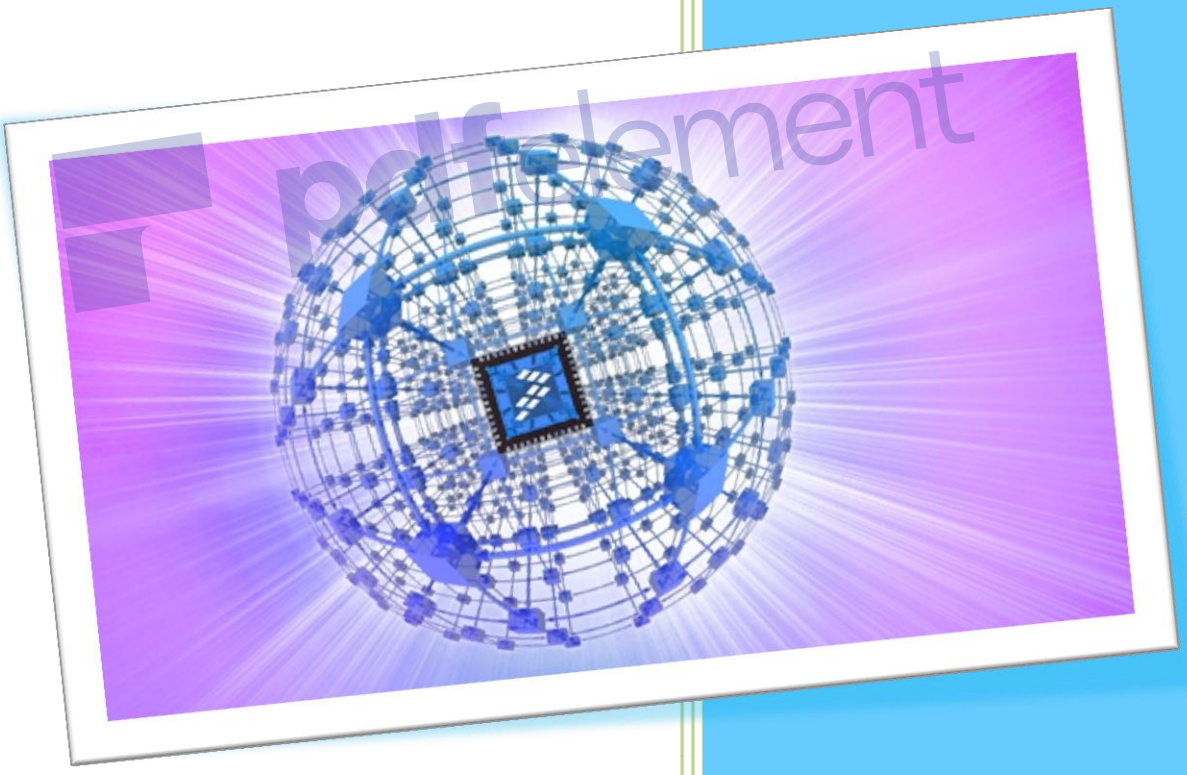
Republic of Iraq
Ministry of Higher Education & Scientific Research
Northern Technical University
Technical College of Kirkuk
Electronic & Control Dept.

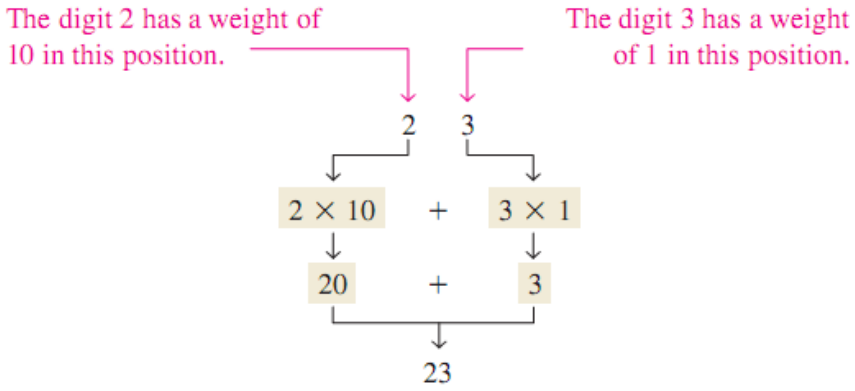
Digital Electronics



1-Number Systems

1.1 Decimal Numbers

You are familiar with the decimal number system because you use decimal numbers every day. Although decimal numbers are commonplace, their weighted structure is often not understood. The decimal number system has ten digits. Each of the ten digits, 0 through 9, represents a certain quantity



1.2 Binary Number

The binary number system is another way to represent quantities. It is less complicated than the decimal system because it has only two digits. The decimal system with its ten digits is a base-ten system; the binary system with its two digits is a base-two system. The two binary digits (bits) are 1 and 0. The weights in a binary number are based on powers of two.

Table 1

Decimal Number	Binary Number			
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

As you have seen in Table1, four bits are required to count from zero to 15. In general, with n bits you can count up to a number equal to $2^n - 1$. The weight or value of a bit increases from right to left in a binary number.

1.2.1 Binary-to-Decimal Conversion

The decimal value of any binary number can be found by adding the weights of all bits that are 1 and discarding the weights of all bits that are 0.

Example:
Convert the binary number 1101101 to decimal.
Solution

$$weight: 2^6 \quad 2^5 \quad 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0$$

$$binary \quad number: 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1$$

$$1101101 = \quad 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$$

$$= 64 + 32 + 8 + 4 + 1 = 109$$

Example:
Convert the fractional binary number 0.1011 to decimal.
Solution

$$weight: 2^{-1} \quad 2^{-2} \quad 2^{-3} \quad 2^{-4}$$

$$binary \quad number: 0. \quad 1 \quad 0 \quad 1 \quad 1$$

$$0.1011 = \quad 2^{-1} \quad 2^{-3} \quad 2^{-4}$$

$$= 0.5 + 0.125 + 0.0625 = 0.6875$$

1.2.2 Decimal-to-Binary Conversion

1- *Sum-of-Weights Method*
To get the binary number for a given decimal number, find the binary weights that adds up to the decimal number.

Example:
Convert the following decimal numbers to binary:
(a) 12 (b) 25 (c) 58 (d) 82
Solution

$a)$ $12 = 8 + 4 = 2^3 + 2^2 \rightarrow 1100$

$b)$ $25 = 16 + 8 + 1 = 2^4 + 2^3 + 2^0 \rightarrow 11001$

$c)$ $58 = 32 + 16 + 8 + 2 = 2^5 + 2^4 + 2^3 + 2^1 \rightarrow 111010$

$d)$ $82 = 64 + 16 + 2 = 2^6 + 2^4 + 2^1 \rightarrow 1010010$

2- *Repeated Division-by-2 Method*

To get the binary number for a given decimal number, divide the decimal number by 2 until the quotient is 0. Remainders form the binary number.

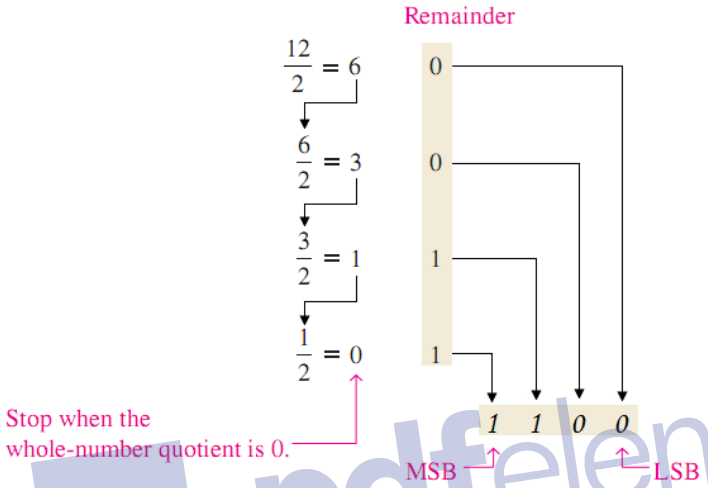
Example:

Convert the following decimal numbers to binary:

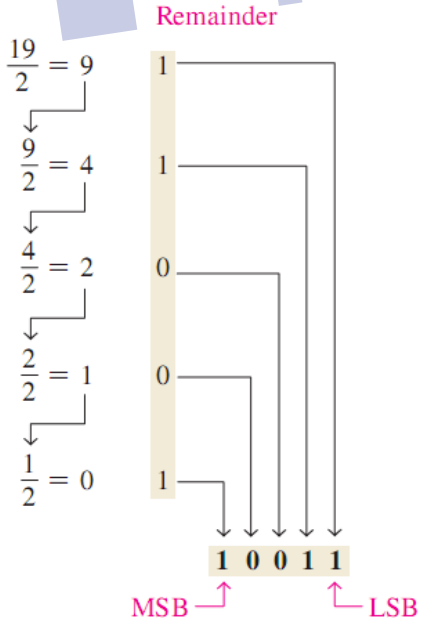
a) 12 b) 19 c) 45

Solution

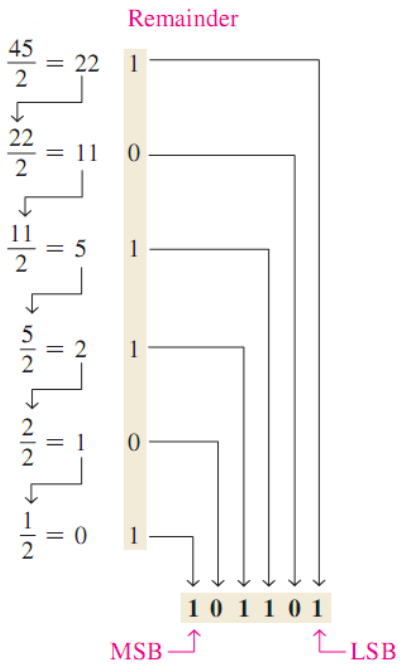
a)



b)



c)



MSB: Most Significant Bit.

LSB: Least Significant Bit

1.2.3 Converting Decimal Fractions to Binary

1- *Sum-of-Weights*

The sum-of-weights method can be applied to fractional decimal numbers, as shown in the following example:

$0.625 = 0.5 + 0.125 = 2^{-1} + 2^{-3} = 0.101$

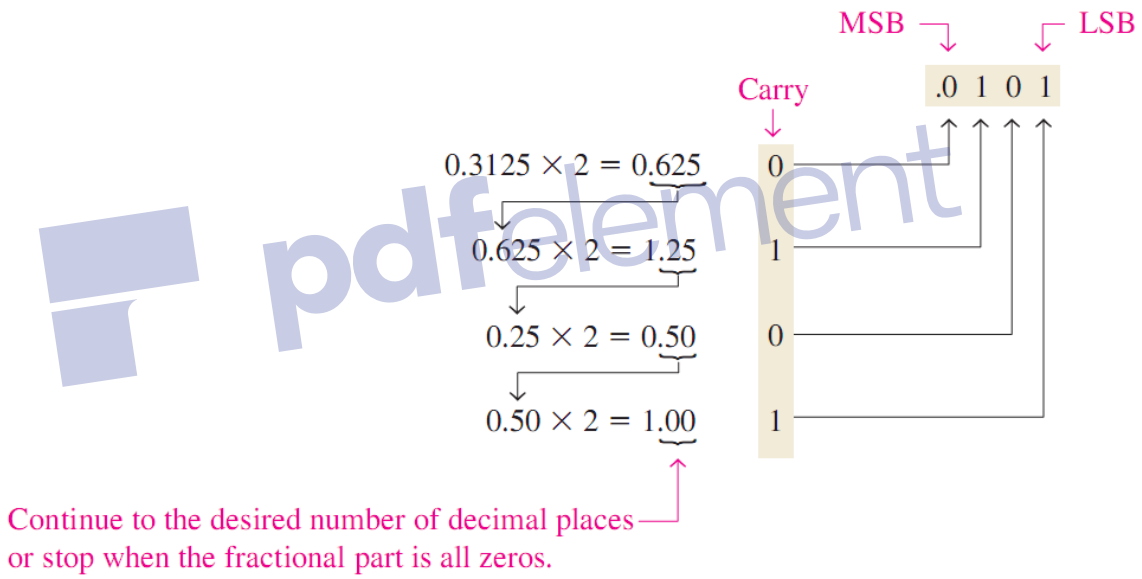
There is a 1 in the 2^{-1} position, a 0 in the 2^{-2} position, and a 1 in the 2^{-3} position.

2- *Repeated Multiplication by 2*

Decimal fractions can be converted to binary by repeated multiplication by 2.

Example

Convert the decimal fraction 0.3125 to binary.



- H.W

1. Convert each decimal number to binary by using the sum-of-weights method:
   (a) 23 (b) 57 (c) 45.5
2. Convert each decimal number to binary by using the repeated division-by-2 method (repeated multiplication-by-2 for fractions):
   (a) 14 (b) 21 (c) 0.375

1.2.4 Binary Arithmetic

Binary arithmetic is essential in all digital computers and in many other types of digital systems. To understand digital systems, you must know the basics of binary addition, subtraction, multiplication, and division.

1- *Binary Addition*

The four basic rules for adding binary digits (bits) are as follows:

$0 + 0 = 0$ Sum of 0 with carry of 0

$0 + 1 = 1$ Sum of 1 with carry of 0

$1 + 0 = 1$ Sum of 1 with carry of 0

$1 + 1 = 10$ Sum of 0 with carry of 1

When there is a carry of 1, you have a situation in which three bits are being added (a bit in each of the two numbers and a carry bit). This situation is illustrated as follows:

carry
bits
↓

$1 + 0 + 0 = 01$ Sum of 1 with carry of 0

$1 + 1 + 0 = 10$ Sum of 0 with carry of 1

$1 + 0 + 1 = 10$ Sum of 1 with carry of 1

$1 + 1 + 1 = 11$ Sum of 1 with carry of 1

Example

Add the following binary numbers:

(a) 11 + 11 (b) 100 + 10

(c) 111 + 11 (d) 110 + 100

Solution

The equivalent decimal addition is also shown for reference.

$$a)\ \begin{array}{r} 11 \\ +11 \\ \hline 110 \end{array} \quad \begin{array}{r} 3 \\ +3 \\ \hline 6 \end{array} \qquad b)\ \begin{array}{r} 100 \\ +10 \\ \hline 110 \end{array} \quad \begin{array}{r} 4 \\ +2 \\ \hline 6 \end{array} \qquad c)\ \begin{array}{r} 111 \\ +11 \\ \hline 1010 \end{array} \quad \begin{array}{r} 7 \\ +3 \\ \hline 10 \end{array} \qquad d)\ \begin{array}{r} 110 \\ +100 \\ \hline 1010 \end{array} \quad \begin{array}{r} 6 \\ +4 \\ \hline 10 \end{array}$$

2- *Binary Subtraction*

The four basic rules for subtracting bits are as follows:

$0 - 0 = 0$

$1 - 1 = 0$

$1 - 0 = 1$

$10 - 1 = 1 \quad 0 - 1 \quad$ with a borrow of 1

Example

Perform the following binary subtractions:

(a) 11 - 01 (b) 11 – 10

Solution

$$a)\quad \begin{array}{r} 11 \\ -01 \\ \hline 10 \end{array} \quad \begin{array}{r} 3 \\ -1 \\ \hline 2 \end{array} \qquad b)\quad \begin{array}{r} 11 \\ -10 \\ \hline 01 \end{array} \quad \begin{array}{r} 3 \\ -2 \\ \hline 1 \end{array}$$

Example

Subtract 011 from 101.

Solution

$$\begin{array}{r} 101 \\ -011 \\ \hline 010 \end{array} \quad \begin{array}{r} 5 \\ -3 \\ \hline 2 \end{array}$$

Left column:
When a 1 is borrowed, a 0 is left, so 0 − 0 = 0.

Middle column:
Borrow 1 from next column to the left, making a 10 in this column, then 10 − 1 = 1.

$$\begin{array}{r} {}^{0}\cancel{1}^{1}01 \\ -011 \\ \hline 010 \end{array}$$

Right column:
1 − 1 = 0

- H.W

1- Perform the following binary subtractions.
   a) 111 - 100.
   b) 110-001
2- Subtract 101 from 110.

3- *Binary Multiplication*

The four basic rules for multiplying bits are as follows:

$0 \times 0 = 0$

$0 \times 1 = 0$

$1 \times 0 = 0$

$1 \times 1 = 1$

Example

Perform the following binary multiplications:

(a) 11×11 (b) 101×111

Solution

a)

$$\begin{array}{r} 11 \\ \times 11 \\ \hline 11 \\ +110 \\ \hline 1001 \end{array} \qquad \begin{array}{r} 3 \\ \times 3 \\ \hline 9 \end{array}$$

b)

$$\begin{array}{r} 111 \\ \times 101 \\ \hline 111 \\ 0000 \\ +11100 \\ \hline 100011 \end{array} \qquad \begin{array}{r} 7 \\ \times 5 \\ \hline 35 \end{array}$$

4- *Binary Division*

Division in binary follows the same procedure as division in decimal.

Example

Perform the following binary divisions:

(a) $110 \div 11$ (b) $110 \div 10$ (c) $1001 \div 11$ (d) $10100 \div 100$

Solution

a)

$$\begin{array}{r} 10 \\ 11\overline{)110} \\ -110 \\ \hline 000 \end{array}$$

b)

$$\begin{array}{r} 11 \\ 10\overline{)110} \\ -10 \\ \hline 10 \\ -10 \\ \hline 00 \end{array} \qquad \begin{array}{r} 3 \\ 2\overline{)6} \\ -6 \\ \hline 0 \end{array}$$

c)

$$\begin{array}{r} 11 \\ 11\overline{)1001} \\ -11 \\ \hline 011 \\ -11` \\ \hline 00 \end{array} \qquad \begin{array}{r} 3 \\ 3\overline{)9} \\ -9 \\ \hline 0 \end{array}$$

d)

$$\begin{array}{r} 101 \\ 100\overline{)10100} \\ -100 \\ \hline 00100 \\ -100` \\ \hline 00 \end{array} \qquad \begin{array}{r} 5 \\ 4\overline{)20} \\ -20 \\ \hline 0 \end{array}$$

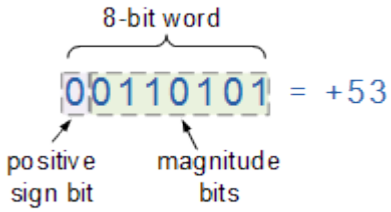
- نأخذ الاعداد من اليسار.
- في حالة ناتج الطرح ليس صفر العدد أصغر يضاف صفر الى الناتج ثم يتم إنزال العدد الذي يليه
- في حالة بقاء اصفار فقط وناتج الطرح اصفار فقط يتم وضعها مباشرة في الناتج

➲ H.W

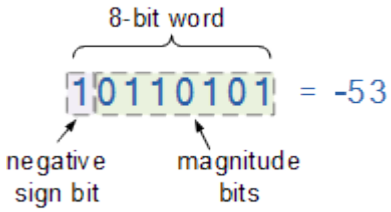
1. Perform the following binary additions:
   a) $1101+1010$ (b) $10111+01101$
2. Perform the following binary subtractions:
   a) $1101-0100$ (b) $1001-0111$
3. Perform the indicated binary operations:
   a) 110×111 (b) $1100\div011$ (c) 1101×1010 (d) $1111\div101$

1.2.5 Signed Binary Numbers

Positive Signed Binary Numbers



Negative Signed Binary Numbers



unsigned numbers can have a wide range of representation. But whereas, in case of signed numbers, we can represent their range only from

$$-\left(2^{n-1}-1\right) to \quad +\left(2^{n-1}-1\right)$$

Where n is the number of bits (including sign bit).

Example:
For a 5 bit signed binary number (including 4 magnitude bits & 1 sign bit), the range will be

$$-(2^{(5-1)}-1) \text{ to } +(2^{(5-1)}-1)$$
$$-(2^{(4)}-1) \text{ to } +(2^{(4)}-1)$$
$$-15 \text{ to } +15$$

Unsigned 8- bit binary numbers will have range from 0-255. The 8 – bit signed binary number will have maximum and minimum values as shown below.
The maximum positive number is 0111 1111 +127
The maximum negative number is 1000 0000 -127

There are three common ways to represent negative numbers within the computer. They are

1) Signed magnitude representation.
2) 1's compliment representation.
3) 2's complement representation.

1- Signed magnitude representation

The binary numbers which can be identified by their MSB (Most Significant Bit), whether they are positive or negative are called "Signed binary numbers".

Ex: $1001 \rightarrow +9(positive)$, $1001 \rightarrow -1(negative)$

This is the simplest way of representing the both positive and negative numbers in binary system. In the signed magnitude representation,

- Positive number is represented with '0' at its most significant bit (MSB).
- Negative number is represented with '1' at its most significant bit (MSB).

2- One's Complement of a Signed Binary Number

1's complement is another way of feeding the negative binary number to the computer. In one's complement method , the positive binary numbers are unchanged. But the negative numbers are represented by taking 1's complement of unsigned positive number.
A positive number always starts with 0, at its MSB while a negative number always starts with 1, at its MSB.

Finding the 1's Complement

The 1's complement of a binary number is found by changing all 1s to 0s and all 0s to 1s, as illustrated below:

1 0 1 1 0 0 1 0	Binary number
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓	
0 1 0 0 1 1 0 1	1's complement

Ex:If a binary number is 01101001=$(105)_{10}$,
then its one's complement is 10010110 =$(-105)_{10}$
Ex: -33 =?
33 is represented as $(100001)_2$
In 8 bit notation, it is represented as $(0010\ 0001)_2$
Now, -33 is represented in one's compliment as $(1101\ 1110)_2$
Ex : -127 =?
In 8 bit notation, 127 is represented as $(0111\ 1111)_2$
Now, -127 is represented in one's compliment as $(1000\ 0000)_2$
Ex : -1 =?
1 is represented as (001)2
In 8 bit notation, it is represented as (0000 0001)2
Now, -1 is represented in one's compliment as (1111 1110)2

Subtraction using 1's compliment

To subtract a number from another binary number, first it has to be converted into its one's compliment.

There are 3 possible cases for subtracting the negatives numbers by using 1's compliment.

Case 1 : Negative number smaller than positive number.

Ex: $(28)_{10}$ & $(-15)_{10}$

We know 28 is represented in binary number system as $(011100)_2$

15 is represented in binary number system as $(01111)_2$

1's compliment of 15 is $(10000)_2$ i.e. -15

Sign representation	+	0	1	1	1	0	0
		1	1	0	0	0	0
Carry	1	0	0	1	1	0	0
							1
		0	0	1	1	0	1

$(13)_{10}$ is same as 0 01101 in binary system.

Case 2: Negative number greater than positive number.

Ex: $(15)_{10}(-28)_{10}$

We know 28 is represented in binary number system as $(011100)_2$

15 is represented in binary number system as $(01111)_2$

1's compliment of 28 is (100011)2 i.e. -28

Sign representation	+	1	0	0	0	1	1
		0	0	1	1	1	1
		1	1	0	0	1	0

$(-13)_{10}$ is same as 1 10010 in binary system.

Case 3: both are negative.

Ex: $(-28)_{10}$ & $(-15)_{10}$

We know 28 is represented in binary number system as $(011100)_2$

1's compliment of 28 is (100011)2i.e. -28

15 is represented in binary number system as $(01111)_2$

1's compliment of 15 is $(10000)_2$ i.e. -15

Sign representation		1	1	0	0	0	1	1 +
		1	1	1	0	0	0	0
Carry	**1**	1	0	1	0	0	1	1
								1
		1	0	1	0	1	0	0

$(-43)_{10}$ is same as 1010100 in binary system.

3- Two's Complement of a Signed Binary Number

Finding the 2's Complement

The 2's complement of a binary number is found by adding 1 to the LSB of the 1's complement.

2's complement = (1's complement) + 1

Ex Find the 2's complement of 10110010.

10110010	Binary number
01001101	1's complement
+ 1	Add 1
01001110	2's complement

Ex: -33 =?

