Unit 05: Logic Circuits & Flip-Flops

Logic Circuits:

1. **Difference between Combinational & Sequential circuits**
2. **Code converters (BCD, Excess-3 and Gray)**
3. **Multiplexers and De-multiplexers**
4. **Decoders.**

Flip Flops:

1. **SR flip-flop,**
2. **JK flip-flop,**
3. **D flip-flop**
4. **T flip-flop.**

Difference between Combinational & Sequential Circuits

SN	Combinational Circuits	Sequential Circuits
1.	In combinational circuits, output depends only upon present input	Output depends upon present as well as past input
2.	Memory unit is not required	Memory unit is required
3.	These circuits are faster	These circuits are slower
4.	These are easy to design	Comparatively harder to design
5.	Logic gates are basic blocks	Flip flops are basic blocks
6.	Mainly, used for arithmetic as well as Boolean operations	Mainly used for storing data
7.	As combinational circuits don't have clock, they don't require triggering	As sequential circuits are clock dependent they need triggering
Ex.	Adder and Subtractor, Multiplexer, Demultiplexer, Decoder etc.	Flipflops, Counters, Registers etc.

Code Converters

1. BCD Code

BCD is Binary Coded Decimal number, where each digit (0-9) of a decimal number is represented by its equivalent binary number. That means, LSB of a decimal number is represented by its equivalent binary number and similarly other higher significant bits of decimal number are also represented by their equivalent binary numbers. For ex., BCD Code of 142 is ➔ 0001 0100 0010

Steps for the Binary to BCD Conversion

1. Convert the binary number to decimal.
2. Convert decimal number to BCD.

Example – Convert $(11101)_2$ to BCD

Step 1: Converting given binary number to decimal code

$$(11101)_2 = 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 16 + 8 + 4 + 1$$

$$(11101)_2 = 29_{10}$$

Step 2: Convert to BCD [Convert each digit into groups of 4 binary digits]

29 ➔ 0010 1001

4-bit Binary to BCD Code Converter

Decimal Number	4-bit Binary Number				BCD Output				
	B4	B3	B2	B1	A	B	C	D	E
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	0	1	1
4	0	1	0	0	0	0	1	0	0
5	0	1	0	1	0	0	1	0	1
6	0	1	1	0	0	0	1	1	0
7	0	1	1	1	0	0	1	1	1
8	1	0	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	0	1
10	1	0	1	0	1	0	0	0	0
11	1	0	1	1	1	0	0	0	1
12	1	1	0	0	1	0	0	1	0
13	1	1	0	1	1	0	0	1	1
14	1	1	1	0	1	0	1	0	0
15	1	1	1	1	1	0	1	0	1

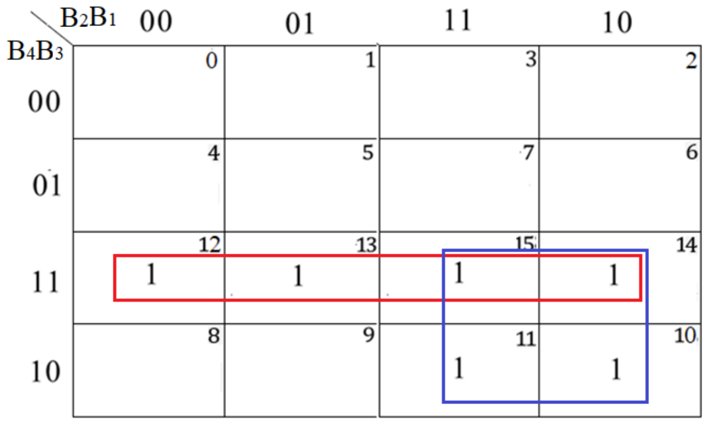
- The input is 4-bit binary. Thus, 16 combinations are possible
- Since, the input is of four bit (i.e., maximum of two decimal digits), the output has to be 8-bit. But since, first three bits will be always 0 for all combinations of inputs, the output can be treated as 5-bit
- From the conversion table, we observe the expressions for BCD outputs are as follows:

$$A = \sum m(10,11,12,13,14,15)$$
$$B = \sum m(8,9)$$
$$C = \sum m(4,5,6,7,14,15)$$
$$D = \sum m(2,3,6,7,12,13)$$
$$E = \sum m(1,3,5,7,9,11,13,15)$$

- By drawing K-maps for the outputs and minimizing them, the minimal expressions for BCD outputs A, B, C, D, E in terms of 4-bit binary inputs B_4, B_3, B_2, B_1 can be obtained.

