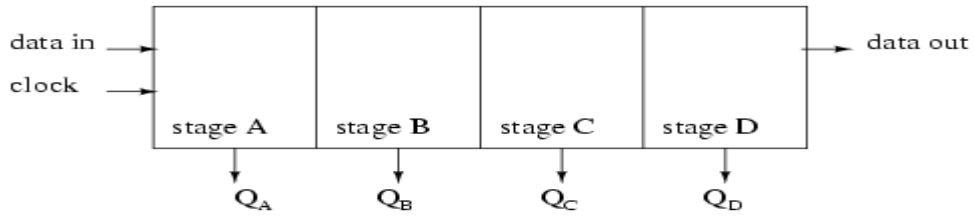


Parallel-in, serial-out shift register with 4-stages

Data is loaded into all stages at once of a parallel-in/serial-out shift register. The data is then shifted out via "data out" by clock pulses. Since a 4- stage shift register is shown above, four clock pulses are required to shift out all of the data. In the diagram above, stage **D** data will be present at the "data out" up until the first clock pulse; stage **C** data will be present at "data out" between the first clock and the second clock pulse; stage **B** data will be present between

the second clock and the third clock; and stage **A** data will be present between the third and the fourth clock. After the fourth clock pulse and thereafter, successive bits of "data in" should appear at "data out" of the shift register after a delay of four clock pulses.

If four switches were connected to D_A through D_D, the status could be read into a microprocessor using only one data pin and a clock pin. Since adding more switches would require no additional pins, this approach looks attractive for many inputs.



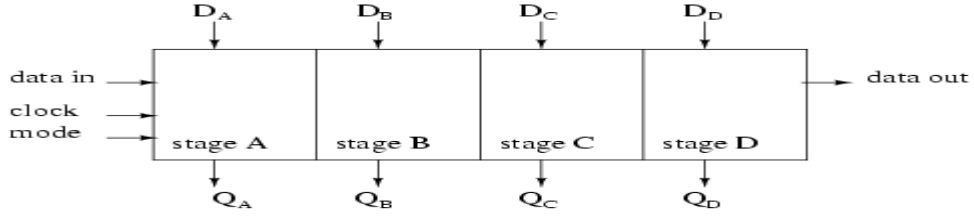
Serial-in, parallel-out shift register with 4-stages

Above, four data bits will be shifted in from "data in" by four clock pulses and be available at Q_A through Q_D for driving external circuitry such as LEDs, lamps, relay drivers, and horns.

After the first clock, the data at "data in" appears at Q_A. After the second clock, The old Q_A data appears at Q_B; Q_A receives next data from "data in". After the third clock, Q_B data is at Q_C. After the fourth clock, Q_C data is at Q_D. This stage contains the data first present at "data in". The shift register should now contain four data bits.

A parallel-in/parallel-out shift register combines the function of the parallel-in, serial-out shift registers with the function of the serial-in, parallel-out shift register to yield the universal shift register. The "do anything" shifter comes at a price– the increased number of I/O (Input/Output) pins may reduce the number of stages which can be packaged.

Data presented at D_A through D_D is parallel loaded into the registers. This data at Q_A through Q_D may be shifted by the number of pulses presented at the clock input. The shifted data is available at Q_A through Q_D. The "mode" input, which may be more than one input, controls parallel loading of data from D_A through D_D, shifting of data, and the direction of shifting.



Parallel-in, parallel-out shift register with 4-stages

If the serial output of a shift register is connected to the serial input, data can be perpetually shifted around the ring as long as clock pulses are present. If the output is inverted before being fed back as shown above, we do not have to worry about loading the initial data into the "ring counter"