33 is represented as $(100001)_2$

In 8 bit notation, it is represented as $(0010\ 0001)_2$

Now, -33 is represented in one's compliment as $(1101\ 1110)_2$

Adding 1 (0000 0001) to it,

The result is $(1101\ 1111)_2$

Therefore, the two's complement of the number – 33 is $(1101\ 1111)_2$.

Ex: -127 =?

In 8 bit notation, 127 is represented as $(0111\ 1111)_2$

Now, -127 is represented in one's compliment as $(1000\ 0000)_2$

Adding 1 (0000 0001) to it,

The result is $(1000\ 0001)_2$

Therefore, the two's complement of the number -127 is $(1000\ 0001)_2$

Ex: -1 =?

1 is represented as $(001)_2$

In 8 bit notation, it is represented as $(0000\ 0001)_2$

Now, -1 is represented in one's compliment as $(1111\ 1110)_2$

Adding 1 (0000 0001) to it,

The result is $(0000\ 0010)_2$

Therefore, the two's complement of the number -1 is $(0000\ 0010)_2$

2's complement subtraction

For subtracting a smaller number from a larger number, the 2's complement method is as follows:

1. Determine the 2's complement of the smaller number.
2. Add the 2's complement to the larger number.
3. Discard the final carry (there is always one in this case).

Example
Use 2's complement to subtract the 11001-10011.
Solution

10011	smaller number
01100	1's complement
+ 1	Add1
01101	2's complement
+11001	larger number
100110	Discard the final carry
000110	result

For subtracting a larger number from a smaller number, the 2's complement method is as follows:

1. Determine the 2's complement of the larger number.
2. Add the 2's complement to the smaller number.
3. There is no carry from the left-most column. The result is in 2's complement form and is negative.
4. Change the sign and take the 2's complement of the result to get the final answer.

Example
Subtract 11100 from 10011 Using 2's complement.
Solution

11100	larger number
00011	1's complement
+ 1	Add1
00100	2's complement
+10011	smaller number
10111	first result
01000	1's complement
+ 1	Add1
01001	result with out sign
-01001	change sign to get the final answer

1.3 Octal Number

The octal number system is composed of eight digits, which are 0, 1,2,3,4,5,6,7. To count above 7, begin another column and start over 10, 11, 12……etc.
The octal number system has a base of 8

Decimal	Octal	Decimal	Octal
0	**0**	8	**10**
1	1	9	11
2	2	10	12
3	3	11	13
4	4	12	14
5	5	13	15
6	6	14	16
7	**7**	15	**17**

1.3.1 Octal-to-Decimal Conversion

The conversion of an octal number to its decimal equivalent is accomplished by multiplying each digit by its weight and summing the products, as illustrated here for $(2374)_8$.

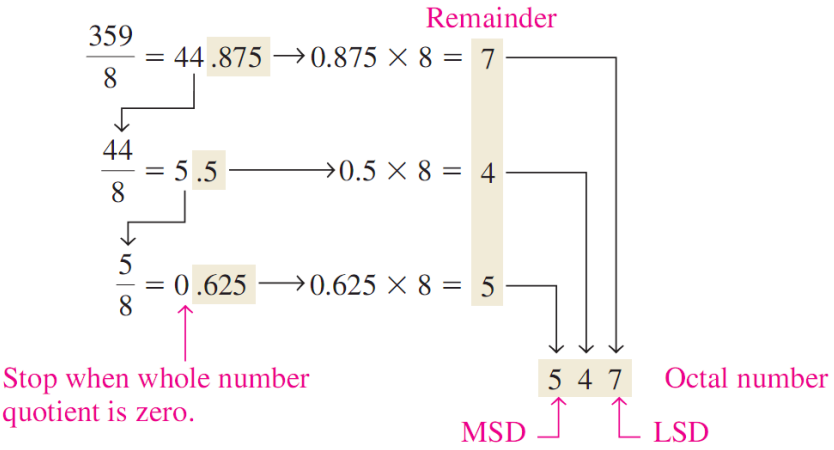
$$\begin{aligned}
&\textit{weight}: 8^3 \quad 8^2 \quad 8^1 \quad 8^0 \\
&\textit{binary number}: 2 \quad 3 \quad 7 \quad 4 \\
(2374)_8 &= (2\times 8^3)+(3\times 8^2)+(7\times 8^1)+(4\times 8^0) \\
&= (2\times 512)+(3\times 64)+(7\times 8)+(4\times 1) \\
&= 1024+192+56+4 \\
&= (1276)_{10}
\end{aligned}$$

- H.W / Determine the decimal value of $(0.325)_8$

1.3.2 Decimal-to-Octal Conversion

A method of converting a decimal number to an octal number is the repeated division-by-8 method.

Example
Convert the following decimal number to octal number 359.
Solution



1.3.3 Octal-to-Binary Conversion

Because each octal digit can be represented by a 3-bit binary number, it is very easy to convert from octal to binary. Each octal digit is represented by three bits as shown in Table 2.

Table 2

Octal/binary conversion.

Octal Digit	0	1	2	3	4	5	6	7
Binary	000	001	010	011	100	101	110	111

To convert an octal number to a binary number, simply replace each octal digit with the appropriate three bits.

Example
Convert each of the following octal numbers to binary:

a) 13_8 b) 25_8 c) 140_8 d) 7526_8

Solution

a)

$$\text{binary number} = \overbrace{001}^{1}\overbrace{011}^{3}$$

b)

$$\text{binary number} = \overbrace{010}^{2}\overbrace{101}^{5}$$

c)

$$\text{binary number} = \overbrace{001}^{1}\overbrace{100}^{4}\overbrace{000}^{0}$$

d)

$$\text{binary number} = \overbrace{111}^{7}\overbrace{101}^{5}\overbrace{010}^{2}\overbrace{110}^{6}$$

1.3.4 Binary-to-Octal Conversion

To convert a binary number to an octal number simply break the binary number to groups of three bits and convert each group in to the appropriate octal digit.

Example
Convert each of the following binary numbers to octal:
a) 110101 (b) 101111001 (c) 100110011010 (d) 11010000100
Solution

a) $\underbrace{110}\underbrace{101}$
$\downarrow \quad \downarrow$
6 5
octal number $= 65_8$

b) $\underbrace{101}\underbrace{111}\underbrace{001}$
$\downarrow \quad \downarrow \quad \downarrow$
5 7 1
octal number $= 571_8$

c) $\underbrace{100}\underbrace{110}\underbrace{011}\underbrace{010}$
$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$
4 6 3 2
octal number $= 4632_8$

d) $\underbrace{011}\underbrace{010}\underbrace{000}\underbrace{100}$
$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$
3 2 0 4
octal number $= 3204_8$

- H.W

1. Convert the following octal numbers to decimal:
   a) 73_8 (b) 125_8
2. Convert the following decimal numbers to octal:
   a) 98_{10} (b) 163_{10}
3. Convert the following octal numbers to binary:
   a) 46_8 (b) 723_8 (c) 5624_8
4. Convert the following binary numbers to octal:
   a) 110101111 (b) 1001100010 (c) 10111111001

1.4 Hexadecimal Numbers

The hexadecimal number system has sixteen characters; it is used primarily as a compact way of displaying or writing binary numbers because it is very easy to convert between binary and hexadecimal.
The hexadecimal number system consists of digits 0–9 and letters A–F.
Each hexadecimal digit represents a 4-bit binary number (as listed in Table 3).

Table 3

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

1.4.1 Binary-to-Hexadecimal Conversion

Converting a binary number to hexadecimal is a straightforward procedure. Simply breaks the binary numbers into 4-bit groups, starting at the right-most bit and replace each 4-bit group with the equivalent hexadecimal symbol.

Example
Convert the following binary numbers to hexadecimal:
a) 1100101001010111 (b) 111111000101101001
Solution

a) $\underbrace{1100}_{C}\underbrace{1010}_{A}\underbrace{0101}_{5}\underbrace{0111}_{7}$
hexadecimal number $= CA57_{16}$

b) $\underbrace{0011}_{3}\underbrace{1111}_{F}\underbrace{0001}_{1}\underbrace{0110}_{6}\underbrace{1001}_{9}$
hexadecimal number $= 3F169_{16}$

Two zeros have been added in part (b) to complete a 4-bit group at the left.

➲ H.W/ Convert the binary number 1001111011110011100 to hexadecimal.

1.4.2 Hexadecimal-to-Binary Conversion

To convert from a hexadecimal number to a binary number, reverse the process and replace each hexadecimal symbol with the appropriate four bits.

Example
Determine the binary numbers for the following hexadecimal numbers:
a) $10A4_{16}$ b) $CF8E_{16}$ c) 9742_{16}
Solution

$a)$

$$\text{binary number} = \overset{1}{\downarrow}\ \overset{0}{\downarrow}\ \overset{A}{\downarrow}\ \overset{4}{\downarrow}\ \ 1000010100100$$

$b)$

$$\text{binary number} = \overset{C}{\downarrow}\ \overset{F}{\downarrow}\ \overset{8}{\downarrow}\ \overset{E}{\downarrow}\ \ 1100111110001110$$

$c)$

$$\text{binary number} = \overset{9}{\downarrow}\ \overset{7}{\downarrow}\ \overset{4}{\downarrow}\ \overset{2}{\downarrow}\ \ 1001011101000010$$

In part (a), the MSB is understood to have three zeros preceding it, thus forming a 4-bit group.

- H.W/ Convert the hexadecimal number 6BD3 to binary.

1.4.3 Hexadecimal-to-Decimal Conversion

One way to find the decimal equivalent of a hexadecimal number is to first convert the hexadecimal number to binary and then convert from binary to decimal.

Example
Convert the following hexadecimal numbers to decimal:
(a) $1C_{16}$ (b) $A85_{16}$
Solution
Remember; convert the hexadecimal number to binary first, then to decimal.

$a)$

$$\text{binary number} = \overset{1}{\downarrow}\ \overset{C}{\downarrow}\ \ 11100 \quad = 2^4 + 2^3 + 2^2 = 16 + 8 + 4 = 28_{10}$$

$b)$

$$\text{binary number} = \overset{A}{\downarrow}\ \overset{8}{\downarrow}\ \overset{5}{\downarrow}\ \ 101010000101 \quad = 2^{11} + 2^9 + 2^7 + 2^2 + 2^0 = 2048 + 512 + 128 + 4 + 1 = 2693_{10}$$

Or using the sum of weights method

Example

Convert the following hexadecimal numbers to decimal:

(a) $E5_{16}$ (b) $B2F8_{16}$

Solution

Recall from Table 3 that letters A through F represent decimal numbers 10 through 15, respectively.

$$a)\quad E5_{16} = (E\times16^1)+(5\times16^0) = (14\times16)+(5\times1) = 224+5 = 229_{10}$$

$$\begin{aligned} b)\quad B2F8_{16} &= (B\times16^3)+(2\times16^2)+(F\times16^1)+(8\times16^0) \\ &= (11\times4096)+(2\times256)+(15\times16)+(8\times1) \\ &= 45056+215+240+8 \\ &= 45816_{10} \end{aligned}$$

- H.W: Convert the following hexadecimal numbers to decimal.
  a) $6BD_{16}$ b) $60A_{16}$

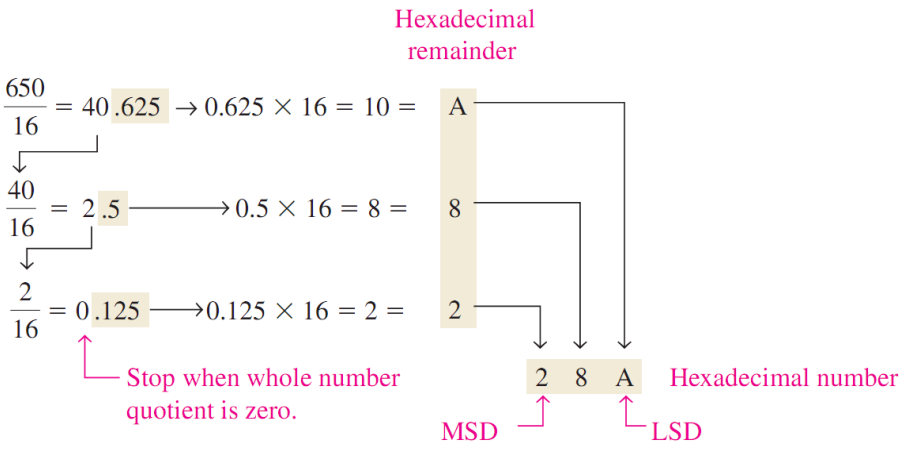
1.4.4Decimal-to-Hexadecimal Conversion

Repeated division of a decimal number by 16 will produce the equivalent hexadecimal number, formed by the remainders of the divisions.

Example

Convert the decimal number 650 to hexadecimal by repeated division by 16.

Solution



- H.W: Convert decimal 2591 to hexadecimal.

1.5 Binary Coded Decimal (BCD)

Binary coded decimal (BCD) is a way to express each of the decimal digits with a binary code. There are only ten code groups in the BCD system, so it is very easy to convert between decimal and BCD.

1.5.1 The 8421 BCD Code

The 8421 code is a type of BCD (binary coded decimal) code. Binary coded decimal means that each decimal digit, 0 through 9, is represented by a binary code of four bits. The designation 8421 indicates the binary weights of the four bits 2^3, 2^2, 2^1, 2^1. The ease of conversion between 8421 code numbers and the familiar decimal numbers is the main advantage of this code. The 8421 code is the predominant BCD code, and when we refer to BCD, we always mean the 8421 code unless otherwise stated.
You should realize that, with four bits, sixteen numbers (0000 through 1111) can be represented but that, in the 8421 code, only ten of these are used. The six code combinations that are not used (1010, 1011, 1100, 1101, 1110, and 1111) are invalid in the 8421 BCD code.

Table 4

Decimal /BCD conversion.

decimal Digit	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
BCD	**0000**	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**

To express any decimal number in BCD, simply replace each decimal digit with the appropriate 4-bit code.

Example
Convert each of the following decimal numbers to BCD:
a) 35 (b) 98 (c)170 (d) 2469
Solution

a)

$$BCD = \overbrace{0011}^{3}\overbrace{0101}^{5}$$

b)

$$BCD = \overbrace{1001}^{9}\overbrace{1000}^{8}$$

c)

$$BCD = \overbrace{0001}^{1}\overbrace{0111}^{7}\overbrace{0000}^{0}$$

d)

$$BCD = \overbrace{0010}^{2}\overbrace{0100}^{4}\overbrace{0110}^{6}\overbrace{0100}^{9}1$$

➲ H.W: Convert the decimal number 9673 to BCD.

It is equally easy to determine a decimal number from a BCD number. Start at the right-most bit and break the code into groups of four bits. Then write the decimal digit represented by each 4-bit group.

Example
Convert each of the following BCD codes to decimal:
a) 10000110 (b) 001101010001 (c) 1001010001110000
Solution

a) $\underbrace{1000}_{8}\underbrace{0110}_{6}$ decimal number = 86	*b*) $\underbrace{0011}_{3}\underbrace{0101}_{5}\underbrace{0001}_{1}$ decimal number = 351	*c*) $\underbrace{1001}_{9}\underbrace{0100}_{4}\underbrace{0111}_{7}\underbrace{0000}_{0}$ decimal number = 9470

➲ H.W: Convert the BCD code 10000010001001110110 to decimal.

1.6 BCD Addition

BCD is a numerical code and can be used in arithmetic operations. Addition is the most important operation because the other three operations (subtraction, multiplication, and division) can be accomplished by the use of addition. Here is how to add two BCD numbers:

Step 1- Add the two BCD numbers, using the rules for binary addition.
Step 2: If a 4-bit sum is equal to or less than 9, it is a valid BCD number.
Step 3: If a 4-bit sum is greater than 9, or if a carry out of the 4-bit group is generated, it is an invalid result. Add 6 (0110) to the 4-bit sum in order to skip the six invalid states and return the code to 8421. If a carry results when 6 is added, simply add the carry to the next 4-bit group.

Example
Add the following BCD numbers:
a) 0011 + 0100 (b) 00100011 + 00010101 (c) 10000110 + 00010011
(d) 010001010000 + 010000010111
Solution
The decimal number additions are shown for comparison.

a)
$$\begin{array}{r} 0011 \\ +0100 \\ \hline 0111 \end{array} \qquad \begin{array}{r} 3 \\ +4 \\ \hline 7 \end{array}$$

b)
$$\begin{array}{rr} 0010 & 0011 \\ +0001 & 0101 \\ \hline 0011 & 1000 \end{array} \qquad \begin{array}{r} 23 \\ +15 \\ \hline 38 \end{array}$$

c)
$$\begin{array}{rr} 1000 & 0110 \\ +0001 & 0011 \\ \hline 1001 & 1001 \end{array} \qquad \begin{array}{r} 86 \\ +13 \\ \hline 99 \end{array}$$

d)
$$\begin{array}{rrr} 0100 & 0101 & 0000 \\ +0100 & 0001 & 0111 \\ \hline 1000 & 0110 & 0111 \end{array} \qquad \begin{array}{r} 450 \\ +417 \\ \hline 867 \end{array}$$

➲ H.W: Add the BCD numbers: 1001000001000011 + 0000100100100101.

Example

Add the following BCD numbers:

(a) 1001 + 0100 (b) 1001 + 1001 (c) 00010110 + 00010101

(d) 01100111 + 01010011

Solution

The decimal number additions are shown for comparison.

a)

	1001		9
	+0100		+4
	1101	Invalid BCD number (> 9)	13
	+0110	Add 6	
0001	0011	valid BCD number	

b)

	1001		9
	+1001		+9
1	0010	Invalid because of carry	18
	+0110	Add 6	
0001	1000	valid BCD number	

c)

0001	0110		16
+0001	0101		+15
0010	1011	Right group is invalid (> 9), left group is valid.	31
1	+0110	Add 6 to invalid code, Add carry 0001 to next group.	
0011	0001	valid BCD number	

d)

	0110	0111		67
	+0101	0011		+53
	1011	1010	Both groups are invalid (> 9)	120
	+0110	+0110	Add 6 both groups.	
0001	0010	0000	valid BCD number	

- H.W
  1- Add the BCD numbers: 01001000 + 00110100.
  2- Convert the following decimal numbers to BCD:
  (a) 6 (b) 15 (c) 273 (d) 849
  3- What decimal numbers are represented by each BCD code?
  (a) 10001001 (b) 001001111000 (c) 000101010111

1.7 Digital Codes

1.7.1 The Gray Code

The Gray code is unweighted and is not an arithmetic code; that is, there are no specific weights assigned to the bit positions. The important feature of the Gray code is that it exhibits only a single bit change from one code word to the next in sequence. This property is important in many applications, such as shaft position encoders, where error susceptibility increases with the number of bit changes between adjacent numbers in a sequence. Table 5 is a listing of the 4-bit Gray code for decimal numbers 0 through 15. Binary numbers are shown in the table for reference. Like binary numbers, the Gray code can have any number of bits. Notice the single-bit change between successive Gray code words. For instance, in going from decimal 3 to decimal 4, the Gray code changes from 0010 to 0110, while the binary code changes from 0011 to 0100, a change of three bits. The only bit change in the Gray code is in the third bit from the right: the other bits remain the same.

Table 5

Decimal	Binary	Gray code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

1.7.2Binary-to-Gray Code Conversion

Conversion between binary code and Gray code is sometimes useful. The following rules explain how to convert from a binary number to a Gray code word:

1. The most significant bit (left-most) in the Gray code is the same as the corresponding MSB in the binary number.
2. Going from left to right, add each adjacent pair of binary code bits to get the next Gray code bit. Discard carries.

Example
Convert the following binary number 10110 to Gray code.
Solution

1	0	1	1	0	Binary
↓	↓	↓	↓	↓	
1	1	1	0	1	Gray

The Gray code is 11101.

1.7.3Gray-to-Binary Code Conversion

To convert from Gray code to binary, use a similar method; however, there are some differences. The following rules apply:

1. The most significant bit (left-most) in the binary code is the same as the corresponding bit in the Gray code.
2. Add each binary code bit generated to the Gray code bit in the next adjacent position. Discard carries.

Example
Convert the following Gray code word 11011 to binary.
Solution

1	1	0	1	1	Gray
↓	↓	↓	↓	↓	
1	0	0	1	0	Binary

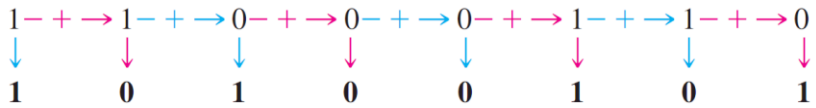
The binary number is 10010

Example

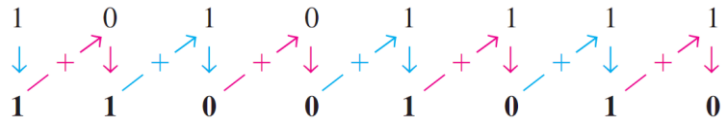
a) Convert the binary number 11000110 to Gray code.
b) Convert the Gray code 10101111 to binary

Solution

(a) Binary to Gray code:



(b) Gray code to binary:



- H.W
  a. Convert binary 101101 to Gray code.
  b. Convert Gray code 100111 to binary.

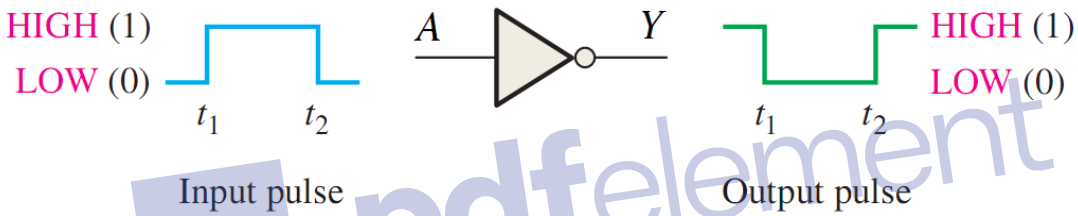
2-Logic Gates

A logic gate is an electronic circuit which makes logic decisions. It has one output and one or more inputs. The output signal appears only for certain combinations of input signals. Logic gates are the basic building blocks from which most of the digital systems.

2.1 Types of Logic Gates

1-The NOT Gate (Inverter).

It's so called because its output is NOT the same as its input. It is also called an inverter because it inverts the input signal. This is the simplest form of logic gate and has only 1 input and 1 output. Simply the purpose of this gate is to invert the input signal so if a 0 is at the input, the output will be at 1 and vice versa. The symbol for a NOT gate is as follows.



The output of a logic gate can also be summarised in the form of a table, called a 'Truth Table'. The truth table for a NOT gate is the simplest of all Truth Tables and is shown below.

Input	Output
A	Y
0	1
1	0

The Boolean expression for a NOT gate is

$$Y = \overline{A}$$

The 'bar' over the A indicates that the output Y is the opposite of A.

2-The AND gate.

The AND gate is one of the basic gates that can be combined to form any logic function. An AND gate can have two or more inputs and performs what is known as logical multiplication. The symbol is:

AND gate produces a 1 output only when all of the inputs are 1. When any of the inputs is 0, the output is 0.

AND Gate Truth Table

The logical operation of a gate can be expressed with a truth table that lists all input combinations with the corresponding outputs, as illustrated in Table for a 2-input AND gate. The truth table can be expanded to any number of inputs.

$$N = 2^n$$

Where N is the number of possible input combinations and n is the number of input variables. To illustrate,

For two input variables: $N = 2^2$ = 4 combinations

For three input variables: $N = 2^3$ = 8 combinations

For four input variables: $N = 2^4$ = 16 combinations

The truth table for the 2 input AND gate is shown below.

Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

The Boolean expression for a 2 input AND gate is

$$Y = A \cdot B$$

Where: '.' between the A and B means AND in Boolean algebra.

The AND gate with 3 inputs.

The symbol is:

$$Y = A \cdot B \cdot C$$

The truth table for the 3 input AND gate is shown below.

Inputs			Output
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Example

If two waveforms, A and B, are applied to the AND gate inputs as in Figure1, what is the resulting output waveform?