1. K-map for A:

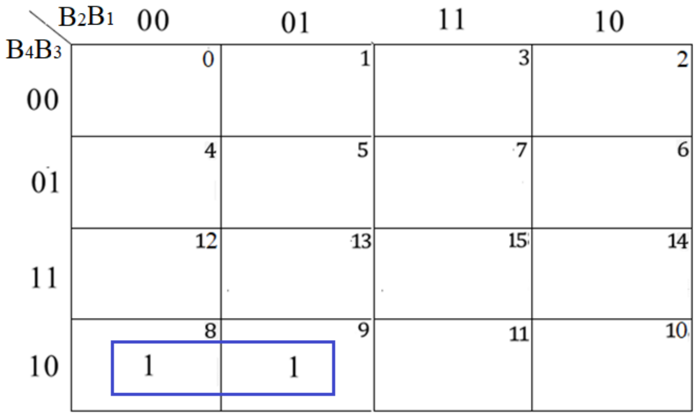
$$A = \sum m(10,11,12,13,14,15)$$



$$A = B_4B_3 + B_4B_2$$

2. K-map for B:

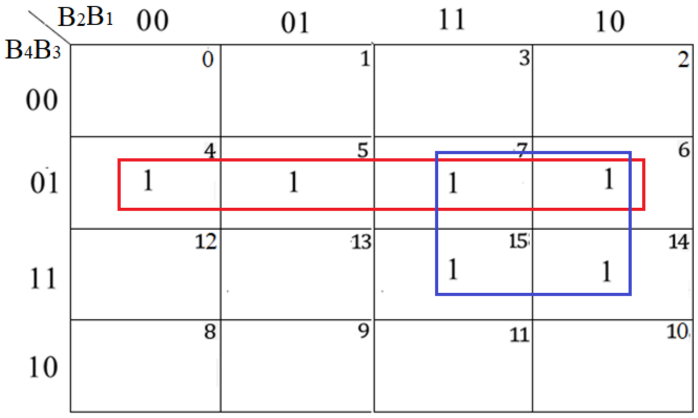
$$B = \sum m(8,9)$$



$$B = B_4\overline{B}_3\overline{B}_2$$

3. K-map for C:

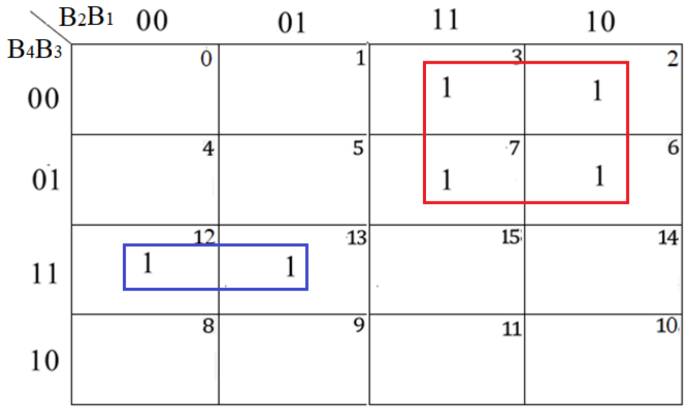
$$C = \sum m(4,5,6,7,14,15)$$



$$C = \overline{B}_4B_3 + B_3B_2$$

4. K-map for D:

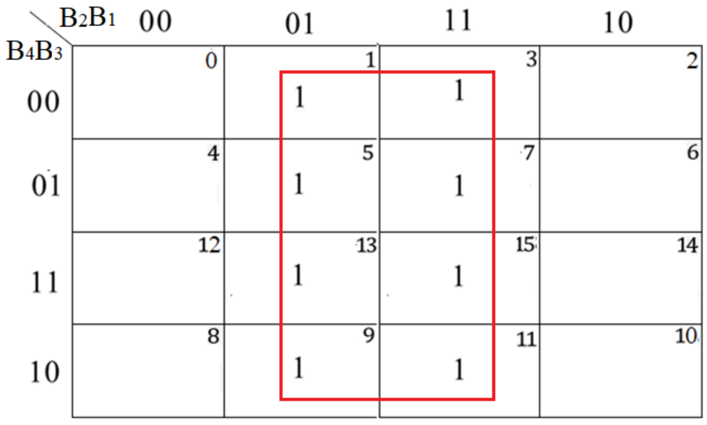
$$D = \sum m(2,3,6,7,12,13)$$



$$D = \overline{B}_4 B_2 + B_4 B_3 \overline{B}_2$$

5. K-map for E:

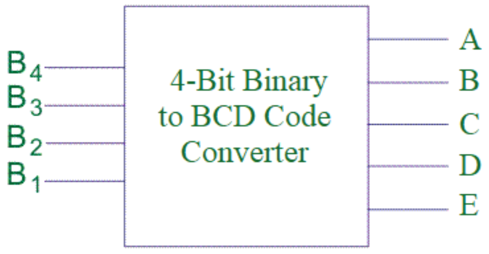
$$E = \sum m(1,3,5,7,9,11,13,15)$$



$$E = B_1$$

Thus, minimal expressions for BCD outputs A, B, C, D, E in terms of 4-bit binary inputs B_4, B_3, B_2, B_1 are as follows:

$A = B_4B_3 + B_4B_2$
$B = B_4\overline{B}_3\overline{B}_2$
$C = \overline{B}_4B_3 + B_3B_2$
$D = \overline{B}_4B_2 + B_4B_3\overline{B}_2$
$E = B_1$



Block diagram of 4-bit binary to BCD code converter

2. Excess-3 (XS-3) Code

The Excess-3 code is a non-weighted code. This code derives his name from the fact that each binary code word is the corresponding 8421 code word plus 0011 (3)
Example: Represent following decimal numbers in Excess-3 code

1] 7
➔ BCD equivalent of 7 is 0111
Corresponding Excess-3 code can be obtained by adding 0011 to its BCD
Thus, 0111 + 0011 = 1010

2] 27
➔ BCD equivalent of 27 is 0010 0111
Corresponding Excess-3 code can be obtained by adding 0011 to each BCD

```
Thus,   0010 0111
      + 0011 0011
       -----------
         0101 1010
```

4-bit Binary to Excess-3 Code Converter

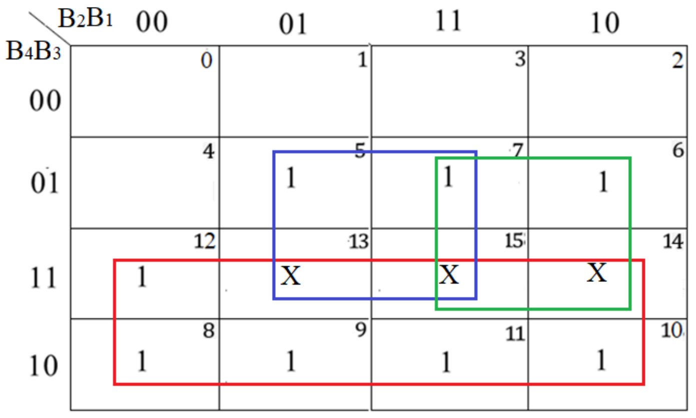
Decimal Number	4-bit Binary Number				Excess-3 Output			
	B4	B3	B2	B1	E4	E3	E2	E1
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0
10	1	0	1	0	1	1	0	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	1	1	1
13	1	1	0	1	X	X	X	X
14	1	1	1	0	X	X	X	X
15	1	1	1	1	X	X	X	X

- The input is 4-bit binary. Thus, 16 combinations are possible
- For last three combinations of binary input (1101, 1110 and 1111), the excess-3 code is invalid and hence, these are shown by don't care (X) conditions. These don't care conditions can be combined with 1s while doing minimization using K-map.
- From conversion table, we observe the expressions for Excess-3 outputs are as follows:

$$E_4 = \sum m(5,6,7,8,9,10,11,12) + d(13,14,15)$$
$$E_3 = \sum m(1,2,3,4,9,10,11,12) + d(13,14,15)$$
$$E_2 = \sum m(0,3,4,7,8,11,12) + d(13,14,15)$$
$$E_1 = \sum m(0,2,4,6,8,10,12) + d(13,14,15)$$

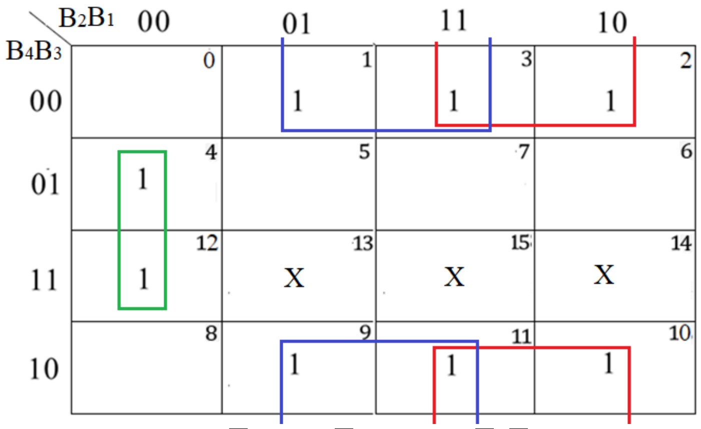
- By drawing K-maps for the outputs and minimizing them, the minimal expressions for Excess-3 outputs E_4, E_3, E_2, E_1 in terms of 4-bit binary inputs B_4, B_3, B_2, B_1 can be obtained.