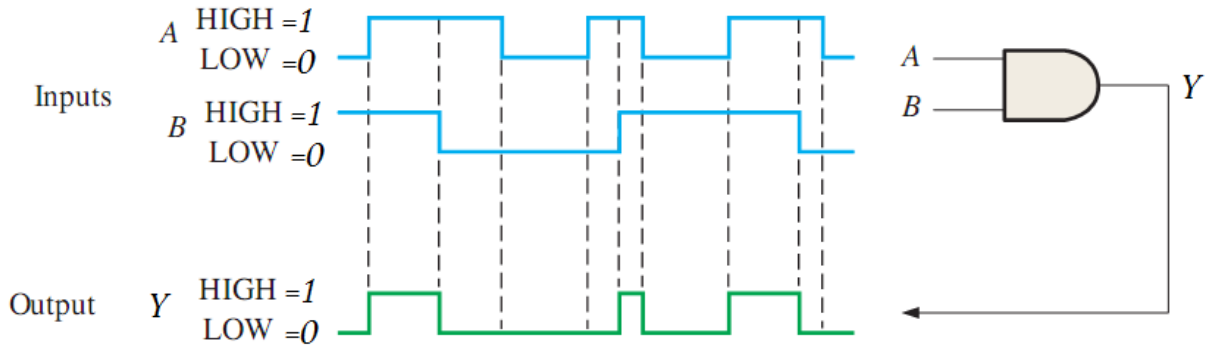
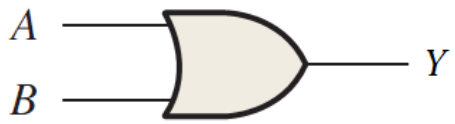


Figure 1

3-The OR gate.

The OR gate is another of the basic gates from which all logic functions are constructed. An OR gate can have two or more inputs and performs what is known as logical addition. The symbol is:



The truth table for the 2 input OR gate is shown below.

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

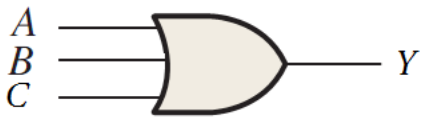
An OR gate produces a 1 on the output when any of the inputs is 1. The output is 0 only when all of the inputs are 0. The Boolean expression for a 2 input OR gate is

$$Y = A + B$$

Where: '+' between the A and B means OR in Boolean algebra.

The OR gate with 3 inputs.

The symbol is:



$$Y = A + B + C$$

The truth table for the 3 input OR gate is shown below.

Inputs			Output
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Example
If the two input waveforms, A and B, in Figure2 are applied to the OR gate, what is the resulting output waveform?
Solution

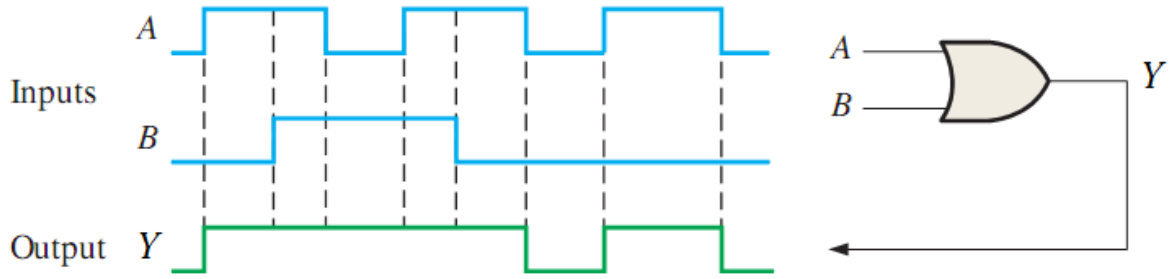


Figure 2

When either or both input waveforms are 1, the output is 1 as shown by the output waveform Y in the timing diagram.

Example
If the 3-input OR gate waveforms, A, B and C, in Figure3, what is the resulting output waveform?
Solution

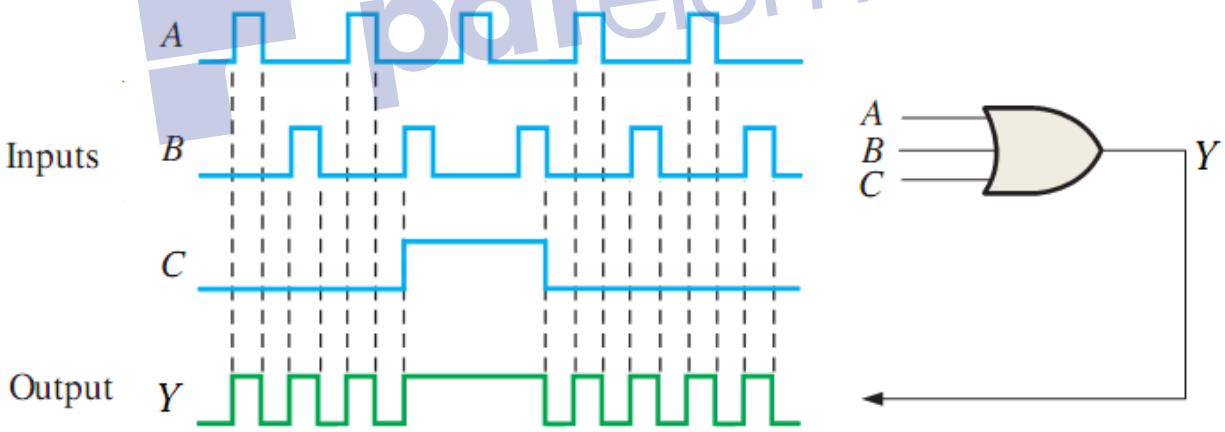
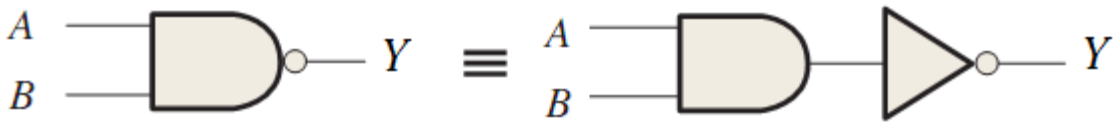


Figure 3

The output is 1 when one or more of the input waveforms are 1 as indicated by the output waveform Y in the timing diagram.

4 The NAND gate.

The NAND gate is a popular logic element because it can be used as a universal gate; that is, NAND gates can be used in combination to perform the AND, OR, and inverter operations. The symbol is:



The truth table for the 2 input NAND gate is shown below.

Inputs		Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

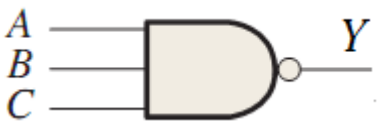
A NAND gate produces a 0 output only when all the inputs are 1. When any of the inputs is 0, the output will be 1.The Boolean expression for a 2 input NAND gate is

$$Y = \overline{A \cdot B}$$

Where: '.' between the A and B means AND, and the 'bar' means invert the output in Boolean algebra.

The NAND gate with 3 inputs.

The symbol is:



$$Y = \overline{A \cdot B \cdot C}$$

The truth table for the 3 input NAND gate is shown below.

Inputs			Output
A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Example
If the two waveforms A and B shown in Figure4 below are applied to the NAND gate inputs, determine the resulting output waveform.

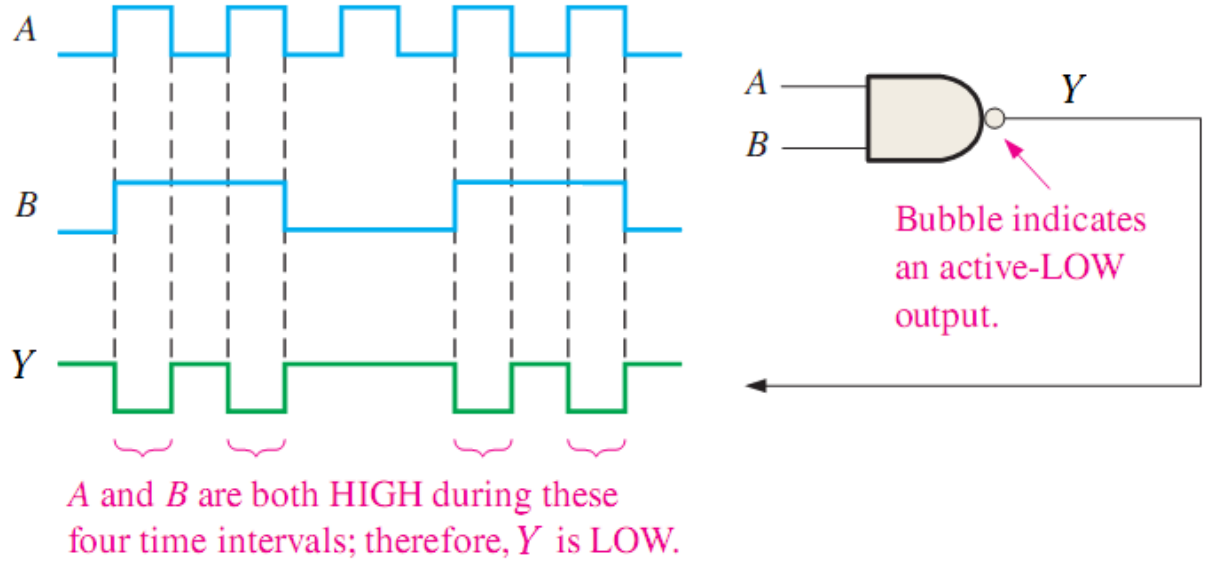


Figure 4

Output waveform Y is 0 only during the four time intervals when both input waveforms A and B are 1 as shown in the timing diagram.

Example
Show the output waveform for the 3-input NAND gate in Figure 5 with its proper time relationship to the inputs.

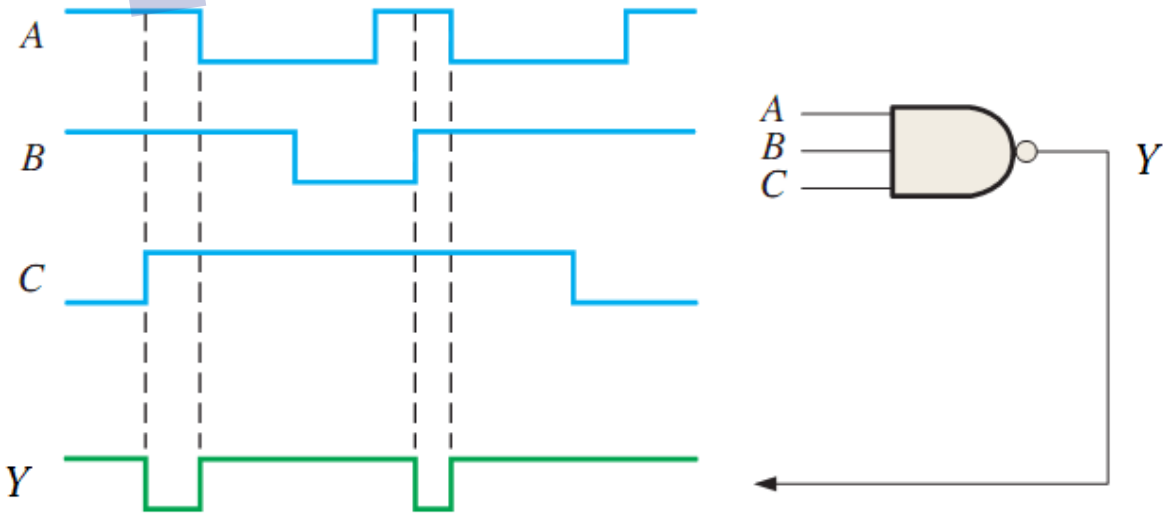
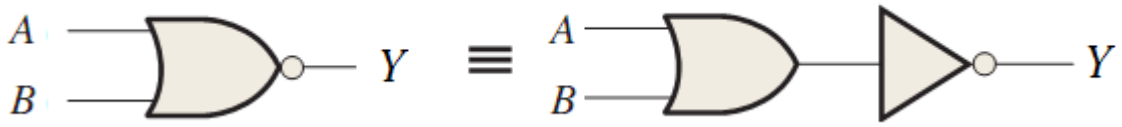


Figure 5

The output waveform Y is 0 only when all three input waveforms are 1 as shown in the timing diagram.

5-The NOR gate.

The NOR gate, like the NAND gate, is a useful logic element because it can also be used as a universal gate; that is, NOR gates can be used in combination to perform the AND, OR, and inverter operations. The symbol is:



The truth table for the 2 input NOR gate is shown below.

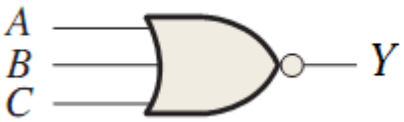
Inputs		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

A NOR gate produces a 0 output when any of its inputs is 1. Only when all of its inputs are 0 is the output HIGH. The Boolean expression for a 2 input NOR gate is

$$Y = \overline{A+B}$$

Where: '+' between the A and B means OR and the 'bar' means invert the result in Boolean Algebra.

The NOR gate with 3 input.
The symbol is:



$$Y = \overline{A+B+C}$$

The truth table for the 3 input NOR gate is shown below.

Inputs			Output
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Example
If the two waveforms shown in Figure 3–36 are applied to a NOR gate, what is the resulting output waveform?

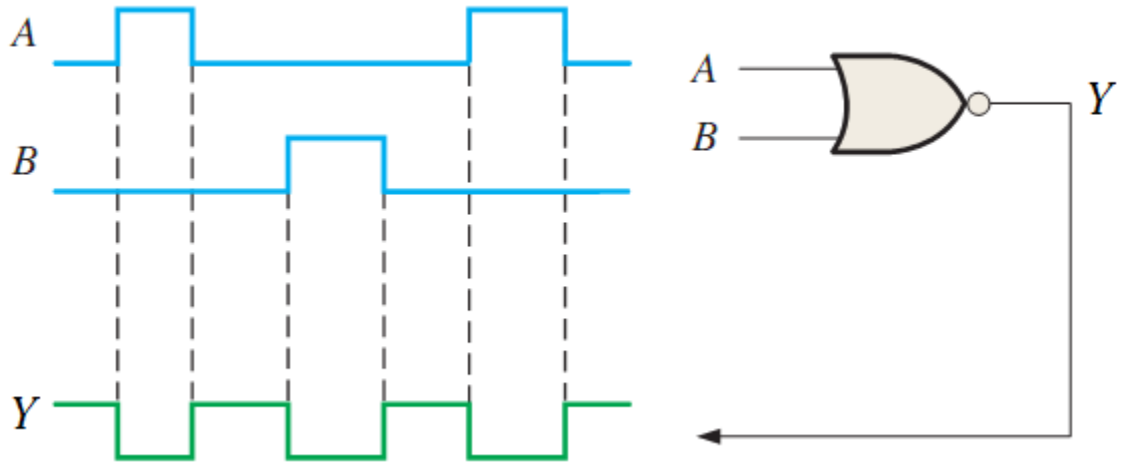


Figure 6

Whenever any input of the NOR gate is 1, the output is 0 as shown by the output waveform Y in the timing diagram.

Example
Show the output waveform for the 3-input NOR gate in Figure 7 with the proper time relation to the inputs.

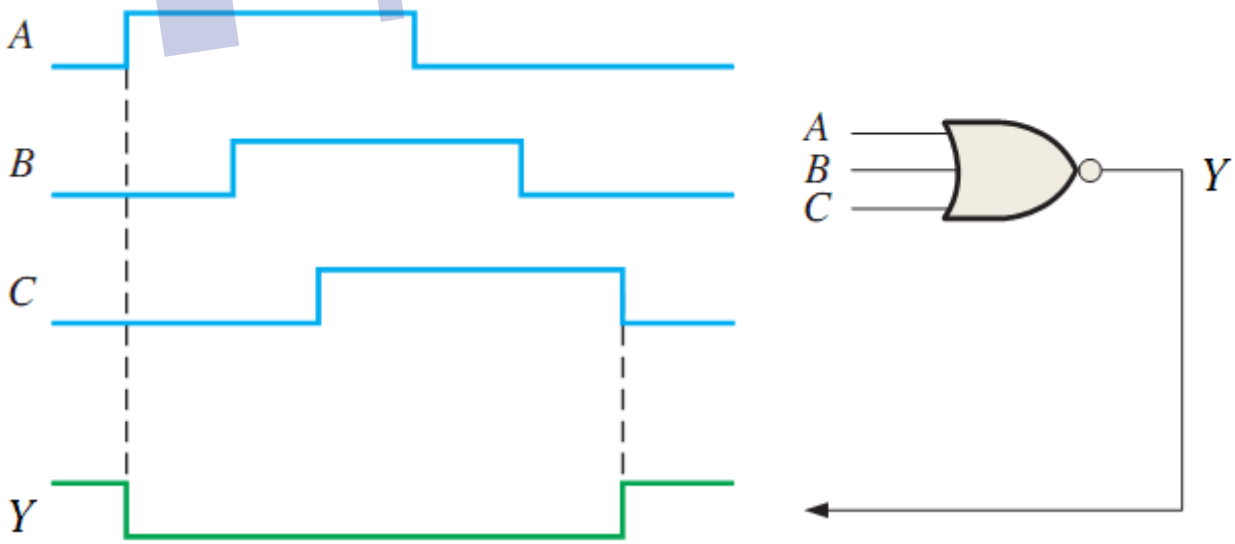


Figure 7

The output Y is 0 when any input is 1 as shown by the output waveform Y in the timing diagram.

6- The XOR gate "Exclusive-OR".

The XOR gate has 2 inputs and is a specialized version of the OR gate. The symbol for a 2 input XOR gate is as follows.

The truth table for the 2 input XOR gate is shown below.

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

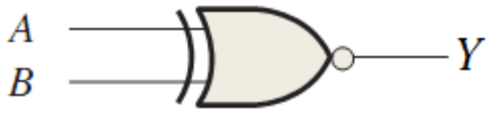
For an XOR gate, output Y is 1 when input A is 0 and input B is 1, or when input A is 1 and input B is 0; Y is 0 when A and B are both 1 or both 0.
The Boolean expression for a 2 input XOR gate is

$$Y = A \oplus B = \overline{A}B + A\overline{B}$$

The '$\oplus$' between the A and B means Exclusive –OR.

7- The XNOR gate.

The XNOR gate has 2 inputs and is the inverted form of the EXOR gate. The symbol for a 2 input XNOR gate is as follows.



The truth table for the 2 input XNOR gate is shown below.

Inputs		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

For an XNOR gate, output Y is 0 when input A is 0 and input B is 1, or when A is 1 and B is 0; Y is 1 when A and B are both 1 or both 0.

The Boolean expression for a 2 input XNOR gate is

$$Y = \overline{A \oplus B} = \overline{A}\,\overline{B} + AB$$

The '$\oplus$' between the A and B means Exclusive OR, and the 'bar' means that the result is inverted.

Example
Determine the output waveforms for the XOR gate and for the XNOR gate, given the input waveforms, A and B, in Figure 8.

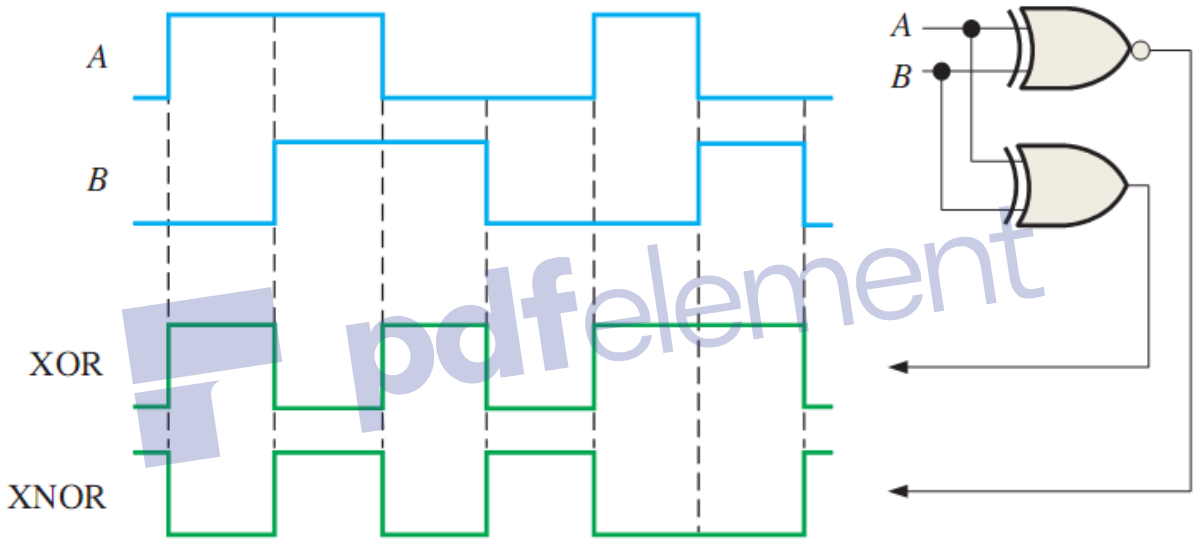


Figure 8

3-Boolean Algebra and Combinational Logic

Boolean algebra is a mathematical system based on logic. It has its own set of fundamental laws which are necessary for manipulating different Boolean expressions:

3.1-Basic rules of Boolean algebra.

Table 6			
Basic rules of Boolean algebra			
1	$\mathbf{A + 0 = A}$	7	$\mathbf{A \,.\, A = A}$
2	$\mathbf{A + 1 = 1}$ or $1+A+B+\ldots = 1$	8	$\mathbf{A \,.\, \bar{A} = 0}$
3	$\mathbf{A \,.\, 0 = 0}$	9	$\mathbf{\bar{\bar{A}} = A}$
4	$\mathbf{A \,.\, 1 = A}$	10	$\mathbf{A + AB = A}$
5	$\mathbf{A + A = A}$	11	$\mathbf{A + \bar{A}B = A + B}$
6	$\mathbf{A + \bar{A} = 1}$	12	$\mathbf{(A + B)(A + C) = A + BC}$

Example
Prove the following Boolean identities.