1. K-map for E_4: $E_4 = \sum m(5,6,7,8,9,10,11,12) + d(13,14,15)$



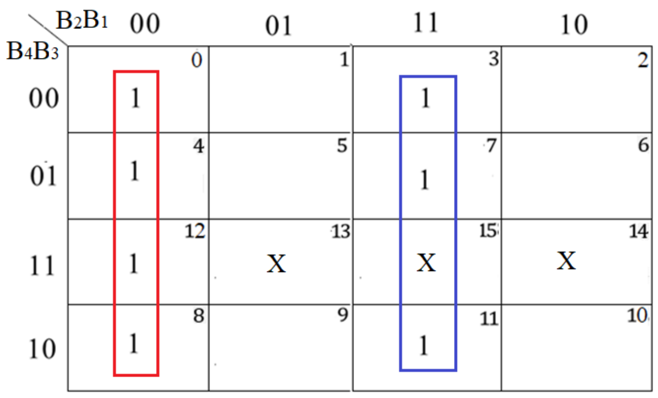
$$E_4 = B_4 + B_3B_1 + B_3B_2$$

2. K-map for E_3: $E_3 = \sum m(1,2,3,4,9,10,11,12) + d(13,14,15)$



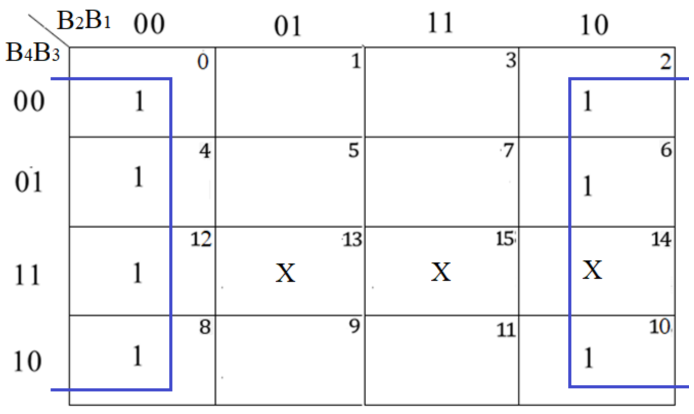
$$E_3 = \overline{B}_3B_1 + \overline{B}_3B_2 + B_3\overline{B}_2\overline{B}_1$$

3. K-map for E_2: $E_2 = \sum m(0,3,4,7,8,11,12) + d(13,14,15)$



$$E_2 = \overline{B}_2\overline{B}_1 + B_2B_1$$

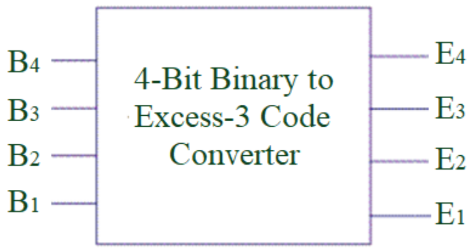
4. K-map for E_1: $E_1 = \sum m(0,2,4,6,8,10,12) + d(13,14,15)$



$$E_1 = \overline{B}_1$$

Thus, minimal expressions for Excess-3 outputs E_4, E_3, E_2, E_1 in terms of 4-bit binary inputs B_4, B_3, B_2, B_1 are as follows:

$$E_4 = B_4 + B_3B_1 + B_3B_2$$
$$E_3 = \overline{B}_3B_1 + \overline{B}_3B_2 + B_3\overline{B}_2\overline{B}_1$$
$$E_2 = \overline{B}_2\overline{B}_1 + B_2B_1$$
$$E_1 = \overline{B}_1$$



Block diagram of 4-bit binary to XS-3 code converter

3. Gray Code

The Gray code is a non-weighted code. It is also called as Reflective Code. The successive code words in this code differ in one bit position only and hence, it is an Unit Distance Code. The binary number can be converted into gray code by following steps:

1. Record the MSB of binary as the MSB of gray code
2. Add this MSB to the next bit in the binary (ignore the carry, if any). This sum bit is the second bit of gray code
3. Add 2nd bit of binary to the 3rd bit of binary (ignore the carry, if any). This sum bit is the 3rd bit of gray code
4. Add 3rd bit of binary to the 4th bit of binary (ignore the carry, if any). This sum bit is the 4th bit of gray code and so on.

Example: Convert 1001 to gray code.

$$\begin{tikzcd}
\Rightarrow \text{Binary} & 1 \arrow[r] \arrow[d] & 0 \arrow[r] \arrow[d] & 0 \arrow[r] \arrow[d] & 1 \arrow[d] \\
\text{Gray} & 1 & 1 & 0 & 1
\end{tikzcd}$$

4-bit Binary to Gray Code Converter

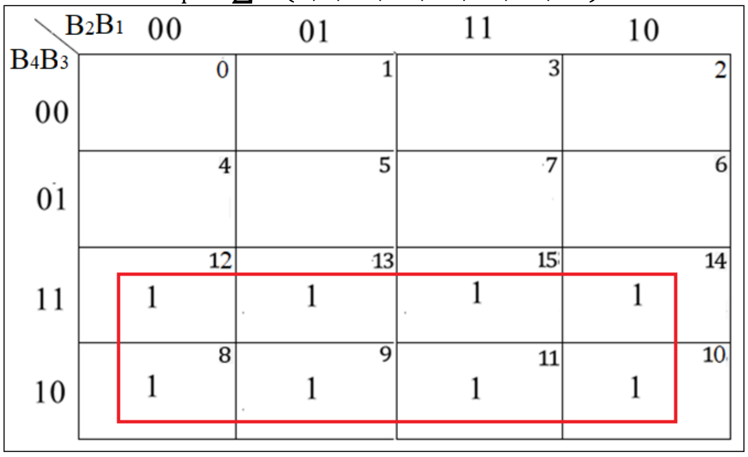
Decimal Number	4-bit Binary Number				Gray Output			
	B4	B3	B2	B1	G4	G3	G2	G1
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

- The input to 4-bit binary to gray code converter is 4-bit binary and output is 4-bit gray code. Thus, 16 input combinations are possible and all of them are valid.
- From conversion table, we observe expressions for Gray code outputs are as follows:

$$G_4 = \sum m(8,9,10,11,12,13,14,15)$$
$$G_3 = \sum m(4,5,6,7,8,9,10,11)$$
$$G_2 = \sum m(2,3,4,5,10,11,12,13)$$
$$G_1 = \sum m(1,2,5,6,9,10,13,14)$$