1- $AC + ABC = AC$
2- $(A + B)(A + C) = A + BC$
3- $A + \bar{A}B = A + B$
4- $(A + B)(A + \bar{B})(\bar{A} + C) = AC$
5- $ABC + A\bar{B}C + AB\bar{C} = A(B + C)$

Solution
4- $AC + ABC = AC$

$$AC + ABC = AC(1 + B)$$
$$= AC$$

5- $(A + B)(A + C) = A + BC$

$$(A + B)(A + C) = A \,.\, A + A \,.\, C + A \,.\, B + B.C$$
$$= A + A\,C + A\,B + BC$$
$$= A\,(1 + C + B) + BC$$
$$= A + BC$$

6- $A + \bar{A}B = A + B$

$$A + \bar{A}B = A.1 + \bar{A}B$$
$$= A(1 + B) + \bar{A}B$$
$$= A + AB + \bar{A}B$$
$$= A + B(A + \bar{A})$$
$$= A + B$$

7- $(A + B)(A + \bar{B})(\bar{A} + C) = AC$

$$(A + B)(A + \bar{B})(\bar{A} + C) = (A.A + A.\bar{B} + A.B + B.\bar{B})(\bar{A} + C)$$
$$= (A + A\bar{B} + AB + 0)(\bar{A} + C)$$
$$= A(1 + \bar{B} + B)(\bar{A} + C)$$
$$= A(1)(\bar{A} + C)$$
$$= A.\bar{A} + AC)$$
$$= AC)$$

8- $ABC + A\bar{B}C + AB\bar{C} = A(B + C)$

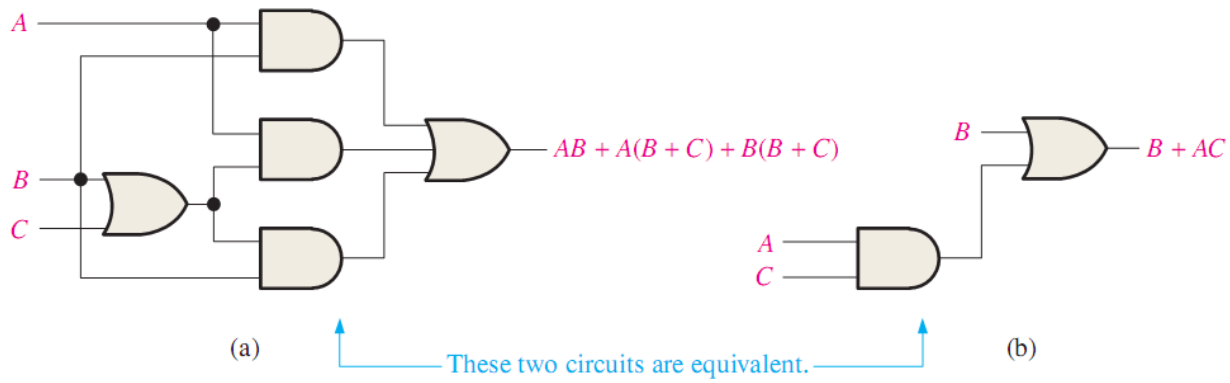
$$= AC(B + \bar{B}) + AB\bar{C}$$
$$= AC + AB\bar{C}$$
$$= A(C + B\bar{C})$$
$$= A(C + B)$$
$$= A(B + C)$$

Example

Simplify the following Boolean expression $Y = AB + A(B + C) + B(B + C)$

Solution

$$Y = AB + A(B + C) + B(B + C)$$
$$= AB + AB + AC + BB + BC$$
$$= AB + AC + B + BC$$
$$= B(A + 1 + C) + AC$$
$$Y = B + AC$$

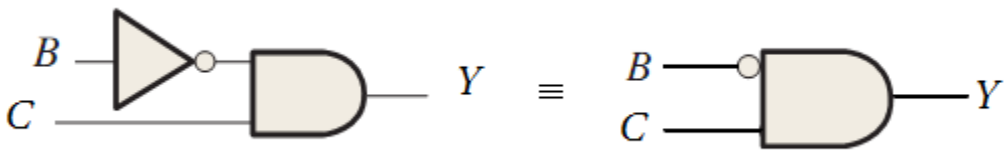


Example

Simplify the following Boolean expression: $Y = [A\bar{B}(C + BD) + \bar{A}\,\bar{B}]C$

Solution

$$
\begin{aligned}
Y &= [A\bar{B}(C + BD) + \bar{A}\,\bar{B}]C \\
&= (A\bar{B}C + A\bar{B}BD + \bar{A}\,\bar{B})C \\
&= (A\bar{B}C + 0 + \bar{A}\,\bar{B})C \\
&= A\bar{B}C + \bar{A}\,\bar{B}C \\
&= \bar{B}C(A + \bar{A}) \\
Y &= \bar{B}C
\end{aligned}
$$

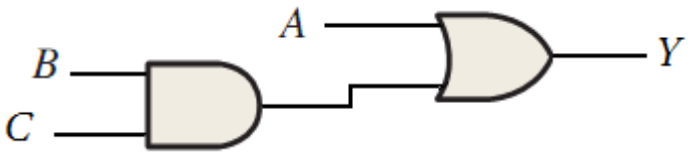


Example

Simplify the following Boolean expression and show the minimum logic gate implementation. $Y = AB\bar{C} + A\bar{B}\bar{C} + \bar{A}BC + ABC + A\bar{B}C$

Solution

$$
\begin{aligned}
Y &= AB\bar{C} + A\bar{B}\bar{C} + \bar{A}BC + ABC + A\bar{B}C \\
&= AB\bar{C} + ABC + A\bar{B}\bar{C} + A\bar{B}C + \bar{A}BC \\
&= AB(\bar{C} + C) + A\bar{B}(\bar{C} + C) + \bar{A}BC \\
&= A(B + \bar{B}) + \bar{A}BC \\
&= A + \bar{A}BC \\
Y &= A + BC
\end{aligned}
$$

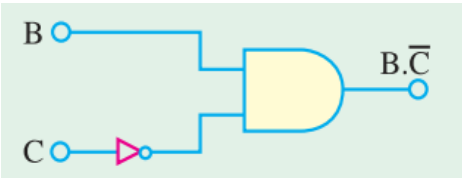


Example

Simplify the following Boolean expression and show the minimum logic gate implementation. $Y = \bar{A}B\bar{C}\bar{D} + AB\bar{C}\bar{D} + B\bar{C}D$

Solution

$$
\begin{aligned}
Y &= \bar{A}B\bar{C}\bar{D} + AB\bar{C}\bar{D} + B\bar{C}D \\
&= B\bar{C}\bar{D}(\bar{A} + A) + B\bar{C}D \\
&= B\bar{C}(\bar{D} + D) \\
Y &= B\bar{C}
\end{aligned}
$$

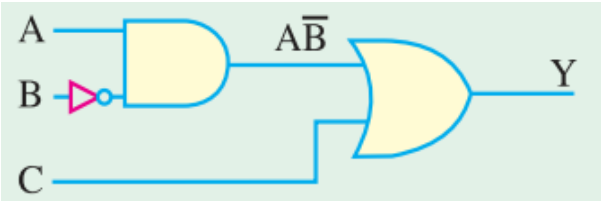


Example

Simplify the following Boolean expression and show the minimum logic gate implementation. $Y = \bar{B}(A + C) + C(\bar{A} + B) + AC$

Solution

$$Y = \bar{B}(A + C) + C(\bar{A} + B) + AC$$
$$= A\bar{B} + \bar{B}C + \bar{A}C + BC + AC$$
$$= A\bar{B} + C(\bar{B} + B + \bar{A} + A)$$
$$Y = A\bar{B} + C$$

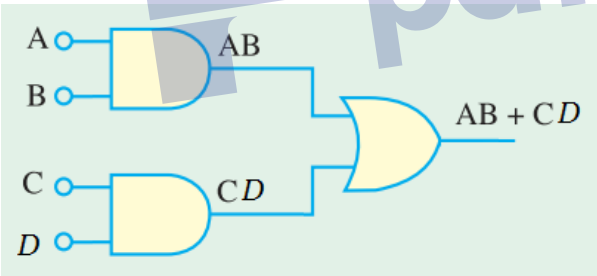


Example

Simplify the expression $Y = (AB + C)(AB + D)$

Solution

$$Y = (AB + C)(AB + D)$$
$$= AB + ABD + ABC + CD$$
$$= AB(1 + D + C) + CD$$
$$= AB + CD$$



Example

Prove that $\mathbf{A} + \bar{\mathbf{A}}\mathbf{B} = \mathbf{A} + \mathbf{B}$ by using truth table.

Solution

A	B	$\bar{A}$	$\bar{A}B$	$A + \bar{A}B$	$A + B$
0	0	1	0	0	0
0	1	1	1	1	1
1	0	0	0	1	1
1	1	0	0	1	1

Example

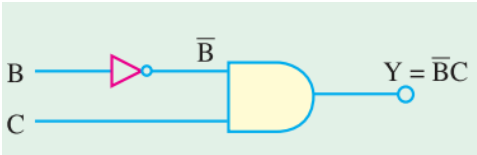
Determine the logic expression for the output Y from the following truth table. Simplify and sketch the logic circuit for the simplified expression.

Solution

1-

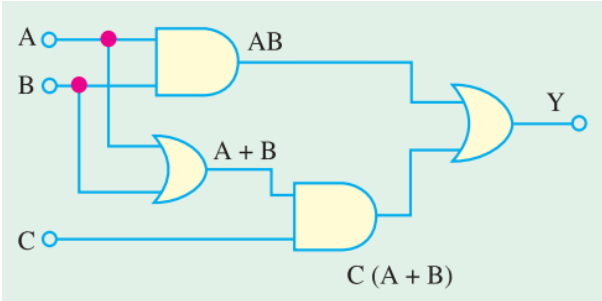
A	B	C	Y	
0	0	0	0	
0	0	1	1	$\bar{A}\bar{B}C$
0	1	0	0	
0	1	1	0	
1	0	0	0	
1	0	1	1	$A\bar{B}C$
1	1	0	0	
1	1	1	0	

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



A	B	C	Y	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$\bar{A}BC$
1	0	0	0	
1	0	1	1	$A\bar{B}C$
1	1	0	1	$AB\bar{C}$
1	1	1	1	ABC

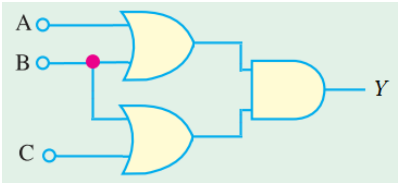
$$Y = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$
$$Y = \bar{A}BC + A\bar{B}C + AB(\bar{C} + C)$$
$$Y = \bar{A}BC + A\bar{B}C + AB$$
$$Y = B(\bar{A}C + A) + A\bar{B}C$$
$$Y = BC + AB + A\bar{B}C$$
$$Y = BC + A(B + \bar{B}C)$$
$$Y = BC + AB + AC)$$
$$Y = C(A + B) + AB$$



Example:

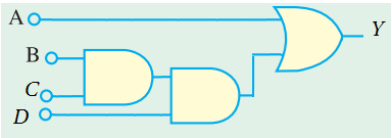
State the logic functions performed by the circuits below

a-



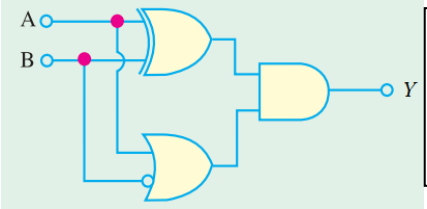
$$Y = (A + B).(B + C)$$

b-



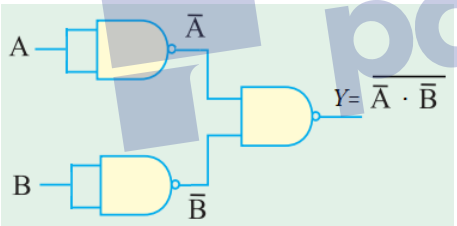
$$Y = A + (B.C.D)$$

c-

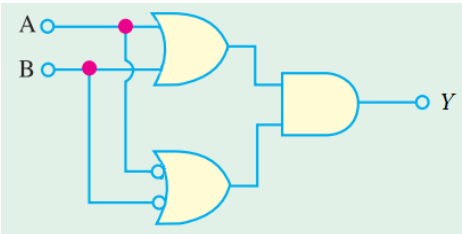


$$\begin{aligned} Y &= (\bar{A}B + A\bar{B})(A + \bar{B}) \\ &= (\bar{A}B + A\bar{B})(A + \bar{B}) \\ &= (0 + 0 + A\bar{B} + A\bar{B}) \\ Y &= A\bar{B} \end{aligned}$$

d-

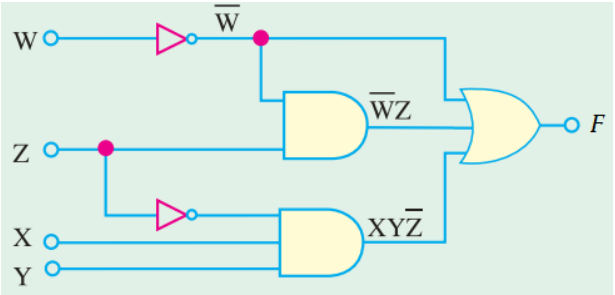


e-



$$\begin{aligned} Y &= (\bar{A} + \bar{B})(A + B) \\ &= (0 + \bar{A}B + A\bar{B} + 0) \\ &= \bar{A}B + A\bar{B} \\ Y &= A \oplus B \end{aligned}$$

f-



$$\begin{aligned} F &= \bar{W} + \bar{W}Z + XY\bar{Z} \\ F &= \bar{W}(1 + Z) + XY\bar{Z} \\ F &= \bar{W} + XY\bar{Z} \end{aligned}$$

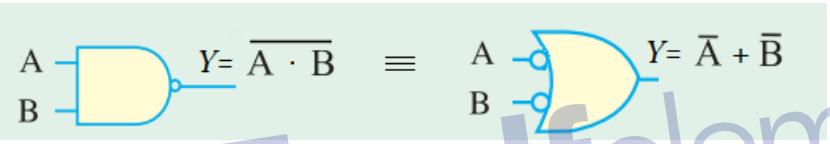
3.2-DeMorgan's Theorems

These theorems consist of two rules of a great help in simplifying complicated logical expression. It can be stated as follows.

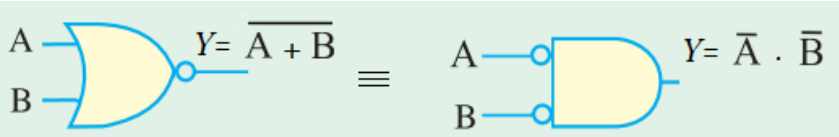
1- $\overline{A+B} = \overline{A}.\overline{B}$
2- $\overline{A.B} = \overline{A} + \overline{B}$

The first statement says that *the complement of a sum equals the product of complements*. The second statement says that *the complement of a product equals the sum of the complements*. In fact, it allows transformation from a sum-of-products from to a product-of-sum from.

These rules are illustrated by the gate equivalencies and truth tables in the following figure.



A	B	$\overline{A.B}$	$\overline{A} + \overline{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0



A	B	$\overline{A+B}$	$\overline{A}.\overline{B}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Example

Simplify the following Boolean expression $Y = \overline{\overline{\overline{A + B\bar{C}} + D(\overline{E + \bar{F}})}}$

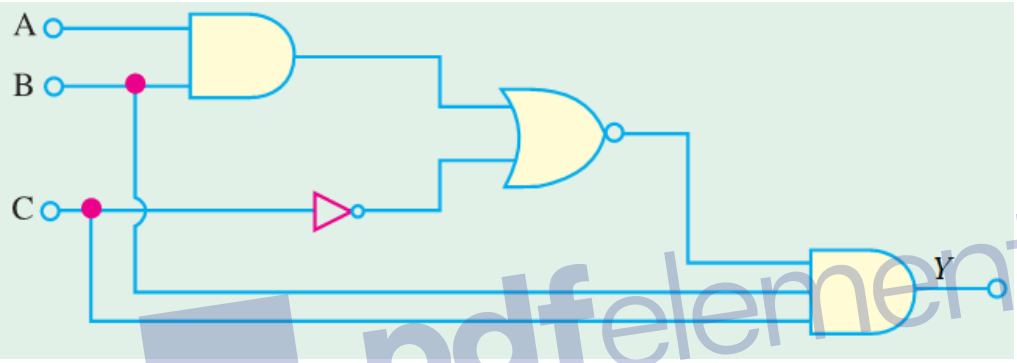
Solution

$$Y = \overline{\overline{\overline{A + B\bar{C}} + D(\overline{E + \bar{F}})}}$$
$$Y = \left(\overline{\overline{A + B\bar{C}}}\right).\overline{D(\overline{E + \bar{F}})}$$
$$Y = (A + B\bar{C}).(\bar{D} + E + \bar{F})$$

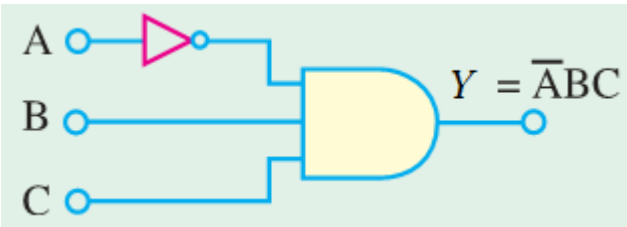
Example

Determine the Boolean expression for the logic circuit shown below. Simplify the Boolean expression using Boolean Laws and De Morgan's theorem. Redraw the logic circuit using the simplified Boolean expression.

Solution



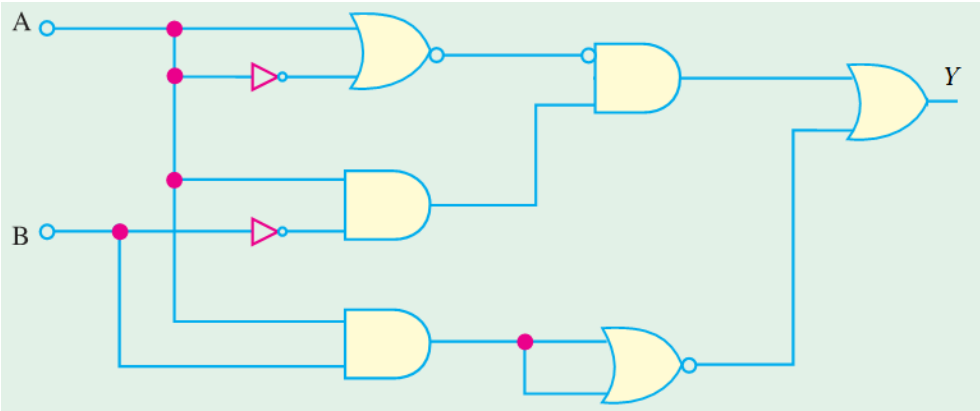
$$Y = (BC).\left(\overline{AB + \bar{C}}\right)$$
$$Y = (BC).\left(\overline{AB}\bar{\bar{C}}\right)$$
$$Y = (BC).(\bar{A} + \bar{B})C$$
$$Y = (BC).(\bar{A}C + \bar{B}C)$$
$$Y = (BC).(\bar{A}C + \bar{B}C)$$
$$Y = \bar{A}BC$$



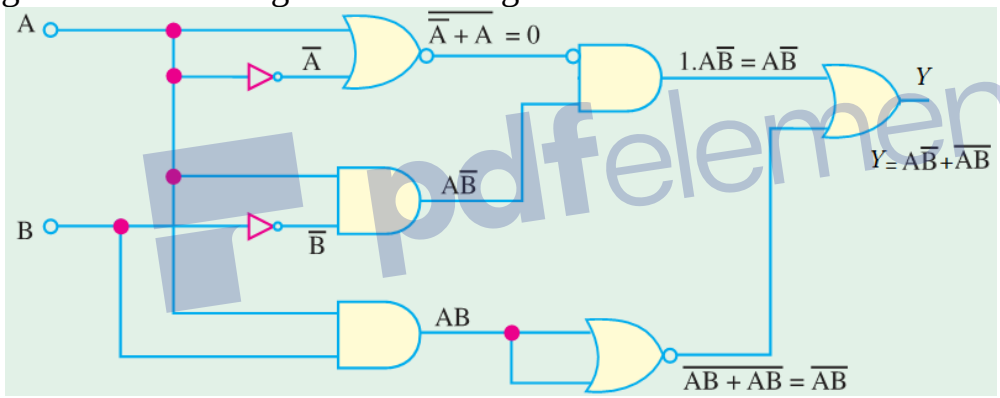
Example

Determine the output of the logic circuit shown in Fig. below. Simplify the output Boolean expression and sketch the logic circuit.

Solution



The output of the circuit can be obtained by determining the output of each logic gate while working from left to right.

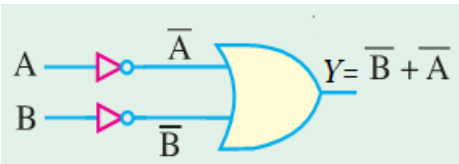


$Y = (A\bar{B}) + (\overline{AB})$

$Y = A\bar{B} + \bar{A} + \bar{B}$

$Y = \bar{B}(A + 1).\bar{A}$

$Y = \bar{A} + \bar{B}$



3.2.1The Universal Property of NAND and NOR Gates

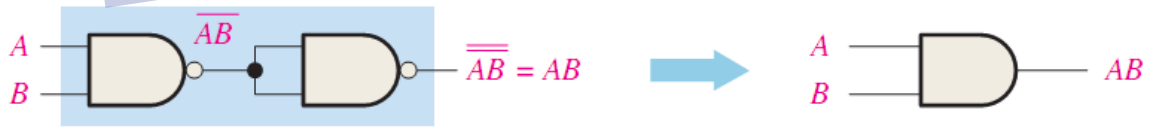
The universality of the NAND gate means that it can be used as an inverter and that combinations of NAND gates can be used to implement the AND, OR, and NOR operations. Similarly, the NOR gate can be used to implement the inverter (NOT), AND, OR, and NAND operations.

The NAND Gate as a Universal Logic Element

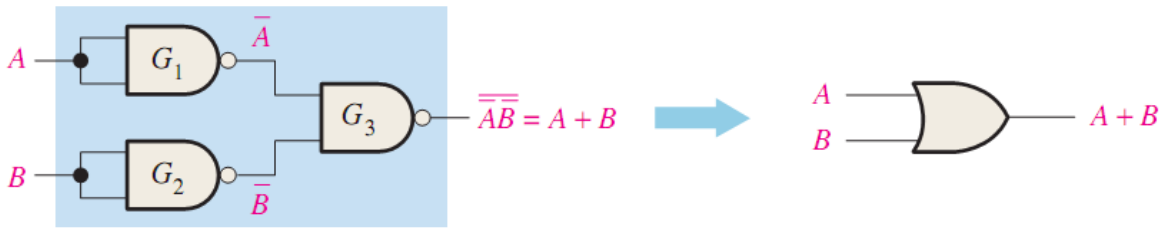
The NAND gate is a universal gate because it can be used to produce the NOT, the AND, the OR, and the NOR functions. An inverter can be made from a NAND gate by connecting all of the inputs together and creating, in effect, a single input, as shown in Figure below part(a) for a 2-input gate. An AND function can be generated by the use of NAND gates alone, as shown in Figure below part (b). An OR function can be produced with only NAND gates, as illustrated in part (c). Finally, a NOR function is produced as shown in part (d).

Combinations of NAND gates can be used to produce any logic function.

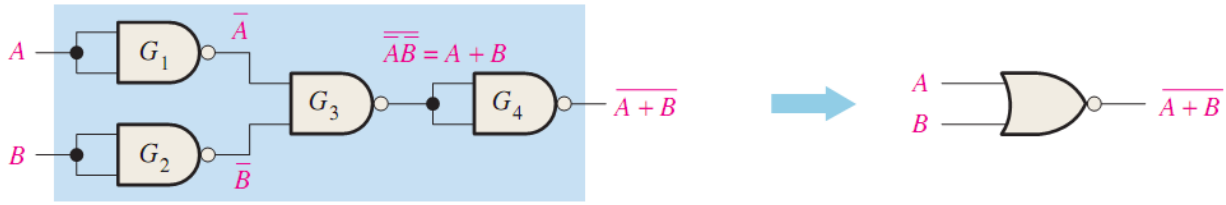
(a) One NAND gate used as an inverter



(b) Two NAND gates used as an AND gate



(c) Three NAND gates used as an OR gate

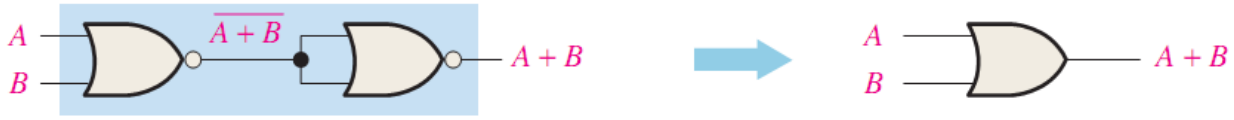


(d) Four NAND gates used as a NOR gate

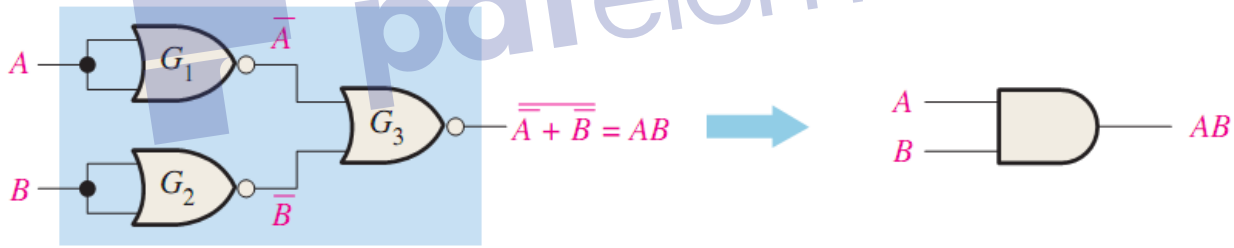
The NOR Gate as a Universal Logic Element

Like the NAND gate, the NOR gate can be used to produce the NOT, AND, OR, and NAND functions. A NOT circuit, or inverter, can be made from a NOR gate by connecting all of the inputs together to effectively create a single input, as shown in Figure below part (a) with a 2-input example. Also, an OR gate can be produced from NOR gates, as illustrated in Figure below part (b). An AND gate can be constructed by the use of NOR gates, as shown below

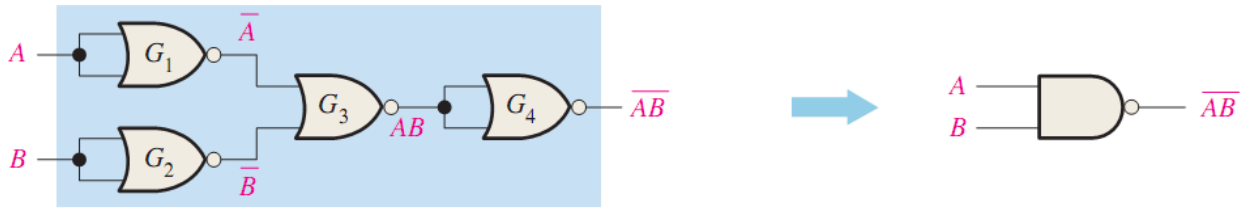
(a) One NOR gate used as an inverter



(b) Two NOR gates used as an OR gate



(c) Three NOR gates used as an AND gate



3.3Standard Forms of Boolean Expressions

All Boolean expressions, regardless of their form, can be converted into either of two standard forms: the sum-of-products form or the product-of-sums form. Standardization makes the evaluation, simplification, and implementation of Boolean expressions much more systematic and easier.

3.3.1 The Sum-of-Products (SOP) Form

A product term was defined as a term consisting of the product (Boolean multiplication) of literals (variables or their complements). When two or more product terms are summed by Boolean addition, the resulting expression is a sum-of-products (SOP). Some examples are

$AB + ABC$

$ABC + CDE + \bar{B}C\bar{D}$

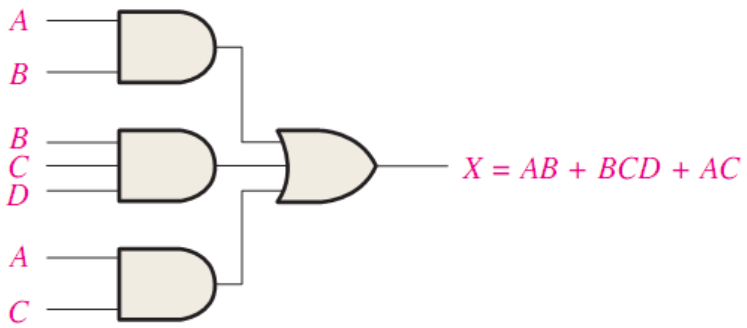
$\bar{A}B + \bar{A}B\bar{C} + AC$

Also, an SOP expression can contain a single-variable term, as in

$A + AB\bar{C} + BC\bar{D}$.

In an SOP expression a single overbar cannot extend over more than one variable; however, more than one variable in a term can have an overbar. For example, an SOP expression can have the term $\bar{A}\,\bar{B}\,\bar{C}$ but not $\overline{ABC}$.

Implementing an SOP expression simply requires ORing the outputs of two or more AND gates. A product term is produced by an AND operation, and the sum (addition) of two or more product terms is produced by an OR operation. Therefore, an SOP expression can be implemented by AND-OR logic in which the outputs of a number as shown below for the expression$AB + BCD + AC$. The output X of the OR gate equals the SOP expression.



Converting Product Terms to Standard SOP:

Each product term in an SOP expression that does not contain all the variables in the domain can be expanded to standard SOP to include all variables in the domain and their complements. As stated in the following steps, a nonstandard SOP expression is converted into standard form using Boolean algebra rule ($A + \bar{A} = 1$) from Table 6: A variable added to its complement equals 1.

Step 1: Multiply each nonstandard product term by a term made up of the sum of a missing variable and its complement. This results in two product terms.

As you know, you can multiply anything by 1 without changing its value.

Step 2: Repeat Step 1 until all resulting product terms contain all variables in the domain in either complemented or uncomplemented form. In converting a product term to standard form, the number of product terms is doubled for each missing variable.

Example

Convert the following Boolean expression into standard SOP form:

$$A\bar{B}C + \bar{A}\bar{B} + AB\bar{C}D$$

Solution

The domain of this SOP expression A, B, C, D.

The first term:

$$A\bar{B}C = A\bar{B}C(D + \bar{D}) = A\bar{B}CD + A\bar{B}C\bar{D}$$

In this case, two standard product terms are the result.

The second term,

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

$$\bar{A}\bar{B}C(D + \bar{D}) + \bar{A}\bar{B}\bar{C}(D + \bar{D}) = \bar{A}\bar{B}CD + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D}$$

In this case, four standard product terms are the result.

The third term, $AB\bar{C}D$, is already in standard form.

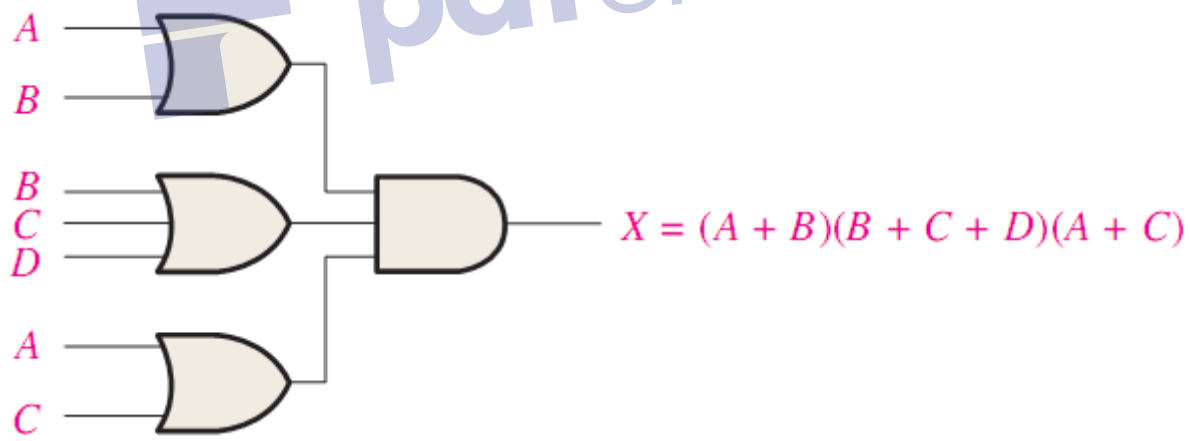
The complete standard SOP form of the original expression is as follows:

$$A\bar{B}CD + A\bar{B}C\bar{D} + \bar{A}\bar{B}CD + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D} + AB\bar{C}D$$

3.3.2 Product of Sums (POS) Form

A sum term was defined before as a term consisting of the sum (Boolean addition) of literals (variables or their complements). When two or more sum terms are multiplied, the resulting expression is a product-of-sums (POS). Some examples are

$(\bar{A} + B)(A + \bar{B} + C)$
$(A + \bar{B} + \bar{C})(C + \bar{D} + E)(B + C + D)$
$(A + \bar{B})(A + \bar{B} + C)(A + C)$
A POS expression can contain a single-variable term, as in
$A(A + B + C)(B + C + D)$.
In a POS expression, a single overbar cannot extend over more than one variable; however, more than one variable in a term can have an overbar.
For example, a POS expression can have the term $\bar{A} + \bar{B} + \bar{C}$ but not $\overline{A + B + C}$.
Implementation of a POS Expression simply requires ANDing the outputs of two or more OR gates. A sum term is produced by an OR operation and the product of two or more sum terms is produced by an AND operation. Figure below shows for the expression$(A + B)(B + C + D)(A + C)$. The output X of the AND gate equals the POS expression.



The Standard POS Form

So far, you have seen POS expressions in which some of the sum terms do not contain all of the variables in the domain of the expression. For example, the expression

$$(A + \bar{B} + C)\,(A + B + \bar{D})\,(A + \bar{B} + \bar{C} + D)$$

has a domain made up of the variables A, B, C, and D. Notice that the complete set of variables in the domain is not represented in e first two terms of the expression; that is, D or $\bar{D}$ is missing from the first term and C or $\bar{C}$ is missing from the second term.
A standard POS expression is one in which all the variables in the domain appear in each sum term in the expression. For example,

$(\bar{A} + \bar{B} + C + D)(A + \bar{B} + C + D)(A + B + C + D)$
is a standard POS expression. Any nonstandard POS expression (referred to simply as POS) can be converted to the standard form using Boolean algebra.

Converting a Sum Term to Standard

POS Each sum term in a POS expression that does not contain all the variables in the domain can be expanded to standard form to include all variables in the domain and their complements. As stated in the following steps, a nonstandard POS expression is converted into standard form using Boolean algebra rule $(A\,.\bar{A} = 0)$ from Table 6:

Step 1: Add to each nonstandard product term a term made up of the product of the missing variable and its complement. This results in two sum terms.
As you know, you can add 0 to anything without changing its value.

Step 2: Apply the rule from Table 6: $A + BC = (A + B)(A + C)$

Step 3: Repeat Step 1 until all resulting sum terms contain all variables in the domain in either complemented or noncomplemented form.

Example
Convert the following Boolean expression into standard POS form:

$$(A + \bar{B} + C)(\bar{B} + C + \bar{D})(A + \bar{B} + \bar{C} + D)$$

Solution
The domain of this POS expression is A, B, C, D.
The first term,

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

The second term,

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

The third term, $(A + \bar{B} + \bar{C} + D)$, is already in standard form.
The standard POS form of the original expression is as follows:
$(\mathbf{A} + \bar{\mathbf{B}} + \mathbf{C} + \mathbf{D})(\mathbf{A} + \bar{\mathbf{B}} + \mathbf{C} + \mathbf{D})(\mathbf{A} + \bar{\mathbf{B}} + \mathbf{C} + \bar{\mathbf{D}})(\bar{\mathbf{A}} + \bar{\mathbf{B}} + \mathbf{C} + \bar{\mathbf{D}})(\mathbf{A} + \bar{\mathbf{B}} + \bar{\mathbf{C}} + \mathbf{D})$

3.4 Minterms and Maxterms

3.4.1 Minterms

Each row of a truth table can be associated with a *minterm*, which is a product (AND) of all variables in the function, in direct or complemented form. A minterm has the property that it is equal to 1 on exactly one row of the truth table.

Here is the three-variable truth table and the corresponding minterms:

A	*B*	*C*	minterm
0	**0**	**0**	$\bar{A}\bar{B}\bar{C} = m_0$
0	**0**	**1**	$\bar{A}\bar{B}C = m_1$
0	**1**	**0**	$\bar{A}B\bar{C} = m_2$
0	**1**	**1**	$\bar{A}BC = m_3$
1	**0**	**0**	$A\bar{B}\bar{C} = m_4$
1	**0**	**1**	$A\bar{B}C = m_5$
1	**1**	**0**	$AB\bar{C} = m_6$
1	**1**	**1**	$ABC = m_7$

The subscript on the minterm is the number of the row on which it equals 1. (The row numbers are obtained by reading the values of the variables on that row as a binary number).

Minterms provide a way to represent any Boolean function algebraically, once its truth table is specified. The function is given by the sum (OR) of those minterms corresponding to rows where the function is 1. By the minterm property, the OR will contain a term equal to 1 (making the function 1) on exactly those rows where the function is supposed to be 1.

Example:
Suppose a function F is defined by the following truth table:

A	B	C	F
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**0**
1	1	1	**1**

Since $F = 1$ on rows 1, 2, 4, and 7, we obtain

$$\boldsymbol{F = m_1 + m_2 + m_4 + m_7}$$

$$F = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$

A compact notation is to write only the numbers of the minterms included in F, using the Greek letter capital sigma to indicate a sum:

$$F = \sum(1,2,4,7)$$

This form can be written down immediately by inspection of the truth table.

3.4.2 Maxterm

Each row of a truth table is also associated with a *maxterm*, which is a sum (OR) of all the variables in the function, in direct or complemented form. A maxterm has the property that it is equal to 0 on exactly one row of the truth table.

Here is the three-variable truth table and the corresponding maxterms:

A	B	C	maxterms
0	**0**	**0**	$\boldsymbol{A + B + C = M_0}$
0	**0**	**1**	$\boldsymbol{A + B + \bar{C} = M_1}$
0	**1**	**0**	$\boldsymbol{A + \bar{B} + C = M_2}$
0	**1**	**1**	$\boldsymbol{A + \bar{B} + \bar{C} = M_3}$
1	**0**	**0**	$\boldsymbol{\bar{A} + B + C = M_4}$
1	**0**	**1**	$\boldsymbol{\bar{A} + B + \bar{C} = M_5}$
1	**1**	**0**	$\boldsymbol{\bar{A} + \bar{B} + C = M_6}$
1	**1**	**1**	$\boldsymbol{\bar{A} + \bar{B} + \bar{C} = M_7}$

Like minterms, maxterms also provide a way to represent any Boolean function algebraically once its truth table is specified. The function is given by the product (AND) of those maxterms corresponding to rows where the function is 0. By the maxterm property, the AND will contain a term equal to 0 (making the function 0) on exactly those rows where the function is supposed to be 0.

Example:
For the same function as previously, we observe that it is 0 on rows 0, 3, 5, and 6.
Solution

A	*B*	*C*	*F*
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**0**
1	1	1	**1**

$$F = M_0 . M_3 . M_5 . M_6$$
$$F = (A + B + C)(A + \bar{B} + \bar{C})(\bar{A} + B + \bar{C})(\bar{A} + \bar{B} + C)$$

This form also lends itself to a compact notation: using the Greek letter capital pi to denote a product, we write only the numbers of the maxterms included in F:

$$F = \prod(0,3,5,6)$$

Two Boolean functions are equivalent if their Π forms are the same.

- Note that each maxterm is the complement of its corresponding minterm and vice versa.

3.5 KARNAUGH MAP MINIMIZATION

A Karnaugh map provides a systematic method for simplifying Boolean expressions and, if properly used, will produce the simplest SOP or POS expression possible.

The map format:- the k-map is composed of an arrangement of adjacent cells each representing are particular combination of variables in product form. Since the total number of combination of n variables and their complement is 2^n , the k-map consist of 2^n cells. For example, there are four combinations of the products of two variables (A and B) and their complements $\bar{A}\bar{B}, \bar{A}B, A\bar{B}$ and AB, therefore, the k-map must have four cells, with each cell representing one of the variables combinations , as illustrated below.

	$\bar{A}$	A
$\bar{B}$	$\bar{A}\bar{B}$	$A\bar{B}$
B	$\bar{A}B$	AB

	$\bar{A}$	A
$\bar{B}$	0	2
B	1	3

Extensions of the k-map to three and four variables are shown below

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	0	2	6	4
C	1	3	7	5

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	0	4	12	8
$\bar{C}D$	1	5	13	9
CD	3	7	15	11
$C\bar{D}$	2	6	14	10

Notice that the cells are arranged such that there is only a single variable change between any adjacent cells.

K-maps of five, six or more variables can be constructed, but they are quite impractical except when implemented a computer.

- Plotting a Boolean expression:- Once a Boolean expression is in the sum-of-product form, you can plot it on the k-map by placing a 1 in each cell corresponding to a term in the sum-of-products expression. For example, the 3-variable expression $\bar{A}BC + AB\bar{C} + ABC$ is plotted in the k-map below.

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	0	0	1	0
C	0	1	1	0

- Grouping cells for simplification:- You can group 1's that are in adjacent cells according to the following rules by drawing a loop around those cells:
  1- Adjacent cells are cells that differ by only one variable (for example $ABCD$ and$ABC\bar{D}$.
  2- The 1's in adjacent cells must be command in groups of 1, 2, 4, 8, 16, and so on.
  3- Each group of 1's should be maximized to include the largest number of adjacent cells as possible in accordance with rule 2.
  4- Every 1's in the map must be included in at least one group. There can be overlapping groups if they include non common 1's.
  For example

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	0	0	1	1
$\bar{C}D$	1	1	1	1
CD	1	1	1	1
$C\bar{D}$	0	1	0	0

- Simplifying the expression:- In order to write the simplified Boolean expression, follow the rules:
  1- Each group of 1's creates a product term composed of all variables that appear in only one form (complemented or uncomplemented) within the group variables that appear both uncomplemented and complemented are eliminated.
  2- The final simplified expression is formed by summing the product terms of all the groups. for example

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	0	0	1	1
$\bar{C}D$	1	1	1	1
CD	1	1	1	1
$C\bar{D}$	0	1	0	0

$A\bar{C}$

D

$\bar{A}BC$

The simplified expression:- $F = A\bar{C} + D + \bar{A}BC$

❖ Summary of using the k-map

1- Construct a 2^n-cell k-map for the n variables
2- Put 1's in the cells corresponding to the terms of the Boolean expression to be simplified, and put 0's elsewhere.
3- Combine the cells containing 1's as you have learned before.
4- Write the simplified expression.

Example

Minimize the expression:- $X = A\bar{B}C + \bar{A}BC + \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C}$

Solution

$X = \bar{A}C + \bar{B}$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	1	0	0	1
C	1	1	0	1

Example

Reduce the following 4-variables function to its minimum sum-of-product form:-

$\boldsymbol{X = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}\bar{D} + AB\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + \bar{A}BCD + \bar{A}BCD + \bar{A}BCD + ABC\bar{D} + A\bar{B}C\bar{D}}$

Solution

$X = \bar{B}C + \bar{D}$

We have simplified X from ten 4-inputs ANDs and one 10-input OR to one 2-input AND and one 2-input OR.

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	1	1	1	1
$\bar{C}D$	0	0	0	0
CD	1		0	1
$C\bar{D}$	1	1	1	1

Example

Minimize the expression:-

$X = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + A\bar{B}\bar{C}\bar{D} + \bar{A}CD + A\bar{B}C\bar{D}$

Solution

The term $\bar{A}CD$ must be expanded into $\bar{A}BCD$ and $\bar{A}\bar{B}CD$ to get the standard SOP expression, which is then mapped; the cells are grouped as shown below

$\bar{A}CD(B + \bar{B}) = \bar{A}BCD + \bar{A}\bar{B}CD$

$X = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + A\bar{B}\bar{C}\bar{D} + \bar{A}BCD + \bar{A}\bar{B}CD + A\bar{B}C\bar{D}$

$X = \sum(0,2,3,7,8,10)$

The corner of the map can be grouped together when they are 1's

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	1	0	0	1
$\bar{C}D$	0	0	0	0
CD	1	1	0	0
$C\bar{D}$	1	0	0	1

$X = \bar{B}\bar{D} + \bar{A}CD$

Example

Use a Karnaugh map to minimize the following standard POS expression:

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

Solution

$F = \prod(0,1,2,3,6)$

$\bar{F} = \bar{A} + B\bar{C}$

$F = \overline{\bar{A} + B\bar{C}}$

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

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	0	0	0	1
C	0	0	1	1

Example

Use a Karnaugh map to minimize the following POS expression:

$\mathbf{F} = (\boldsymbol{B}+\boldsymbol{C}+\boldsymbol{D})(\bar{\boldsymbol{A}}+\boldsymbol{B}+\boldsymbol{C}+\boldsymbol{D})(\boldsymbol{A}+\boldsymbol{B}+\bar{\boldsymbol{C}}+\boldsymbol{D})(\bar{\boldsymbol{A}}+\boldsymbol{B}+\boldsymbol{C}+\bar{\boldsymbol{D}})(\boldsymbol{A}+\bar{\boldsymbol{B}}+\boldsymbol{C}+\boldsymbol{D})(\bar{\boldsymbol{A}}+\bar{\boldsymbol{B}}+\boldsymbol{C}+\boldsymbol{D})$

Solution

The term (B + C + D)must be expanded into $A+B+C+D$ and $\bar{A}+B+C+D$ to get a standard POS expression, which is then mapped; and the cells are grouped as shown in Figure below.

$F = \prod(0,2,4,8,9,12)$

$\bar{F} = \bar{C}\bar{D} + A\bar{B}\bar{C} + \bar{A}\bar{B}\bar{D}$

$F = (C+D)(\bar{A}+B+C)(A+B+C)$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	0	0	0	0
$\bar{C}D$	1	1	1	0
CD	1	1	1	1
$C\bar{D}$	0	1	1	1

Don't-Care

In some situations, we don't care about the value of a logic function. For example, if we use A, B, C, D to represent a number from 0 to 9, we need not worry about the function value produced for $A, B, C, D = 10 \ldots 15$.

For these situations, the function can be assigned an output in order to make the resulting circuit as simple as possible.

Suppose we wish to implement the function

$F(A,B,C,D) = \sum(3,5,6,7)$

And we have the don't-care condition of

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

The sum-of-products implementation:

$F = CD + BD + BC$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	0	0	x	0
$\bar{C}D$	0	1	x	0
CD	1	1	x	x
$C\bar{D}$	0	1	x	x

Example

Simplify the following Boolean function, with the don't-care conditions d

:$F(A,B,C,D) = \sum(1,7,10,11,13) + \sum d(5,15)$

Solution

$F = BD + \bar{A}\bar{C}D + A\bar{B}C$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	0	0	0	0
$\bar{C}D$	1	x	1	0
CD	0	1	x	1
$C\bar{D}$	0	0	0	1

Example

Simplify the following Boolean function, with the don't-care conditions d:

$F(A,B,C,D) = \sum(0,1,2,4,5) + \sum d(3,6,7)$

Solution

$F = 1$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	1	1	X	1
C	1	X	X	1

Example

Simplify the following Boolean function, with the don't-care conditions d:

$F(A,B,C,D) = \sum(0,6,8,13,14) + \sum d(2,4,10)$

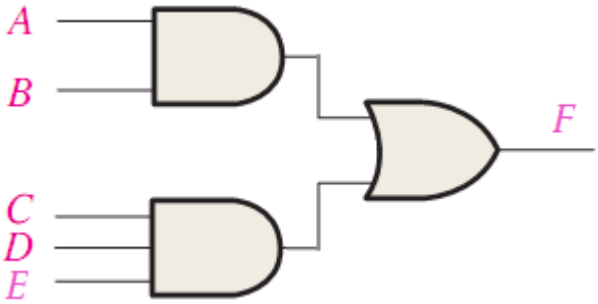
Solution

$F = C\bar{D} + \bar{B}\bar{D} + AB\bar{C}D$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	1	x	0	1
$\bar{C}D$	0	0	1	0
CD	0	0	0	0
$C\bar{D}$	x	1	1	x

3.6 Designing combinational logic circuits

In this section we start with an equation or truth table that describes algebraic function and from it we will determine the circuit required to implement the function. For example, the Boolean expression: $F = AB + CDE$. by inspection we can tell that this function is composed of two terms AB and CDE the first term could be implemented by ANDing A with B and similarly , that second term could be implemented by ANDing C,D and E together. Next, the output forms the first and the second AND gates are ORed to give the final value of the function F as shown below.



Sometimes, we'll begin with the truth table for algebraic function. In such case we can write the Boolean expression form the truth table, simplify it when possible, and then implement the simplified logic circuit.

A General design procedure:

1- The number of variables, input variables and variables in determined.
2- The input and output variables are assigned letters (symbols).
3- The truth table that defines the required relationships between input and output variables is derived.
4- The simplified Boolean functions for each output are obtained.
5- The logic diagram is drawn.

The following examples will make the design procedure of logic circuit clear and easy.

Example

Design logic circuit for the following expression.