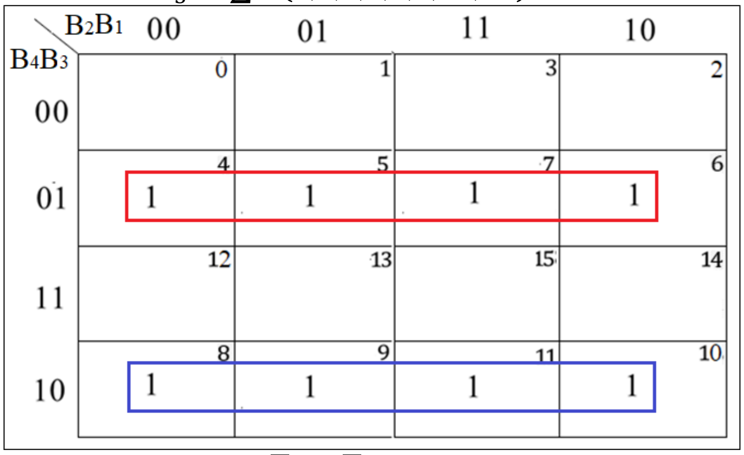
  - By drawing K-maps for the outputs and minimizing them, the minimal expressions gray code outputs G_4, G_3, G_2, G_1 in terms of 4-bit binary inputs B_4, B_3, B_2, B_1 can be obtained.

1. K-map for G_4: $G_4 = \sum m(8,9,10,11,12,13,14,15)$



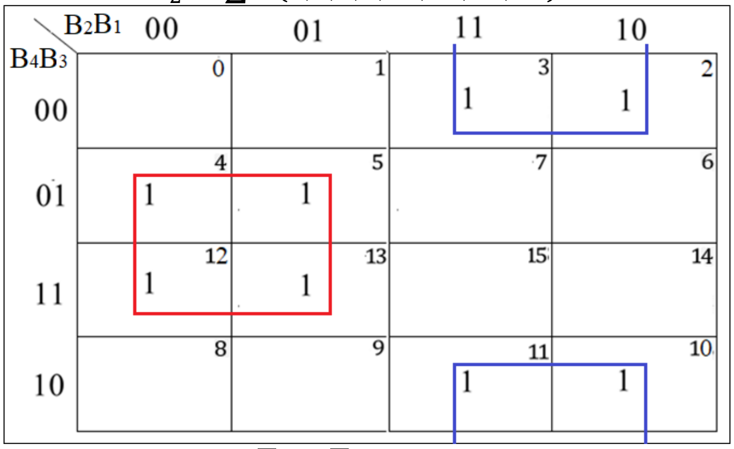
$$G_4 = B_4$$

2. K-map for G_3: $G_3 = \sum m(4,5,6,7,8,9,10,11)$



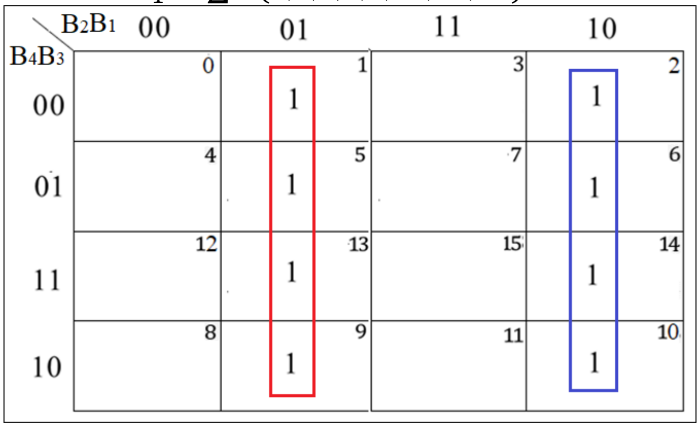
$$G_3 = B_4\overline{B}_3 + \overline{B}_4B_3 = B_4 \oplus B_3$$

3. K-map for G_2: $G_2 = \sum m(2,3,4,5,10,11,12,13)$



$$G_2 = B_3\overline{B}_2 + \overline{B}_3B_2 += B_3 \oplus B_2$$

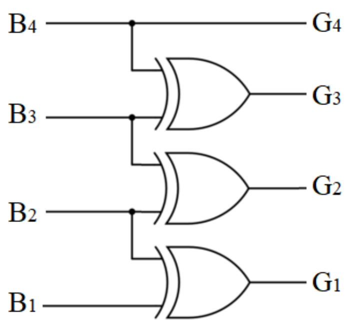
4. K-map for G_1: $G_1 = \sum m(1,2,5,6,9,10,13,14)$



$$G_1 = B_2\overline{B}_1 + \overline{B}_2B_1 = B_2 \oplus B_1$$

Thus, minimal expressions for Excess-3 outputs G_4, G_3, G_2, G_1 in terms of 4-bit binary inputs B_4, B_3, B_2, B_1 are as follows:

$$G_4 = B_4$$
$$G_3 = B_4 \oplus B_3$$
$$G_2 = B_3 \oplus B_2$$
$$G_1 = B_2 \oplus B_1$$



Logic diagram

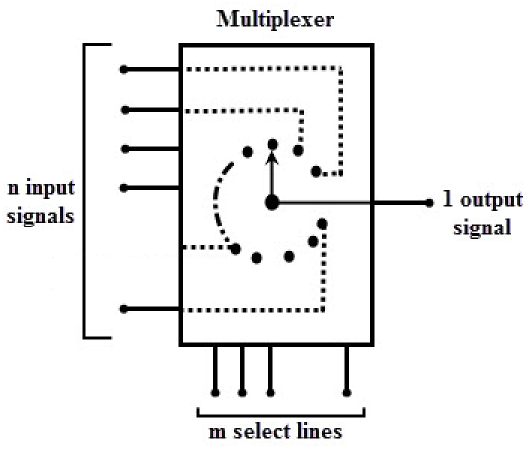
Multiplexers (Data Selectors)

A multiplexer (MUX) or data selector is a logic circuit that accepts several data inputs and allows only one of them at a time to get through to the output. The desired data inputs can be selected by SELECT inputs. Figure shows functional diagram of a general multiplexer:

Let, select lines of MUX = m then,
Number of input lines (n) is given by:

$$n = 2^m$$

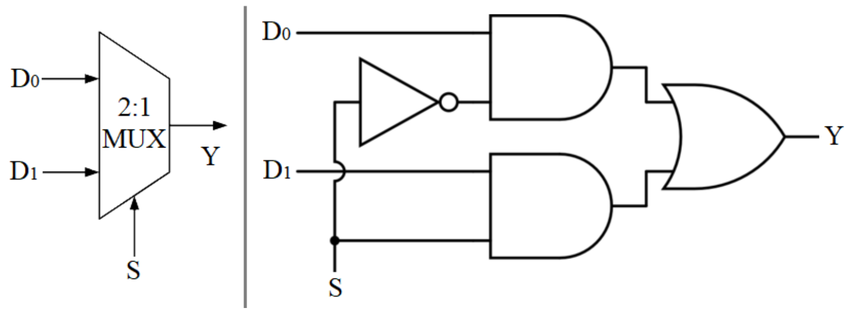
Multiplexer acts like a digitally controlled multi-position switch. The digital code is applied to SELECT inputs determines which input will be switched to output.