1- $F = (\overline{A.B})(\overline{A.B})(\overline{\bar{A}.\bar{B}})$
2- $F = ABC\bar{D} + A\bar{B}C\bar{D} + \bar{A}BCD + \bar{A}BC\bar{D} + \bar{A}BCD$

Solution

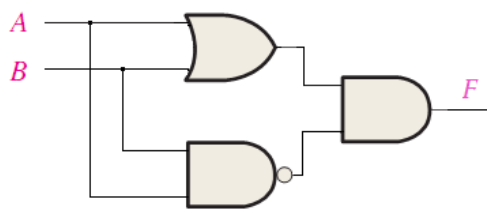
1-

$F = (\overline{A.B})(\overline{A.B})(\overline{\bar{A}.\bar{B}})$

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

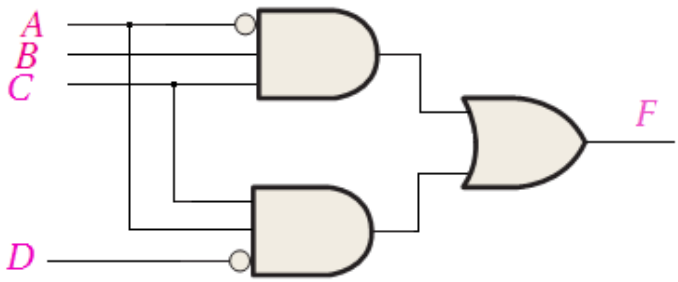
$F = (\overline{AB})(A + B)$

The logic cct is as shown beside



2- $F = ABC\bar{D} + A\bar{B}C\bar{D} + \bar{A}BCD + \bar{A}BC\bar{D} + \bar{A}BCD$

$$F = ABC\bar{D} + A\bar{B}C\bar{D} + \bar{A}BCD + \bar{A}BC\bar{D}$$
$$F = AC\bar{D}(B + \bar{B}) + \bar{A}BC(D + \bar{D})$$
$$F = AC\bar{D} + \bar{A}BC$$

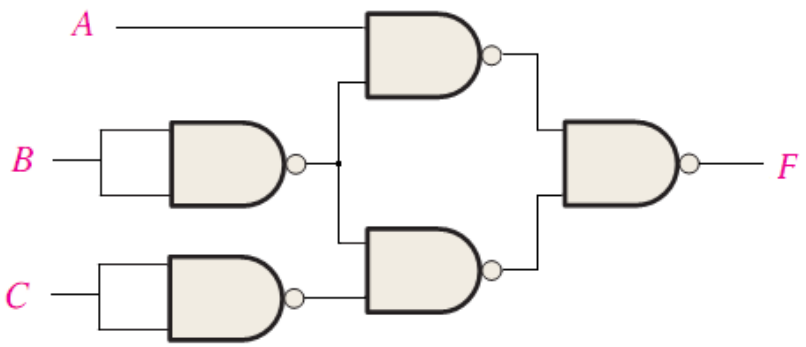


Example
Design the logic circuit that can implement the truth table below using NAND gates only.

A	B	C	F
0	0	0	**1**
0	0	1	**0**
0	1	0	**0**
0	1	1	**0**
1	0	0	**1**
1	0	1	**1**
1	1	0	**0**
1	1	1	**0**

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	1	0	0	1
C	0	0	0	1

Solution
$F = A\bar{B} + \bar{B}\bar{C}$
By using NAND
$F = \overline{\overline{A\bar{B} + \bar{B}\bar{C}}}$
$F = \overline{(\overline{A\bar{B}}).(\overline{\bar{B}\bar{C}})}$



Example

Design the logic circuit that can implement the truth table below using POS and SOP forms.

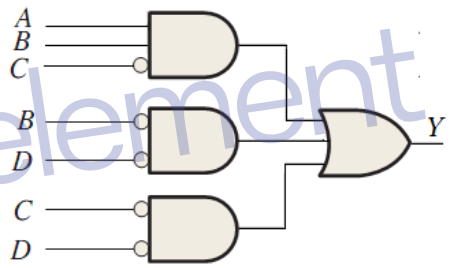
Inputs				Output
A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	x
1	0	0	0	1
1	0	0	1	0
1	0	1	0	x
1	0	1	1	0
1	1	0	0	x
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

Solution

1-SOP forms

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	1	1	x	1
$\bar{C}D$	0	0	1	0
CD	0	x	0	0
$C\bar{D}$	1	0	0	x

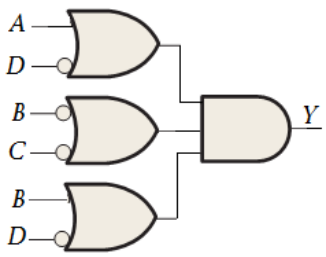
$$Y = \bar{C}\bar{D} + \bar{B}\bar{D} + AB\bar{C}$$



2-POS forms

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	1	1	x	1
$\bar{C}D$	0	0	1	0
CD	0	x	0	0
$C\bar{D}$	1	0	0	x

$$\bar{Y} = \bar{A}D + BC + \bar{B}D$$
$$Y = \bar{\bar{Y}} = \overline{\bar{A}D + BC + \bar{B}D}$$
$$Y = (\overline{\bar{A}D})(\overline{BC})(\overline{\bar{B}D})$$
$$Y = (A + \bar{D})(\bar{B} + \bar{C})(B + \bar{D})$$



Example

Design the logic circuit that can implement the truth table below using POS and SOP forms.

Inputs				Output
A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	x
1	0	1	1	x
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x

Solution

SOP and POS forms

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	1	1	x	1
$\bar{C}D$	0	0	x	0
CD	0	0	x	x
$C\bar{D}$	1	1	0	x

1- SOP $\rightarrow Y = \bar{D}$

2- POS$\rightarrow \bar{Y} = D$

$\therefore Y = \bar{D}$

Example

Design the logic circuit that can implement the truth table below using POS and SOP forms.

Inputs			Output
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	x
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	x
1	1	1	x

Solution

1-SOP forms

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	1	x	x	1
C	0	1	x	0

$Y = B + \bar{C}$

2-POS forms

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	1	x	x	1
C	0	1	x	0

$\bar{Y} = \bar{B}C$

$Y = \bar{\bar{Y}} = \overline{\bar{B}C} = B + \bar{C}$

4-Function of Combinational Logic

4.1Adders and Subtractor:

In this section we'll consider the following

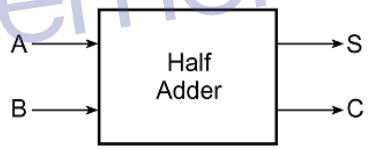
1- The Half-Adder (HA).
2- The Full-Adder (FA).
3- The Half-Subtractor (HS).
4- The Full- Subtractor (FS).

4.1.1 The Half-Adder (HA)

It can add two binary digits (bits) at time. As we know, binary addition of two bits always produces a 2-bit output data, i.e. one for the SUM and one for the CARRY. For example, (1+1) gives a sum 0 and a carry of 1. Also, (0+0) gives a sum 0 and a carry of 1. That is why the adder has two outputs: one for the SUM and the other for the CARRY.

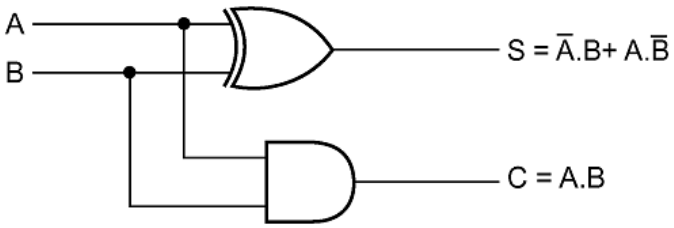
Table 7
Half-Adder truth table.

inputs		outputs	
A	*B*	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Where:
S = sum
C= output carry
A and B = input variables (operands)

The sum S output has the same logic pattern as when A XORed with B. Also the C carry output has the same logic pattern as when A is ANDed with B as shown below.

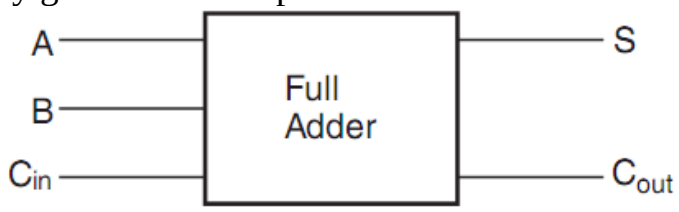


Logic circuit of a half- adder

The circuit is called a half-adder because it cannot accept a carry in from previous additions. For this purpose, we need a 3-input adder called the full-adder.

4.1.2 The Full-Adder (FA).

As shown in the block diagram below, it has three inputs and two outputs. It can add three bits at a time. The bits A and B are to be added and the third input C_{in} comes from the carry generated from pervious addition.



One of the outputs is a sum $\sum$ and the other is a carry-outC_{out}. The truth table gives all possible input / output relationships for the full-adder.

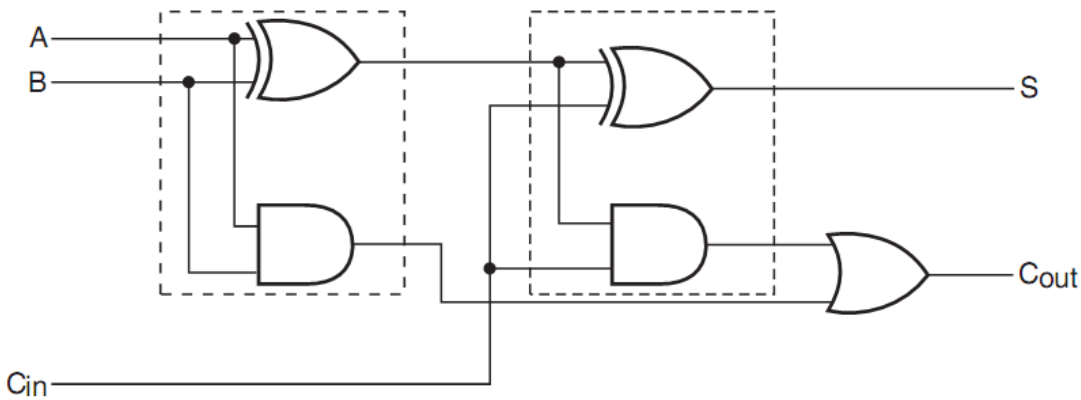
$$S = \sum = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC$$

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = \bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in}$$
$$C_{out} = (\bar{A}B + A\bar{B})C_{in} + AB(\bar{C}_{in} + C_{in})$$
$$C_{out} = (A \oplus B)C_{in} + AB$$

Table 8

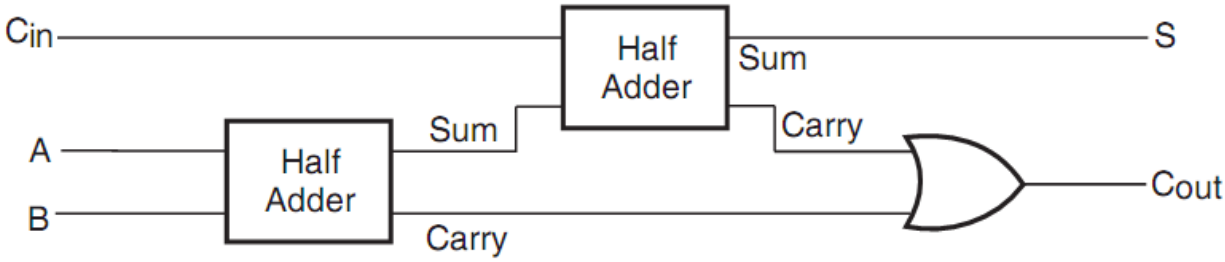
Full-Adder truth table.

inputs			outputs	
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



Complete logic circuit for a full-adder

The full-adder can be constructed from two half adder and one OR gate as shown below.



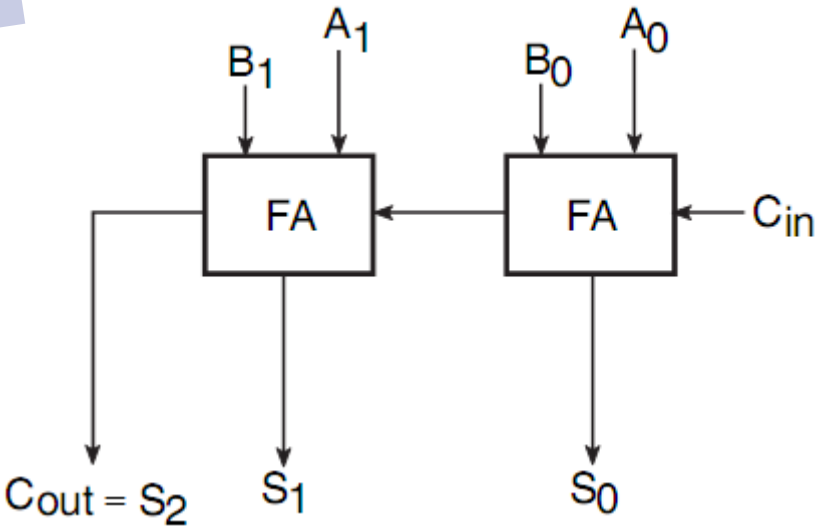
Arrangement of two half-adders to form a full-adder

- Note: these adders can also perform subtraction by the method of 2's complement.

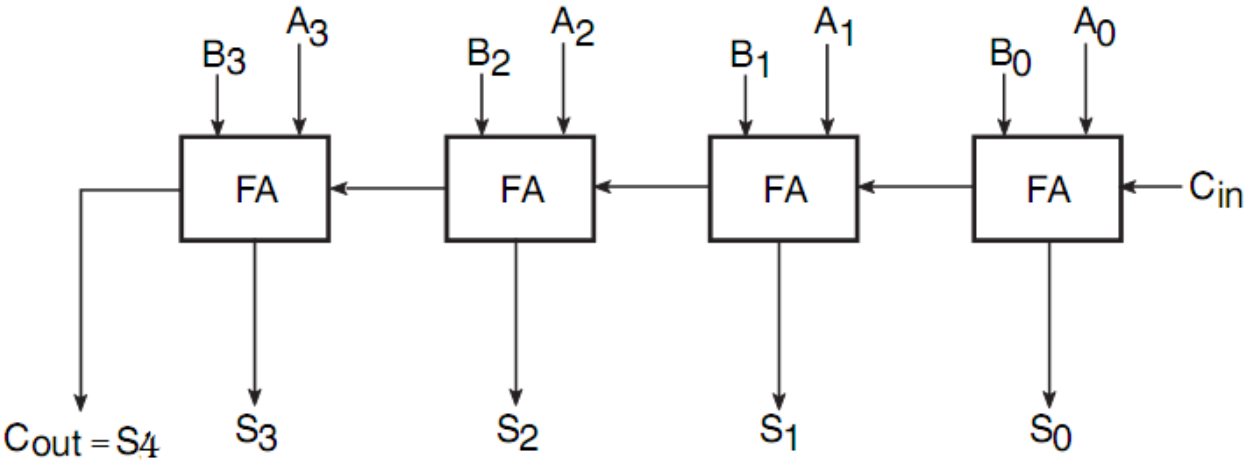
4.1.3Parallel Binary Adders

To add two binary numbers, a full-adder (FA) is required for each bit in the numbers. So for 2-bit numbers, two adders are needed; for 4-bit numbers, four adders are used; and so on. The carry output of each adder is connected to the carry input of the next higher-order adder, as shown in Figure below for a 2-bit adder.

- Notice that either a half-adder can be used for the least significant position or the carry input of a full-adder can be made 0 (grounded) because there is no carry input to the least significant bit position.



Block diagram of a basic 2-bit parallel adder using two full-adders.

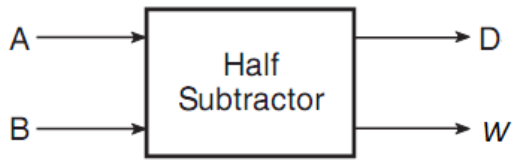


Block diagram for 4-bit parallel adder

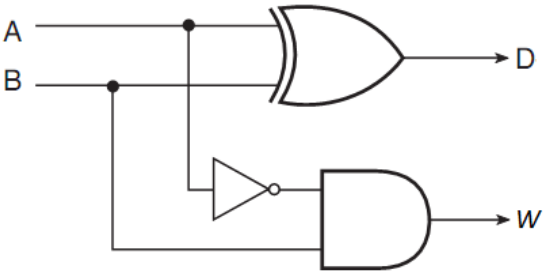
4.1.4 The Half-Subtractor (HS)

It can subtract two bits at time and produce an output of a difference and another for borrow.

Table 9			
Half- Subtractor truth table.			
inputs		outputs	
A	*B*	D	W
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



The operation of a half-Subtractor is based on the rules of binary subtractions. The difference (D) in the 3rd column has the same logic pattern as when A is XORed with B. The borrow output in the 4th column (W) can be obtaining by ANDing $\bar{A}$ with B. therefore; the logic circuit for the HS is as shown below:



$$D = A \oplus B$$
$$W = \bar{A}B$$

Logic circuit of a half- subtractor

4.1.5 The Full- Subtractor (FS).

A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend, and also takes into consideration whether a '1' has already been borrowed by the previous adjacent lower minuend bit or not. As a result, there are three bits to be handled at the input of a full subtractor, namely the two bits to be subtracted and a borrow bit designated as B_{in}. There are two outputs, namely the DIFFERENCE output D and the BORROW outputB_O. The BORROW output bit tells whether the minuend bit needs to borrow a '1' from the next possible higher minuend bit. The truth table of a full subtractor is as shown in the table 10.

Table 10				
Full- subtractor truth table.				
inputs			outputs	
Minuend A	Subtrahend B	Borrow In B_{in}	Difference D	Borrow Out B_o
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

The Boolean expressions for the two output variables are given by the equations

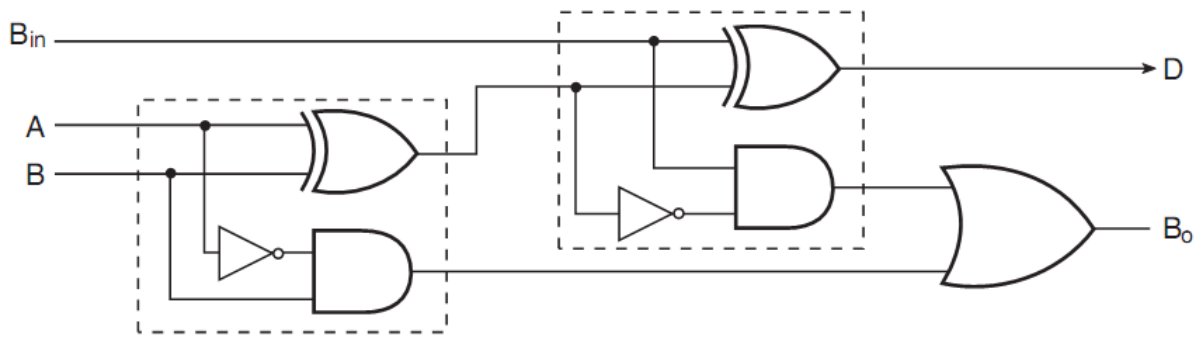
$$D = \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}\bar{B}_{in} + ABB_{in}$$

$$D = A \oplus B \oplus B_{in}$$

$$B_O = \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + \bar{A}BB_{in} + ABB_{in}$$

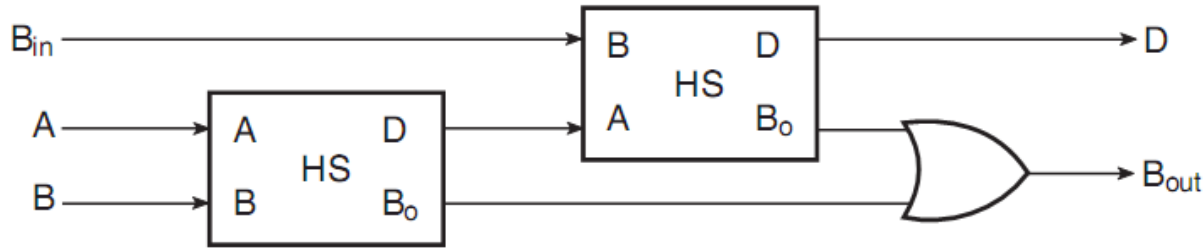
$$B_O = B_{in}(\bar{A}\bar{B} + AB) + \bar{A}B(\bar{B}_{in} + B_{in})$$

$$B_O = B_{in}\left(\overline{A \oplus B}\right) + \bar{A}B$$



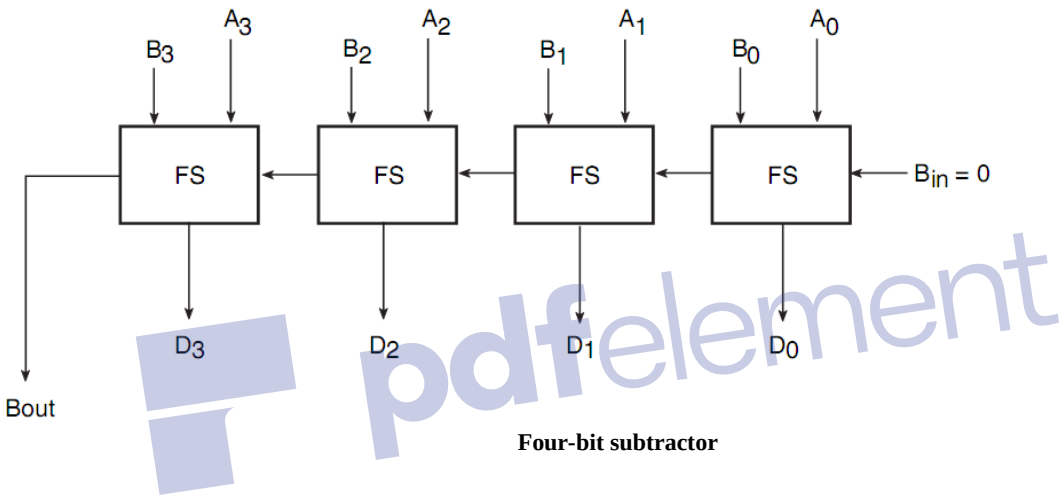
Complete logic circuit for a full-subtractor

As shown in figure below a full subtractor can be constructed from two HSs and an OR gate.



Implementation of a full subtractor with half-subtractors

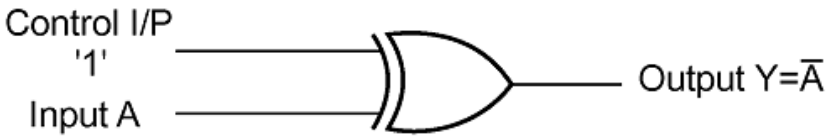
It may be remarked have that by cascading 4 full-subtractors, we can directly subtract 4-bit number, i.e. we can B_3 , B_2 , B_1 , B_0 fromA_3 , A_2 , A_1 , A_0.



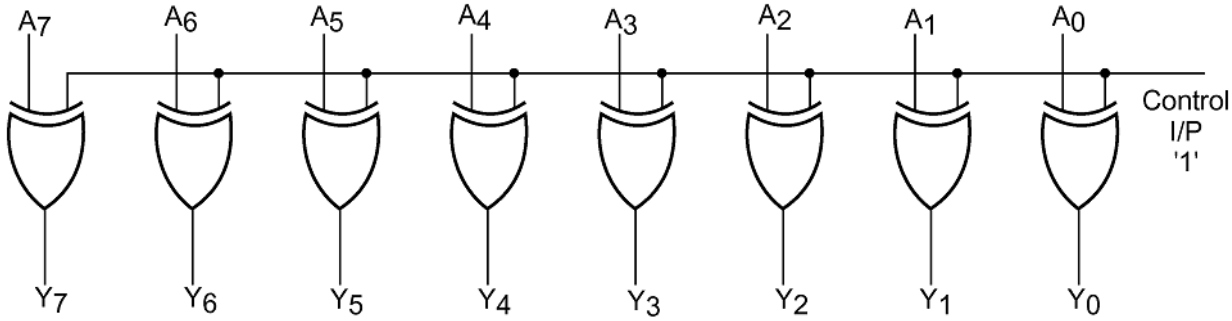
Four-bit subtractor

4.1.6 Controlled Inverter

A controlled inverter is needed when an adder is to be used as a subtractor. As outlined earlier, subtraction is nothing but addition of the 2's complement of the subtrahend to the minuend. Thus, the first step towards practical implementation of a subtractor is to determine the 2's complement of the subtrahend. And for this, one needs firstly to find 1's complement. A controlled inverter is used to find 1's complement. A one-bit controlled inverter is nothing but a two-input XOR gate with one of its inputs treated as a control input, as shown in Fig. below part (a). When the control input is LOW, the input bit is passed as such to the output. (Recall the truth table of an XOR gate.) When the control input is HIGH, the input bit gets complemented at the output. Figure part(b) shows an eight-bit controlled inverter of this type. When the control input is LOW, the output (Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0) is the same as the input (A7 A6 A5 A4 A3 A2 A1 A0). When the control input is HIGH, the output is 1's complement



(a)

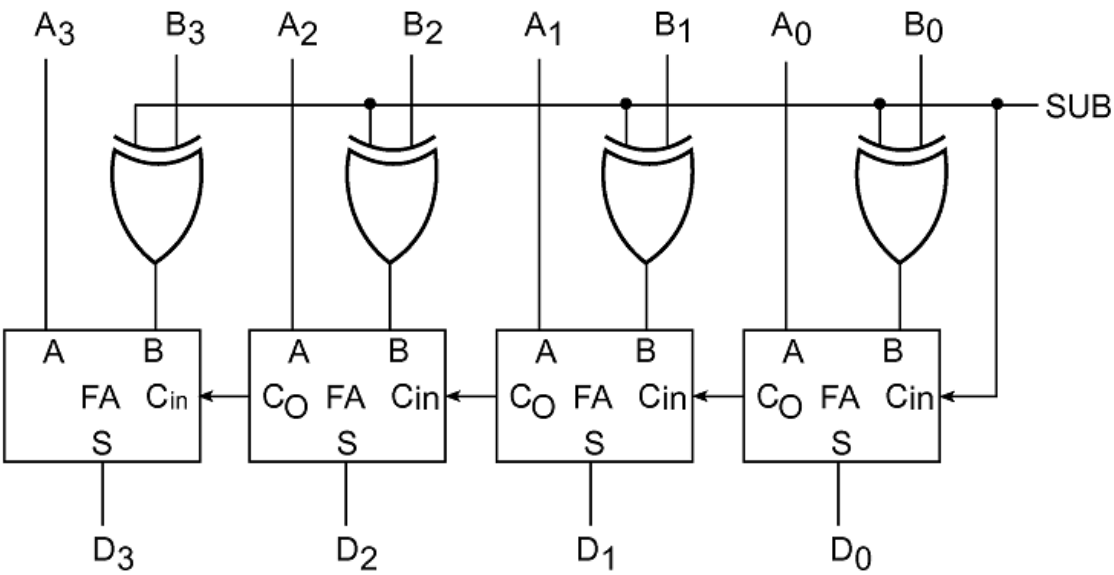


(b)

(a) One-bit controlled inverter and (b) eight-bit controlled inverter

Adder–Subtractor

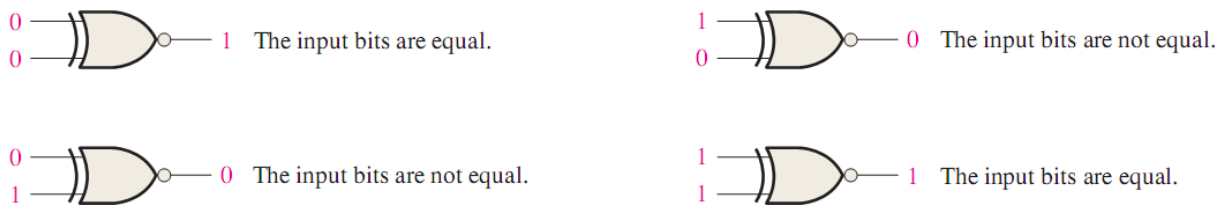
Subtraction of two binary numbers can be accomplished by adding 2's complement of the subtrahend to the minuend and disregarding the final carry, if any. If the MSB bit in the result of addition is a '0', then the result of addition is the correct answer. If the MSB bit is a '1', this implies that the answer has a negative sign. The true magnitude in this case is given by 2's complement of the result of addition.



Four-bit adder-subtractor

4.2 Magnitude Comparators

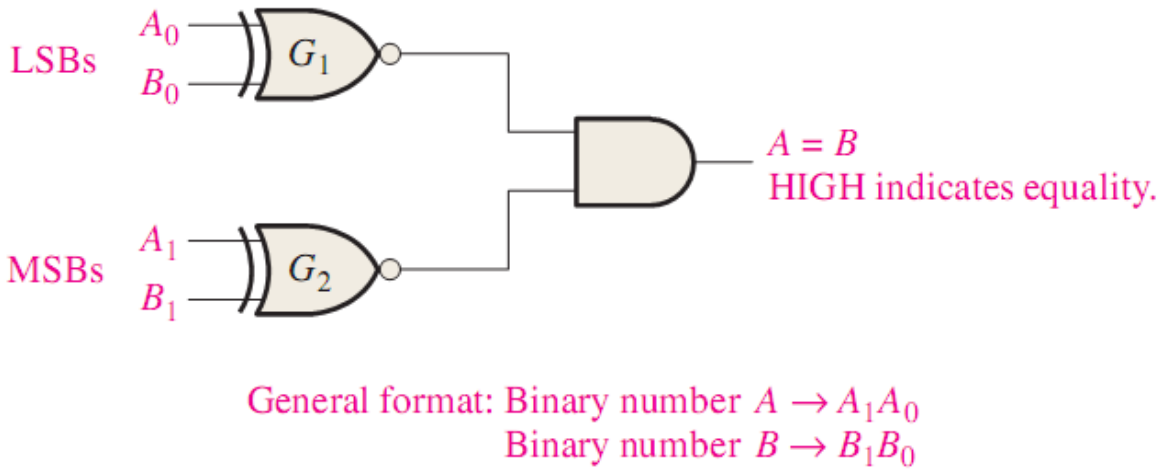
The basic function of a comparator is to compare the magnitudes of two binary quantities to determine the relationship of those quantities. In its simplest form, a comparator circuit determines whether two numbers are equal. The XNOR gate can be used as a basic comparator because its output is a 0 if the two input bits are not equal and a 1 if the input bits are equal. Figure below shows the XNOR gate as a 2-bit comparator.



Basic comparator operation

In order to compare binary numbers containing two bits each, an additional XNOR gate is necessary. The two least significant bits (LSBs) of the two numbers are compared by gate G1, and the two most significant bits (MSBs) are compared by gate G2, as shown in Figure below. If the two numbers are equal, their corresponding bits are the same, and the output of each XNOR gate is a 1. If the corresponding sets of bits are not equal, a 0 occurs on that XNOR gate output.

In order to produce a single output indicating an equality or inequality of two numbers, an AND gate can be combined with XNOR gates. The output of each XNOR gate is applied to the AND gate input. When the two input bits for each XNOR are equal, the corresponding bits of the numbers are equal, producing a 1 on both inputs to the AND gate and thus a 1 on the output. When the two numbers are not equal, one or both sets of corresponding bits are unequal, and a 0 appears on at least one input to the AND gate to produce a 0 on its output. Thus, the output of the AND gate indicates equality (1) or inequality (0) of the two numbers.



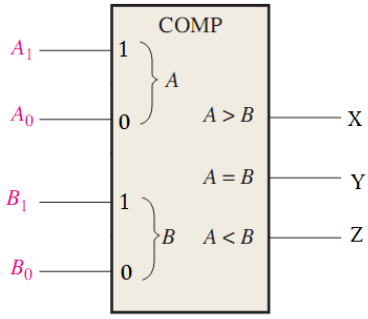
Logic diagram for equality comparison of two 2-bit numbers

Example

Design a complete magnitude comparator that compares two 2-bit binary numbers.

Solution

$A > B$ Then X = 1
$A = B$ Then Y = 1
$A < B$ Then Z = 1



For simplicity, let $A_1 = A$, $A_0 = B$, $B_1 = C$ and $B_0 = D$

Inputs				Output		
A	B	C	D			
A_0	A_1	B_1	B_0	X	Y	Z
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

For the output X

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	0	1	1	1
$\bar{C}D$	0	0	1	1
CD	0	0	0	0
$C\bar{D}$	0	0	1	0

$X = A\bar{C} + B\bar{C}\bar{D} + AB\bar{D}$
$X = A_1\bar{B}_1 + A_0\bar{B}_1\bar{B}_0 + A_1A_0\bar{B}_0$

For the output Y

$Y = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}D + ABCD + A\bar{B}C\bar{D}$
$Y = \bar{A}\bar{C}(\bar{B}\bar{D} + BD) + AC(\bar{B}\bar{D} + BD)$
$Y = (\bar{B}\bar{D} + BD).(\bar{A}\bar{C} + AC)$
$Y = (\overline{B \oplus D}).(\overline{A \oplus C})$
$Y = (\overline{A_0 \oplus B_0}).(\overline{A_1 \oplus B_1})$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	1	0	0	0
$\bar{C}D$	0	1	0	0
CD	0	0	1	0
$C\bar{D}$	0	0	0	1

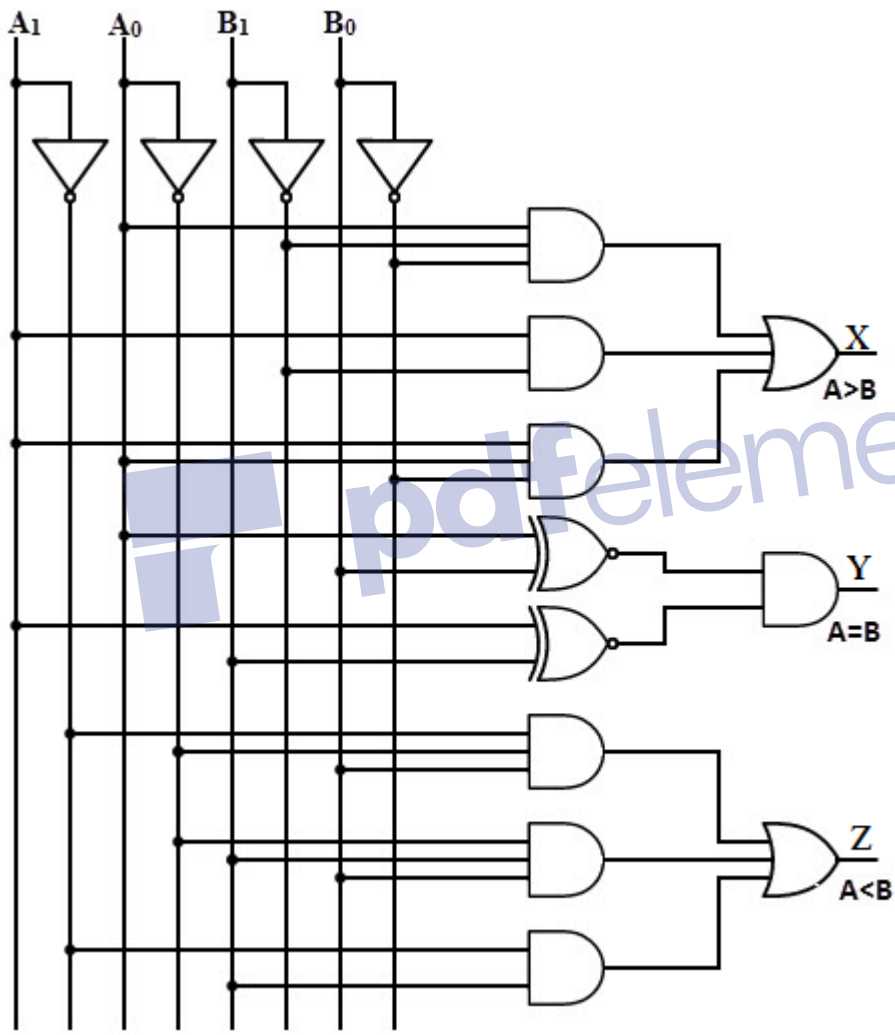
For the output Z

$Z = \bar{A}C + \bar{B}CD + \bar{A}\bar{B}D$

$Z = \bar{A}_1B_1 + \bar{A}_0B_1B_0 + \bar{A}_1\bar{A}_0B_0$

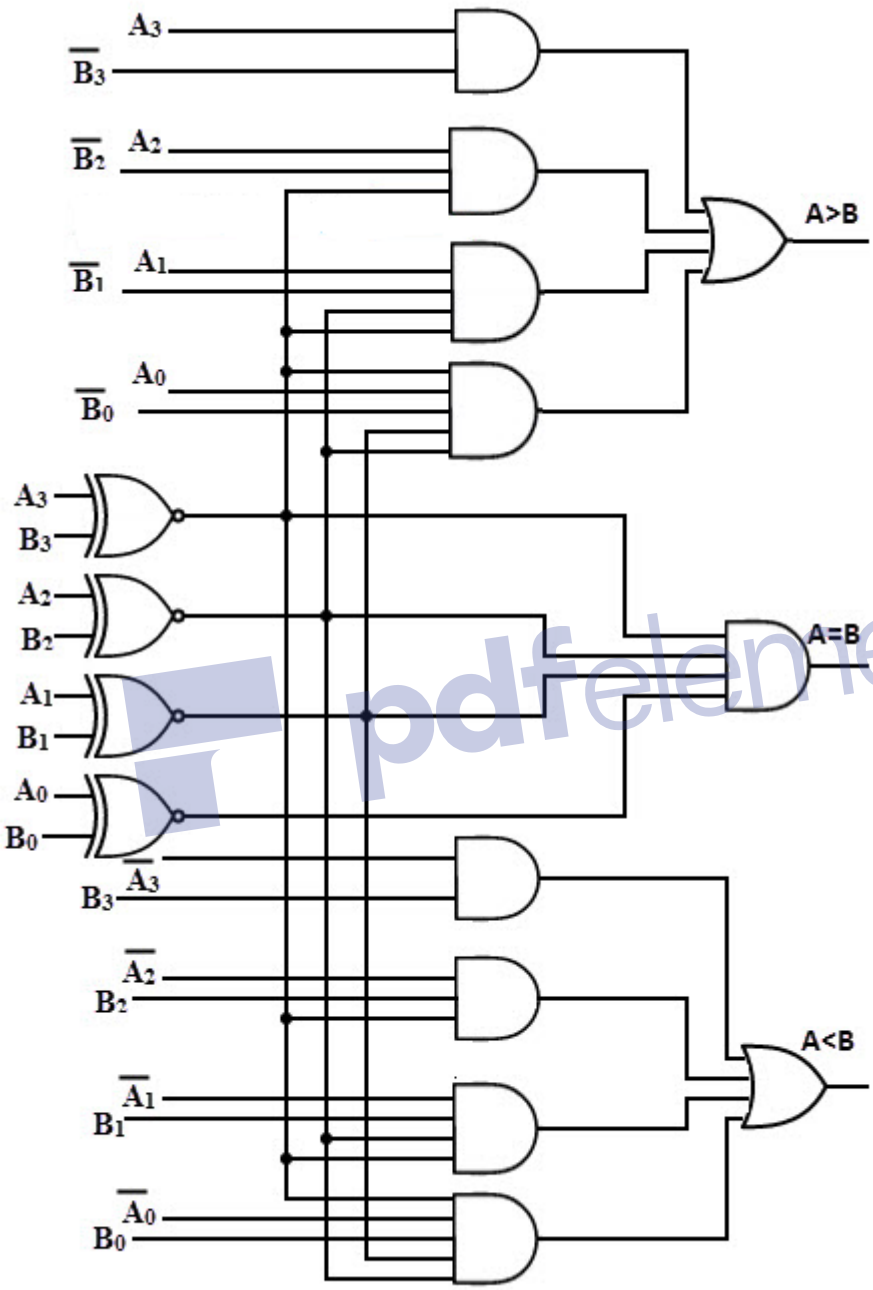
The complete circuit as shown below

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	0	0	0	0
$\bar{C}D$	1	0	0	0
CD	1	1	0	1
$C\bar{D}$	1	1	0	0



General 4-bit comparator:

The logic circuit for a complete 4-bit comparator that indicates whether $A > B$, $A = B$ or $A < B$ is given below:

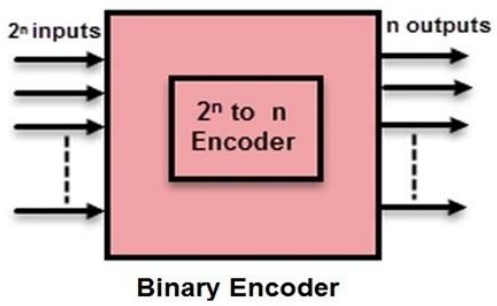


A practical magnitude comparator IC is the 7485 4-bit comparator

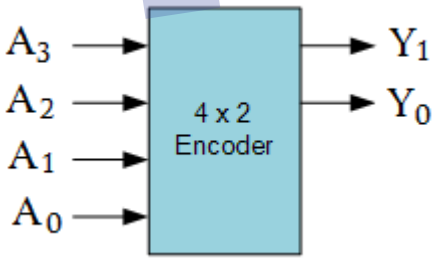
4.3 Encoders and Decoders

4.3.1 Encoders:

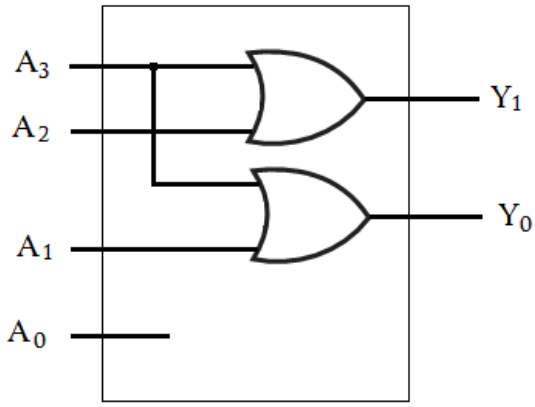
An encoder is a combinational logic circuit that essentially assigns a binary code of n-bits to an active input out of 2^n input lines. The inputs may represent octal or decimal digits and/or alphabetic characters. Therefore, this of process of converting from familiar symbols or numbers to a coded format is called encoding. The simplest encoder is a $2^n - to - n$ binary encoder, where it has only one of 2^n inputs = 1 and the output is the n-bit binary number corresponding to the active input. It can be built from OR gates



4-to-2 Bit Binary Encoder



Inputs				Outputs	
A_0	A_1	A_2	A_3	Y_1	Y_0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1



$Y_1 = A_3 + A_2$

$Y_0 = A_1 + A_3$

Octal-to-Binary Encoder

Octal-to-Binary take 8 inputs and provides 3 outputs, thus doing the opposite of what the 3-to-8 decoder does. At any one time, only one input line has a value of 1. The figure below shows the truth table of an Octal-to-binary encoder.

Inputs								Outputs		
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	Y_2	Y_1	Y_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

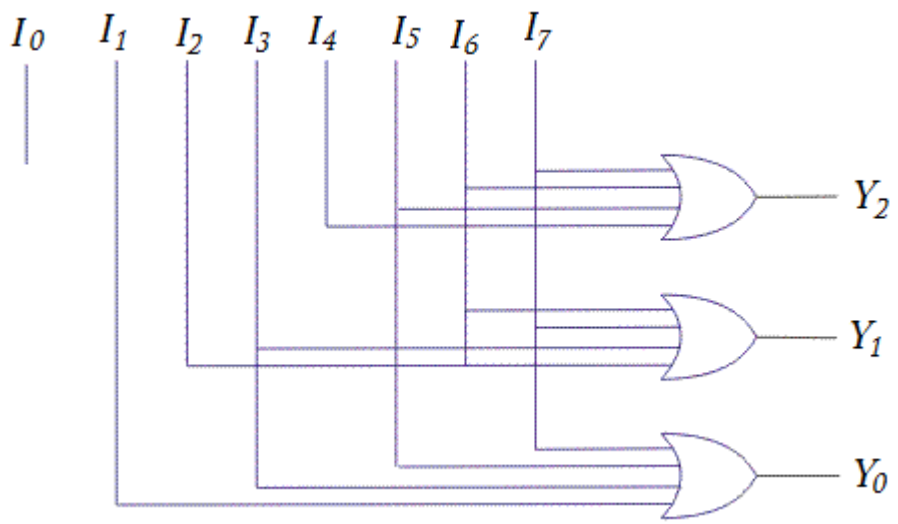
For an 8-to-3 binary encoder with inputs $I_0 - I_7$ the logic expressions of the outputs $Y_0 - Y_2$ are:

$Y_2 = I_7 + I_6 + I_5 + I_4$

$Y_1 = I_7 + I_6 + I_3 + I_2$

$Y_0 = I_7 + I_5 + I_3 + I_1$

Based on the above equations, we can draw the circuit as shown below



Priority Encoder

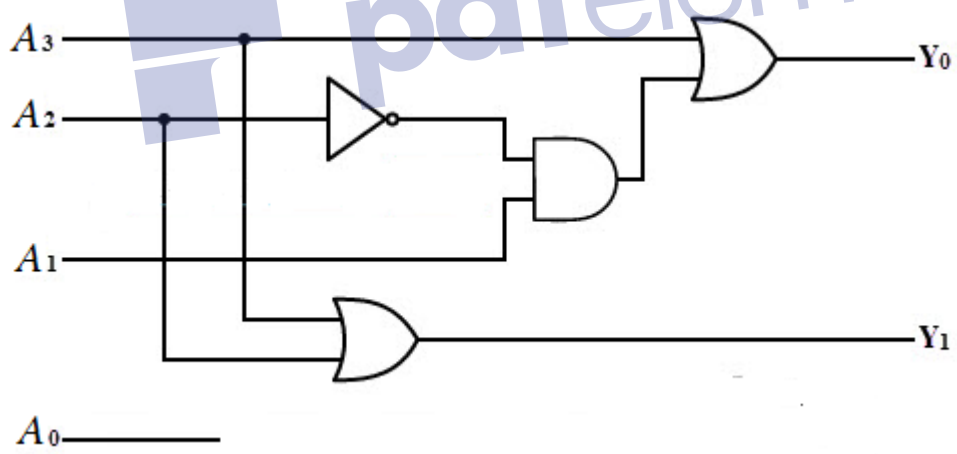
This is a special type of encoder. Priority is given to the input lines. If two or more input line are 1 at the same time, then the input line with highest priority will be considered. There are four inputsA_0 , A_1 , A_2 , A_3 and two outputsY_0, Y_1. Out of the four input A_3 has the highest priority and A_0 has the lowest priority. That means if $A_3 = 1$ then $Y_1Y_0 = 11$ irrespective of the other inputs. Similarly if $A_3 = 0$ and $A_2 = 1$ then $Y_1Y_0 = 10$ irrespective of the other inputs.

4-to-2 Priority Encoder

In 4-to-2 Priority Encoder A_3has the highest priority and A_0has the lower priority

Inputs				Outputs	
A_0	A_1	A_2	A_3	Y_1	Y_0
1	0	0	0	0	0
x	1	0	0	0	1
x	x	1	0	1	0
x	x	x	1	1	1

$Y_1 = A_3 + A_2\bar{A}_3 = A_3 + A_2$
$Y_0 = A_1\bar{A}_2\bar{A}_3 + A_3 = A_1\bar{A}_2 + A_3$



8 – to – 3 Priority Encoder or Octal – to – Binary Priority Encoder

The truth table of an octal – to – binary priority encoder is shown below. This type of encoder has 8 inputs and three outputs that generate corresponding binary code. A priority is assigned to each input so that when two or more inputs are 1 at a time, the input with highest priority is represented in the output.