2:1 Multiplexer

Figure shows logic circuitry and function table for a 2-input MUX with data inputs D_0, D_1 and data select input S. Logic level applied to S determines which AND gate is enabled, so that data input passes through OR gate to the output. The output Y is given by following equation:

$$Y = D_0\overline{S} + D_1S$$

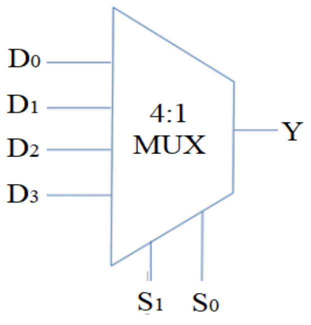


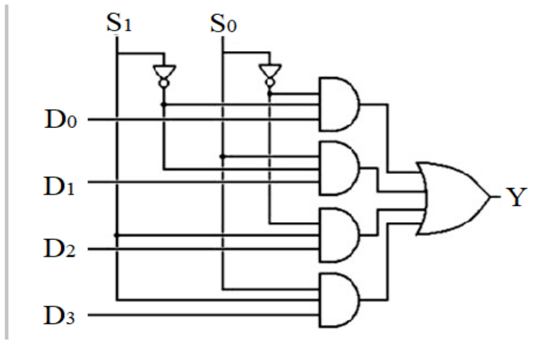
S	Y
0	D_0
1	D_1

4:1 Multiplexer

Figure shows logic circuitry and function table for a 4-input MUX with data inputs D_0, D_1, D_2, D_3 and data select input S_0, S_1. Logic levels applied to S_0, S_1 determine which AND gate is enabled, so that data input passes through OR gate to output. Output Y is given by following equation:

$$Y = D_0\bar{S}_1\bar{S}_0 + D_1\bar{S}_1S_0 + D_2S_1\bar{S}_0 + D_3S_1S_0$$





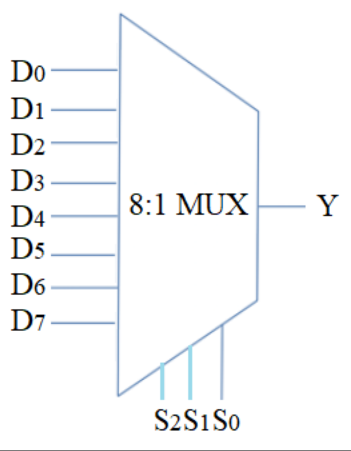
S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

8:1 Multiplexer

Figure shows logic circuitry and function table for a 4-input MUX with data inputs D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7 and data select input S_0, S_1, S_2. Logic levels applied to S_0, S_1 and S_2 determine which AND gate is enabled, so that data input passes through OR gate to output. Output Y is given by :

$$Y = D_0\bar{S}_2\bar{S}_1\bar{S}_0 + D_1\bar{S}_2\bar{S}_1S_0 + D_2\bar{S}_2S_1\bar{S}_0 + D_3\bar{S}_2S_1S_0 + D_4S_2\bar{S}_1\bar{S}_0 +$$

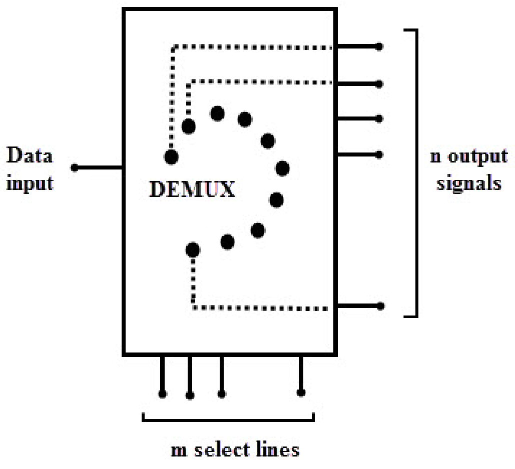
$$D_5S_2\bar{S}_1S_0 + D_6S_2S_1\bar{S}_0 + D_7S_2S_1S_0$$



S_2	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

Demultiplexers (Data Distributors)

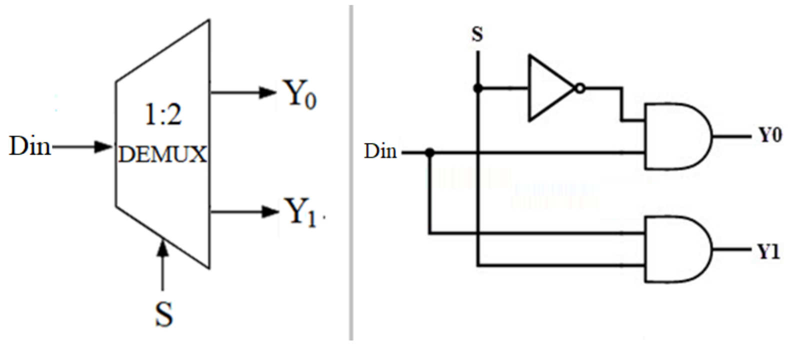
A demultiplexer (DEMUX) or Data Distributor is a logic circuit that accepts single input and distributes it to several outputs. It transmits same data to different destinations. Select input determines output line to which input data will be transmitted. Figure shows functional diagram of a general multiplexer:



Let, select lines of MUX = m then,
Number of output lines (n) is given by: $n = 2^m$
Demultiplexer takes one input data source and selectively distribute it to one of 'n' output channels just like a multi-position switch.

1:2 Demultiplexer

Figure shows logic circuitry and function table for a 2-output DEMUX with data outputs Y_0, Y_1 and data select input S. Logic level applied to 'S' determines which AND gate is enabled, so that data input passes through the gate to the output.

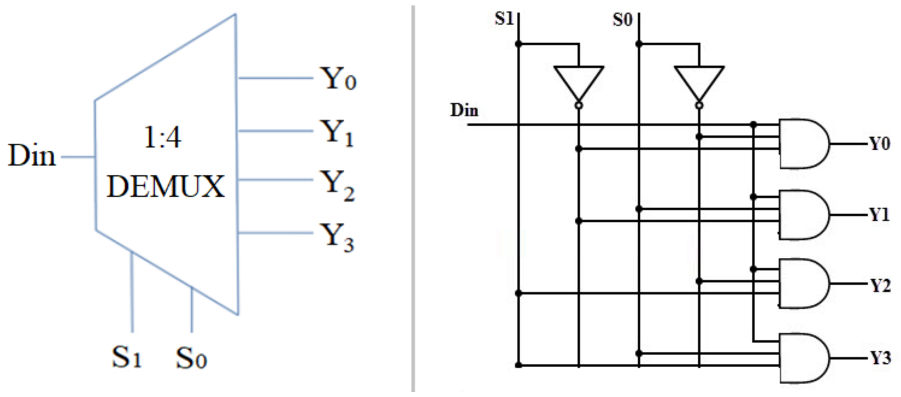


S	Y_1	Y_0
0	0	D_{in}
1	D_{in}	0

1:4 Demultiplexer

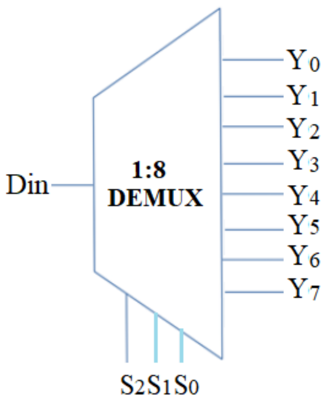
Figure shows logic circuitry and function table for a 4-output DEMUX with data outputs Y_0, Y_1, Y_2, Y_3 & select inputs S_0, S_1. Logic levels applied to S_0, S_1 determine which AND gate is enabled, so that data input passes through gate to output.

S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	D_{in}
0	1	0	0	D_{in}	0
1	0	0	D_{in}	0	0
1	1	D_{in}	0	0	0



1:8 Demultiplexer

Figure shows logic circuitry and function table for a 8-output DEMUX with data inputs Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, Y_7 and data select input S_0, S_1, S_2. Logic levels applied to S_0, S_1 and S_2 determine which AND gate is enabled, so that data input passes to output.



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	D_{in}
0	0	1	0	0	0	0	0	0	D_{in}	0
0	1	0	0	0	0	0	0	D_{in}	0	0
0	1	1	0	0	0	0	D_{in}	0	0	0
1	0	0	0	0	0	D_{in}	0	0	0	0
1	0	1	0	0	D_{in}	0	0	0	0	0
1	1	0	0	D_{in}	0	0	0	0	0	0
1	1	1	D_{in}	0	0	0	0	0	0	0

DECODERS

Decoder is a combinational circuit that convert n-bit binary input code into 2^n output lines such that only one output line is activated for each one of the possible combination of inputs. It has 'n' input lines and maximum of 2^n output lines. One of these outputs will be active high based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but minterms of n input variable lines, when it is enabled.

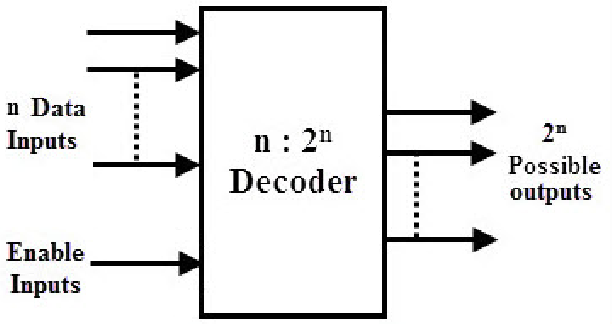
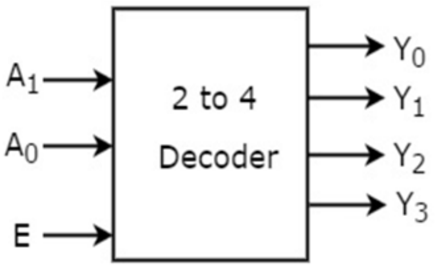


Figure: General decoder block diagram with n inputs and 2^n outputs

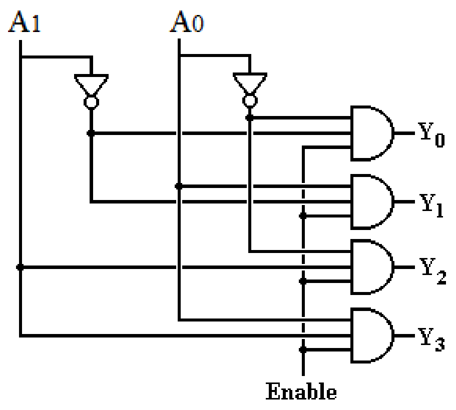
2 Line to 4 Line Decoder

2 Line to 4 line decoder has 2 inputs A_1, A_0 and 4 outputs Y_3, Y_2, Y_1, Y_0. E is an Enable pin and decoder will work only when it is enabled.



The block diagram of 2 Line to 4 Line decoder is shown in the figure:

Each output is having one product term. We can implement these four product terms by using four AND gates having three inputs each & two inverters as shown in the figure.
From the diagram we can write:

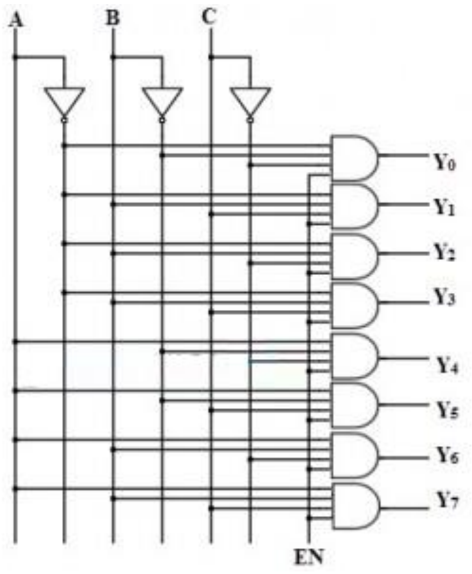


$$Y_0 = E\overline{A}_1\overline{A}_0$$
$$Y_1 = E\overline{A}_1A_0$$
$$Y_2 = EA_1\overline{A}_0$$
$$Y_3 = EA_1A_0$$

Truth Table of 2 Line to 4 Line decoder is shown below:

INPUT			OUTPUT			
EN	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

<u>3 Line to 8 Line Decoder</u>



3 Line to 8 line decoder has 3 inputs A, B, C and 8 outputs Y_7, Y_6, Y_5, Y_4, Y_3, Y_2, Y_1, Y_0. EN is an Enable pin and decoder will work only when it is enabled. Each output is having one product term. We can implement these eight product terms by using eight AND gates having four inputs each & three inverters as shown in the figure.

$$Y_0 = E\overline{A}\overline{B}\overline{C}; \quad Y_1 = E\overline{A}\overline{B}C$$
$$Y_2 = E\overline{A}B\overline{C}; \quad Y_3 = E\overline{A}BC$$
$$Y_4 = EA\overline{B}\overline{C}; \quad Y_5 = EA\overline{B}C$$
$$Y_6 = EAB\overline{C}; \quad Y_7 = EABC$$

Truth Table of 3 Line to 8 Line decoder is shown below:

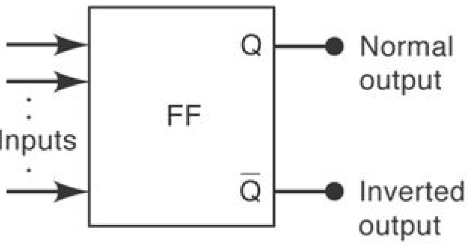
INPUT				OUTPUT							
EN	A	B	C	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0

1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

FLIP-FLOPS

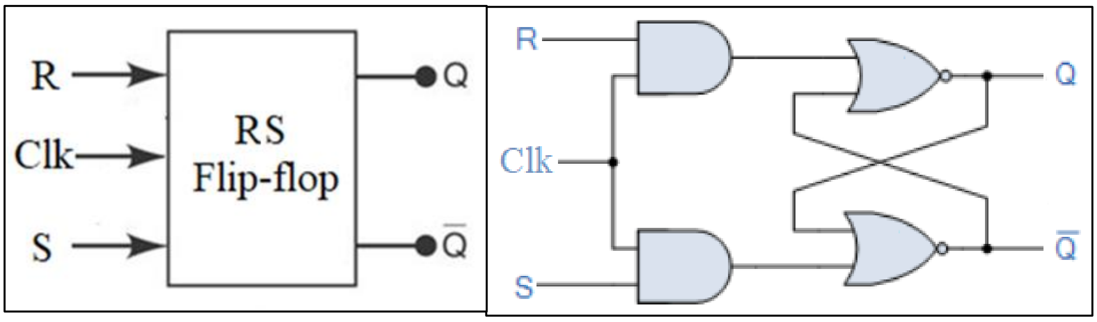
Flip-flops are the basic building block of most sequential circuits. A Flip-flop is the memory element which is made up of an assembly of logic gates. It stores one-bit and hence, is also called as one-bit memory. Even though a logic gate by itself has no storage capability, several logic gates can be connected together in ways that permit information to be stored. A flip-flop (FF) has two stable states. It can remain in either of the states indefinitely. States can be changed by applying the proper signals.

Figure shows the general symbol used for flip-flop. It can have one or more inputs & two outputs labelled as Q and $\overline{Q}$. Q is normal output of flip-flop and $\overline{Q}$ is inverted output.



RESET state ➔ when $Q = 0,\ \overline{Q} = 1$
SET state ➔ when $Q = 1,\ \overline{Q} = 0$

RS Flip-Flop: The RS flip-flop is the simplest flip-flop. It has two outputs and three inputs. The three inputs are SET, RESET and CLOCK. The flip-flop basically uses two NOR gates and two AND gates. The circuit gives output only when the CLOCK pin is high. When Clk input is at logic level "0", the outputs of the two AND gates are also at logic level "0", and outputs Q and $\overline{Q}$ will remain into their last known state. Figure shows symbol and circuit diagram for RS Flip-flop.



Working:

1) When R = 0, S = 0 and Clk = 1 then, the output of both AND gates is 0. These two are inputs to the NOR gate and hence, output of NOR gate will depend on the values of Q and $\overline{Q}$. If we assume Q = 0 and $\overline{Q}$ = 1 then, output of upper NOR will be 0 and output of lower NOR gate will remain at 1. Similarly, if Q is assumed to be 1 and $\overline{Q}$ = 0 then, there will be no change in their values. Hence, this state is called as NO CHANGE

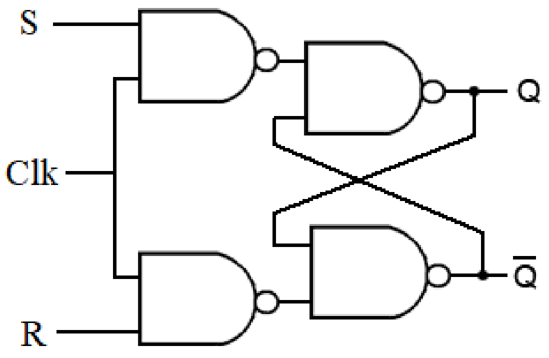
2) When R = 0, S = 1 and Clk = 1 then, the output of upper AND gate will be 0 and output of lower AND gate becomes 1. Outputs of these AND gates are inputs to the NOR gates. Since, one of input to lower NOR gate is 1, we can write its output as $\overline{Q}$ = 0 [by NOR gate principle]. This $\overline{Q}$ is nothing but second input to upper NOR gate. Thus, output of upper NOR gate will be Q = 1. Thus, output Q is SET to 1

3) When R = 1, S = 0 and Clk = 1 then, the output of upper AND gate will be 1 and output of lower AND gate becomes 0. Outputs of these AND gates are inputs to the NOR gates. Since, one of input to upper NOR gate is 1, we can write its output as Q = 0 [by NOR gate principle]. This Q is nothing but second input to lower NOR gate. Thus, output of lower NOR gate will be $\overline{Q}$ = 1. Thus, o/p Q is RESET to 0

TRUTH TABLE of RS FF

CLK	R	S	Q	$\overline{Q}$
0	X	X	No Change	
1	0	0	No Change	
1	0	1	1	0
1	1	0	0	1
1	1	1	Invalid	

4) When R = 1, S = 1 and Clk = 1 then, the output of both AND gates will be 1. Outputs of these AND gates are inputs to the NOR gates. Since, one of input of both NOR gates is 1, their outputs will be 0. That means, Q = 0 and $\overline{Q}$ = 0. But, it can not be true as Q and $\overline{Q}$ are complements of each other. Hence, this state is called as Invalid/Undetermined State.

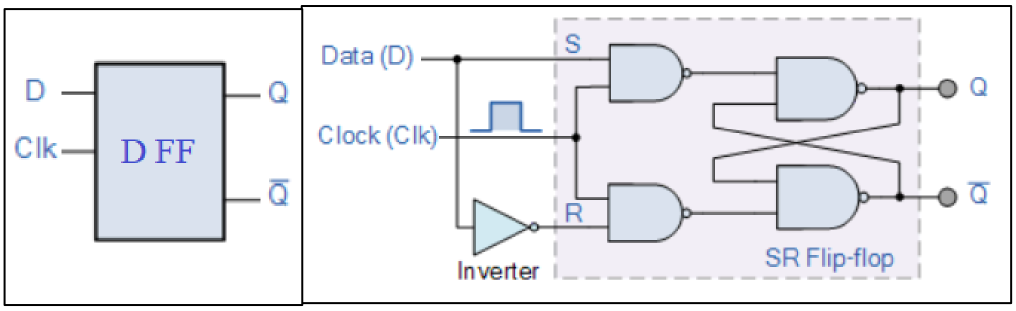


SR Flip-flop using NAND Gates

TRUTH TABLE of SR FF

CLK	S	R	Q	$\overline{Q}$
0	X	X	No Change	
1	0	0	No Change	
1	0	1	0	1
1	1	0	1	0
1	1	1	Invalid	

D Flip-Flop: The D flip-flop is constructed from a gated SR flip-flop with an inverter added between S and R inputs to allow for a single D (Data) input. This single data input, labelled D and is used in place of the SET signal and the inverter is used to generate the complementary RESET input thereby making a D flip-flop from a SR flip-flop (S = D and R = $\overline{D}$) as shown. The circuit gives output only when the CLOCK pin is high. Figure shows symbol and circuit diagram for D Flip-flop.



Working:

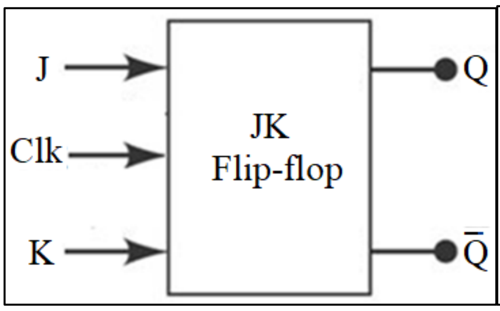
1) D = 0 ➔ S = 0 and R = 1. Thus, output Q will RESET to 0 [i.e, Q = 0, $\overline{Q}$ = 1] Hence, this state is called as RESET
2) D = 1 ➔ S = 1 and R = 0. Thus, output Q will SET to 1 [i.e, Q = 1, $\overline{Q}$ = 0] Hence, this state is called as SET

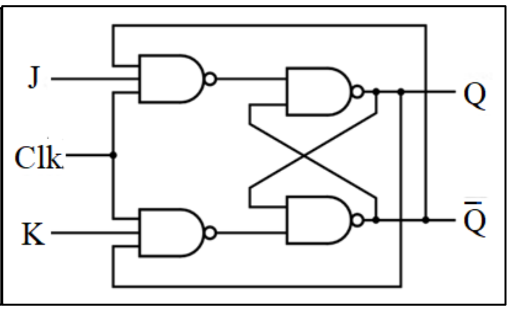
Thus, output of D flip-flop will be whatever logic level is applied to its data terminal (D) as long as the clock input is HIGH.

TRUTH TABLE of D Flip-Flop			
CLK	D	Q	$\overline{Q}$
0	X	No Change	
1	0	0	1
1	1	1	0

JK Flip-Flop

The sequential operation of the JK flip-flop is exactly the same as for the previous SR flip-flop with the same SET and RESET inputs. The difference this time is that the JK flip-flop has no invalid state of the SR flip-flop even when S and R are both at logic 1. Both inputs of the previous SR FF have now been replaced by two inputs called the J and K, respectively after its inventor *Jack Kilby*. [J = S and K = R]. The circuit gives output only when the CLOCK pin is high. Fig. shows symbol and circuit diagram for JK flip-flop





Working:

1) When J = 0, K = 0 and Clk = 1 (assume Q = 0 and $\overline{Q}$ = 1) then, the output of both three input NAND gates is 1. These two are inputs to two input NAND gate and hence, output of NAND gate will depend on the values of Q and $\overline{Q}$. Thus, output of upper NAND will be 0 and output of lower NAND gate will remain at 1. Similarly, if Q is assumed to be 1 and $\overline{Q}$ = 0 then, there will be no change in their values. Hence, it is called as No Change

2) When J = 1, K = 0 and Clk = 1 (assume Q = 0 and $\overline{Q}$ = 1) then, the output of upper three input NAND gates is 0 and output of lower three input NAND gate is 1. These two are inputs to two input NAND gate. Thus, output of upper two i/p NAND will be Q = 1 and output of lower two i/p NAND gate will be $\overline{Q}$ = 0. Similarly, if Q is assumed to be 1 and $\overline{Q}$ = 0 then also, output Q will SET to 1

3) When J = 0, K = 1 and Clk = 1 (assume Q = 0 and $\overline{Q}$ = 1) then, the output of both three input NAND gates is 1. These two are inputs to two input NAND gate. Thus, output of upper two input NAND will be Q = 0 and output of lower two input NAND gate will be $\overline{Q}$ = 1. Similarly, if Q is assumed to be 1 & $\overline{Q}$ = 0 then also, output Q will RESET to 0

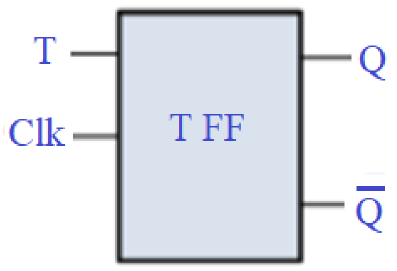
4) When J = 1, K = 1 and Clk = 1 (assume Q = 0 and $\overline{Q}$ = 1) then, output of upper two i/p NAND will be Q = 1 and output of lower two i/p NAND gate will be $\overline{Q}$ = 0. Similarly, if Q is assumed to be 1 and $\overline{Q}$ = 0 then we get Q = 0 & $\overline{Q}$ = 1.

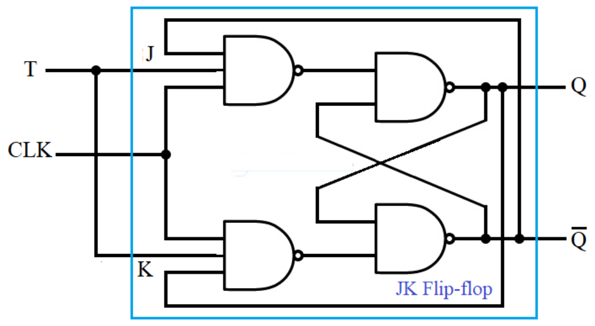
That means for J = K = Clk = 1, output will toggle between two states (i.e., it goes to opposite state of previous one)

TRUTH TABLE of JK FF				
CLK	J	K	Q	$\overline{Q}$
0	X	X	No Change	
1	0	0	No Change	
1	1	0	1	0
1	0	1	0	1
1	1	1	Toggle	

<u>T Flip-Flop:</u>

T flip-flop is also known as Toggle Flip-flop. To avoid the occurrence of intermediate state in SR flip-flop, only one input is provided to the flip-flop called Trigger input or Toggle input (T). Toggling means 'Changing the next state output to complement of the present state output'. T flip-flop can be designed by connecting J and K inputs together and giving them with single input called T. The circuit gives output only when the CLOCK pin is high. Figure shows symbol and circuit diagram for T Flip-flop





<u>WORKING:</u>

1) T = 0 ➔ J = 0 and K = 0. Thus, output Q will remain in its previous state (i.e., flip-flop will be in No Change State)
2) T = 1 ➔ J = 1 and K = 1. Thus, flip-flop will be in toggle mode (i.e., output goes to opposite state of previous one)

Thus, T flip-flop will toggle the output when logic level applied to its toggle terminal (T) is HIGH.

Truth Table of T Flip-flop			
CLK	T	Q	$\overline{Q}$
0	X	No Change	
1	0	No Change	
1	1	TOGGLE	

Important Questions:

1. Distinguish between Combinational & Sequential circuits.
2. Design and draw 4-bit binary to gray code converter.
3. Explain the working of 4: 1 Multiplexer along with its truth table.
4. Describe the operation of 1:4 De-multiplexer along with its truth table.
5. Explain the working of SR flip-flop with suitable circuit diagram.
6. With the help of truth table describe the operation of JK flip-flop.