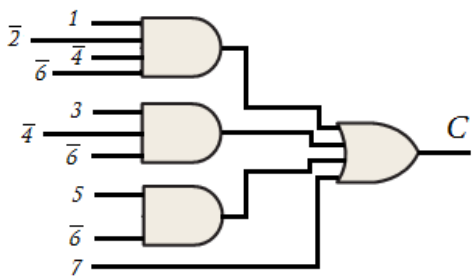
Suppose if the input lines I_2, I_4 and $I_7$are logic 1 simultaneously irrespective of the other inputs, only I_7 will be encoded and the output will be 111.Similarly, if $I_3 = 1$, the state of I_2, I_1 and I_0 is irrelevant or don't care and the output is equal to 011.

Inputs								Outputs		
0	1	2	3	4	5	6	7	*A*	*B*	*C*
1	0	0	0	0	0	0	0	0	0	0
x	1	0	0	0	0	0	0	0	0	1
x	x	1	0	0	0	0	0	0	1	0
x	x	x	1	0	0	0	0	0	1	1
x	x	x	x	1	0	0	0	1	0	0
x	x	x	x	x	1	0	0	1	0	1
x	x	x	x	x	x	1	0	1	1	0
x	x	x	x	x	x	x	1	1	1	1

C is 1 when $1,3,5, or\ 7\ is\ 1$ and if we consider the priorities, we must say that:

$$C \text{ is 1 when } \begin{cases} 1\ is\ 1\ and\ 2,4, and\ 6\ are\ 0 \\ 3\ is\ 1\ and\ 4\ and\ 6\ are\ 0 \\ 5\ is\ 1\ and\ 6\ is\ 0 \\ 7\ is\ 1 \end{cases}$$

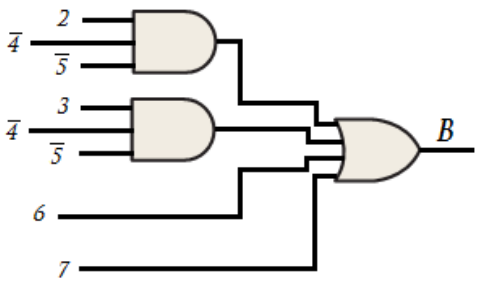
The logic circuit of the output C as shown below:



Similarly, $B = 2 + 3 + 6 + 7$
and

$$B \text{ is 1 when } \begin{cases} 2\ is\ 1\ and\ 4, and\ 5\ are\ 0 \\ 3\ is\ 1\ and\ 4\ and\ 5\ are\ 0 \\ 6\ is\ 1 \\ 7\ is\ 1 \end{cases}$$

And the logic circuit is:

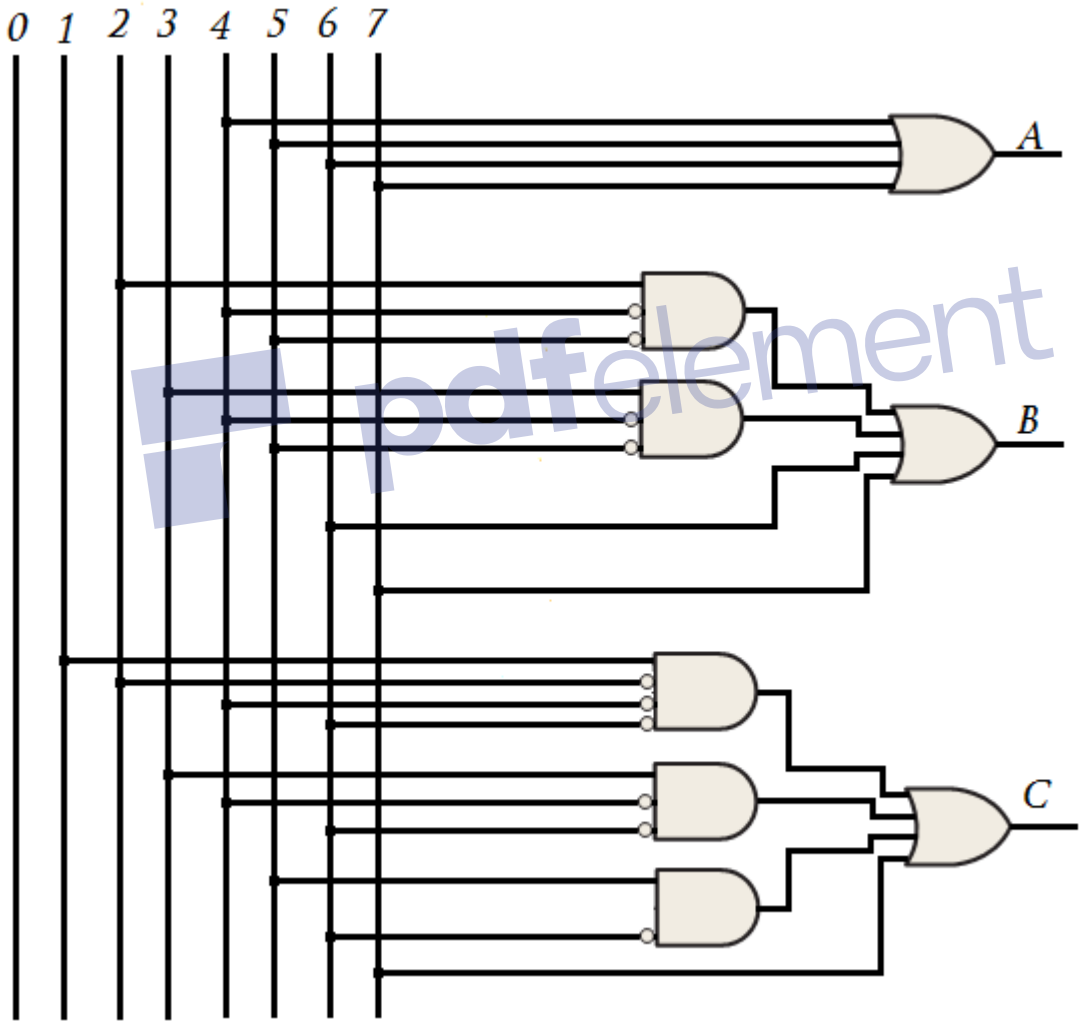


Finally, $A = 4 + 5 + 6 + 7$
and

$$A \text{ is 1 when } \begin{cases} 4 \ is \ 1 \\ 5 \ is \ 1 \\ 6 \ is \ 1 \\ 7 \ is \ 1 \end{cases}$$

And the logic circuit is:

Therefore, the complete logic diagram for the 8-3 priority encoder will be as shown below:

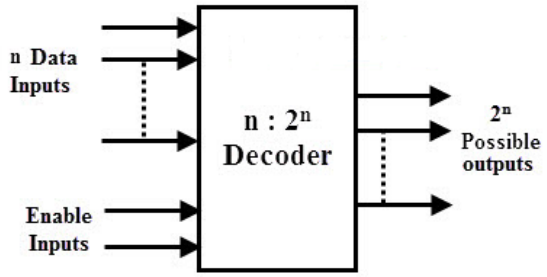


Note

1- The zero input is not connected because the output represents (000) when none of the other inputs are active.
2- The 74147 is a practical 16-4 priority encoder with active low signals.

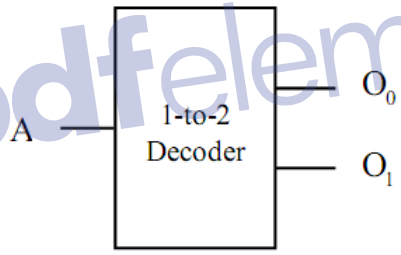
4.3.2 Decoder

A decoder performs the reverse operation of an encoder. That is, a code is returned to the corresponding symbol or digit. In the general form, a decoder has n input lines (to handle n bit) and 2^n output lines.. Only one output is active at any time while the other outputs are maintained at logic 0.



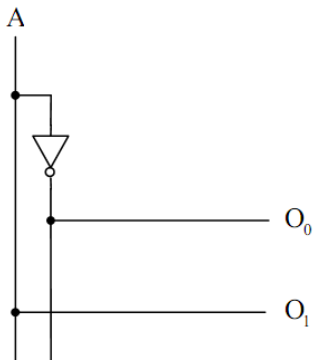
Example
Design 1-to2 decoder without enable
Solution

A	O_0	O_1
0	1	0
1	0	1

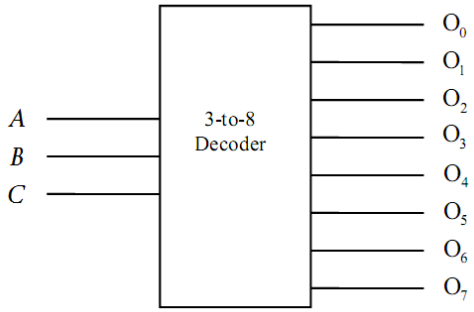


Now, let's write the logic function for each output interms of the inputs:
$O_0 = \bar{A}$
$O_1 = A$

Therefore, the logic circuit is



Example
Design 3-to8 decoder without enable.
Solution

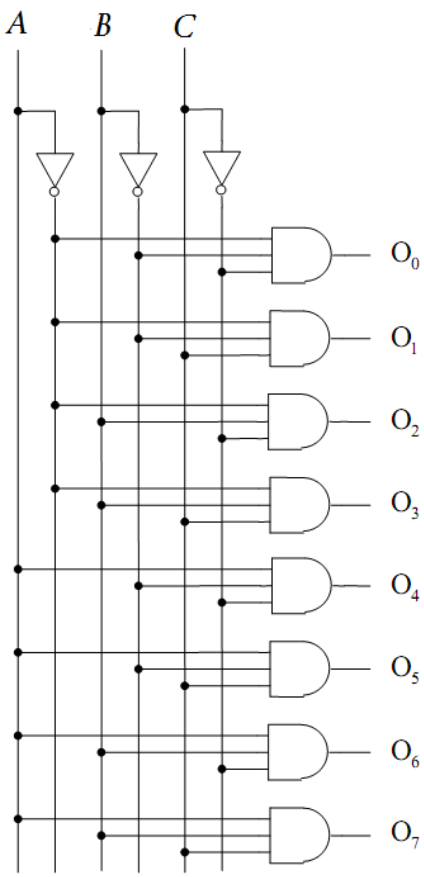


Inputs Lines			Outputs Lines							
A	B	C	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Now, let's write the logic function for each output interms of the inputs:

$O_0 = \bar{A}\bar{B}\bar{C}$
$O_1 = \bar{A}\bar{B}C$
$O_2 = \bar{A}B\bar{C}$
$O_3 = \bar{A}BC$
$O_4 = A\bar{B}\bar{C}$
$O_5 = A\bar{B}C$
$O_6 = AB\bar{C}$
$O_7 = ABC$

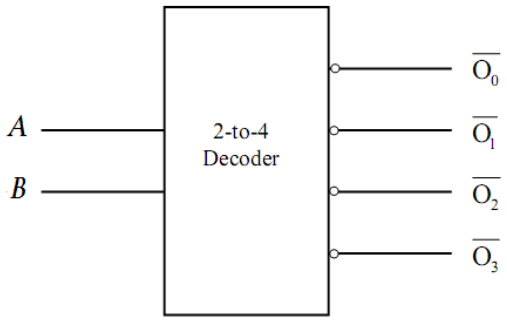
The logic circuit is as shown beside



Example

Design 2-to-4 decoder without enable and active low output

Solution



Inputs Lines		Outputs Lines			
A	B	$\bar{O}_0$	$\bar{O}_1$	$\bar{O}_2$	$\bar{O}_3$
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

Now, let's write the logic function for each output interms of the inputs:

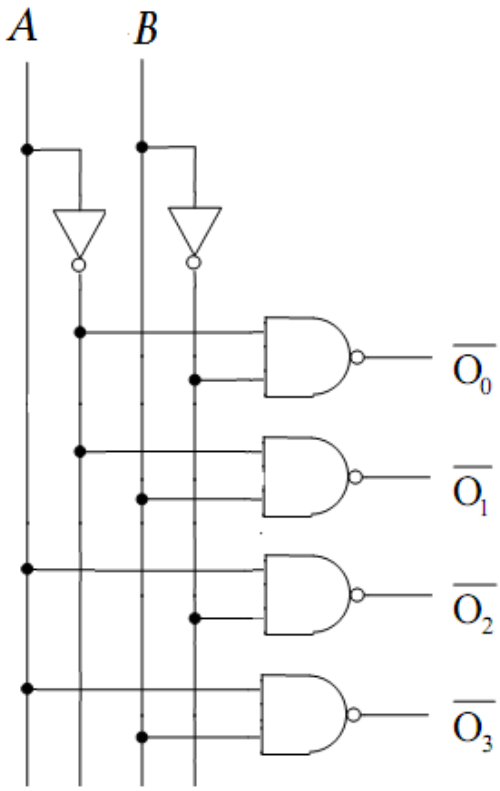
$\bar{O}_0 = \overline{\bar{A}\bar{B}}$

$\bar{O}_1 = \overline{\bar{A}B}$

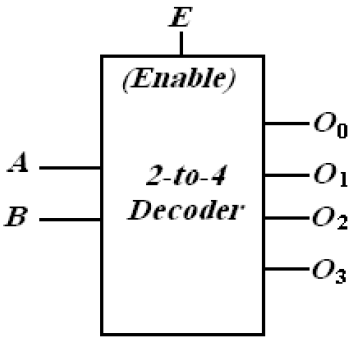
$\bar{O}_2 = \overline{A\bar{B}}$

$\bar{O}_3 = \overline{AB}$

The logic circuit is as shown below



Example
Design 2-to-4 decoder with enable and active high output
Solution

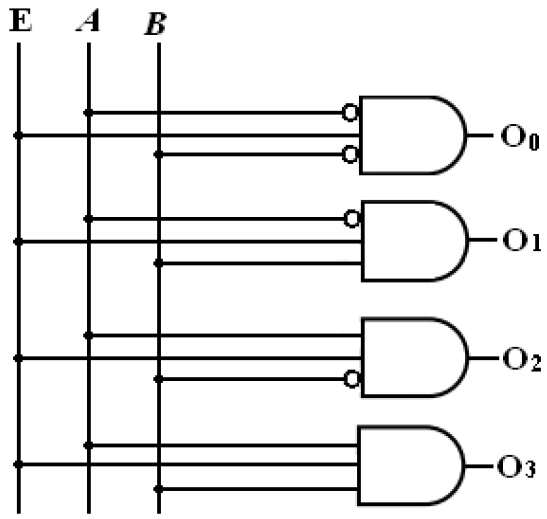


E	A	B	O_0	O_1	O_2	O_3
0	x	x	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

2-to-4 decoder truth table

Now, let's write the logic function for each output interms of the inputs:
$O_0 = E * \bar{A}\bar{B}$
$O_1 = E * \bar{A}B$
$O_2 = E * A\bar{B}$
$O_3 = E * AB$

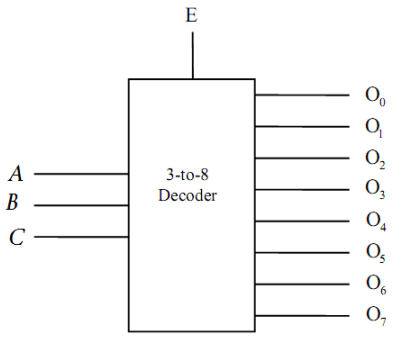
The logic circuit is



Example

Design 3-to-8 Decoder with Enable and active high output

Solution



Enable	Inputs Lines			Outputs Lines							
E	A	B	C	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

Now, let's write the logic function for each output interms of the inputs:

$O_0 = E\bar{A}\bar{B}\bar{C}$

$O_1 = E\bar{A}\bar{B}C$

$O_2 = E\bar{A}B\bar{C}$

$O_3 = E\bar{A}BC$

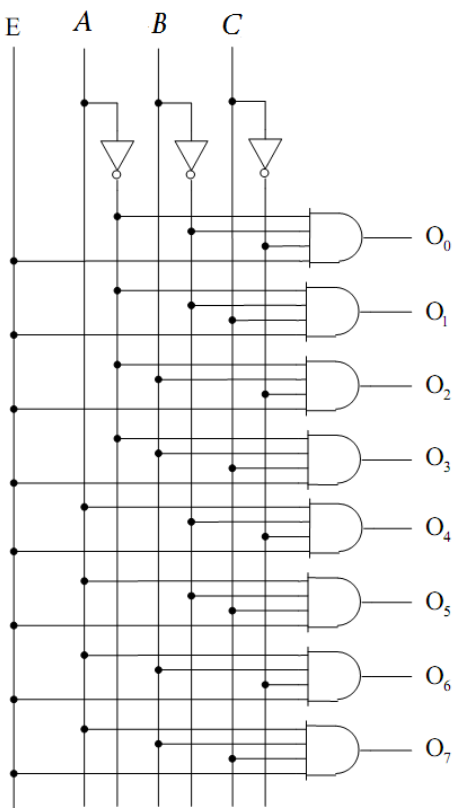
$O_4 = EA\bar{B}\bar{C}$

$O_5 = EA\bar{B}C$

$O_6 = EAB\bar{C}$

$O_7 = EABC$

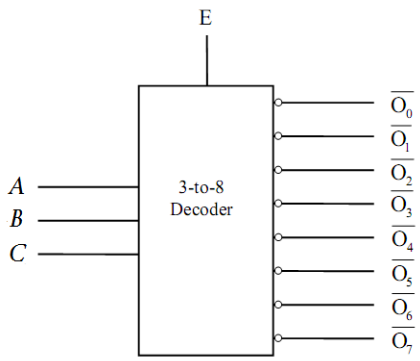
The logic circuit is as shown beside



Example

Design 3-to-8 Decoder with Enable and active low output

Solution



Enable	Inputs Lines			Outputs Lines							
E	A	B	C	$\bar{O}_0$	$\bar{O}_1$	$\bar{O}_2$	$\bar{O}_3$	$\bar{O}_4$	$\bar{O}_5$	$\bar{O}_6$	$\bar{O}_7$
0	x	x	x	1	1	1	1	1	1	1	1
1	0	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	0	1	1	1	1	1	1
1	0	1	0	1	1	0	1	1	1	1	1
1	0	1	1	1	1	1	0	1	1	1	1
1	1	0	0	1	1	1	1	0	1	1	1
1	1	0	1	1	1	1	1	1	0	1	1
1	1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	1	0

Now, let's write the logic function for each output interms of the inputs:

$\bar{O}_0 = \overline{E\bar{A}\bar{B}\bar{C}}$

$\bar{O}_1 = \overline{E\bar{A}\bar{B}C}$

$\bar{O}_2 = \overline{E\bar{A}B\bar{C}}$

$\bar{O}_3 = \overline{E\bar{A}BC}$

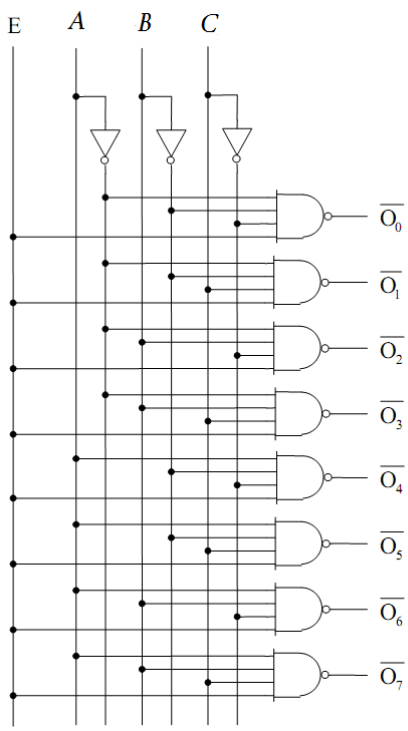
$\bar{O}_4 = \overline{EA\bar{B}\bar{C}}$

$\bar{O}_5 = \overline{EA\bar{B}C}$

$\bar{O}_6 = \overline{EAB\bar{C}}$

$\bar{O}_7 = \overline{EABC}$

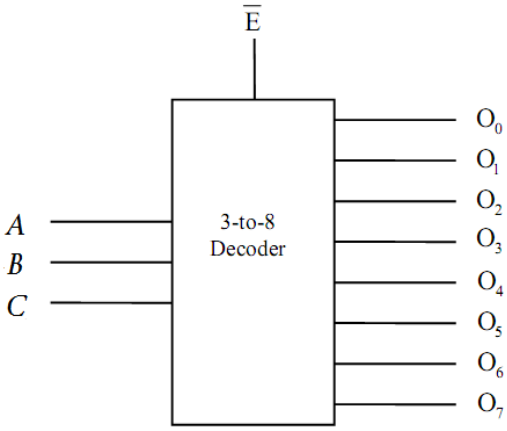
The logic circuit is as shown beside



Example

Design 3-to-8 Decoder with active low Enable and active high output

Solution



Enable	Inputs Lines			Outputs Lines							
E	A	B	C	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7
0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	0	0	0	1	0
0	1	1	1	0	0	0	0	0	0	0	1
1	x	x	x	0	0	0	0	0	0	0	0

Now, let's write the logic function for each output interms of the inputs:

$O_0 = \bar{E}\bar{A}\bar{B}\bar{C}$

$O_1 = \bar{E}\bar{A}\bar{B}C$

$O_2 = \bar{E}\bar{A}B\bar{C}$

$O_3 = \bar{E}\bar{A}BC$

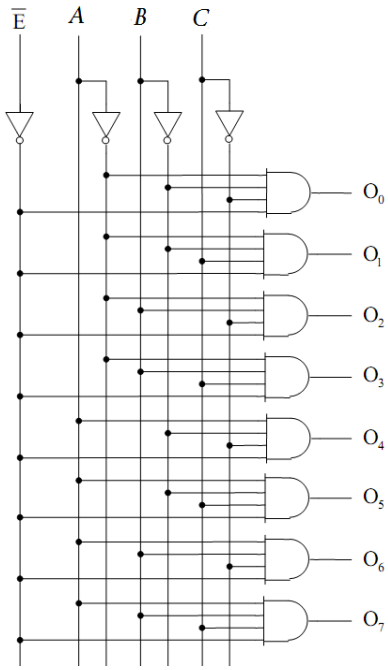
$O_4 = \bar{E}A\bar{B}\bar{C}$

$O_5 = \bar{E}A\bar{B}C$

$O_6 = \bar{E}AB\bar{C}$

$O_7 = \bar{E}ABC$

The logic circuit is as shown beside



- Note: The 74154 is a practical decoder [(4-16) decoder].

Decoder Expansion

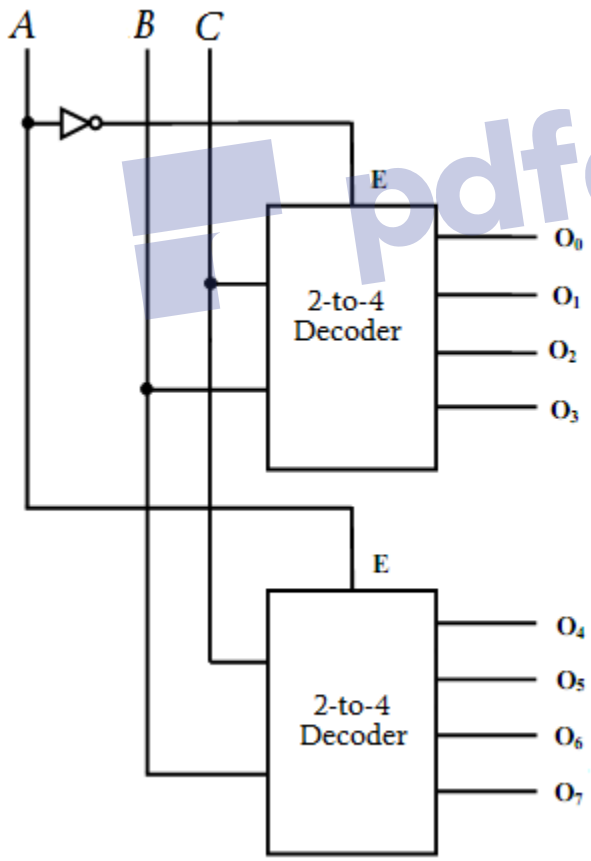
It is possible to combine or cascade two or more decoders to produce a decoder with larger number of input bits with the use of enable input of decoder.

Example

Construct a 3-to-8 decoder using only 2-to-4 decoders with additional gates.

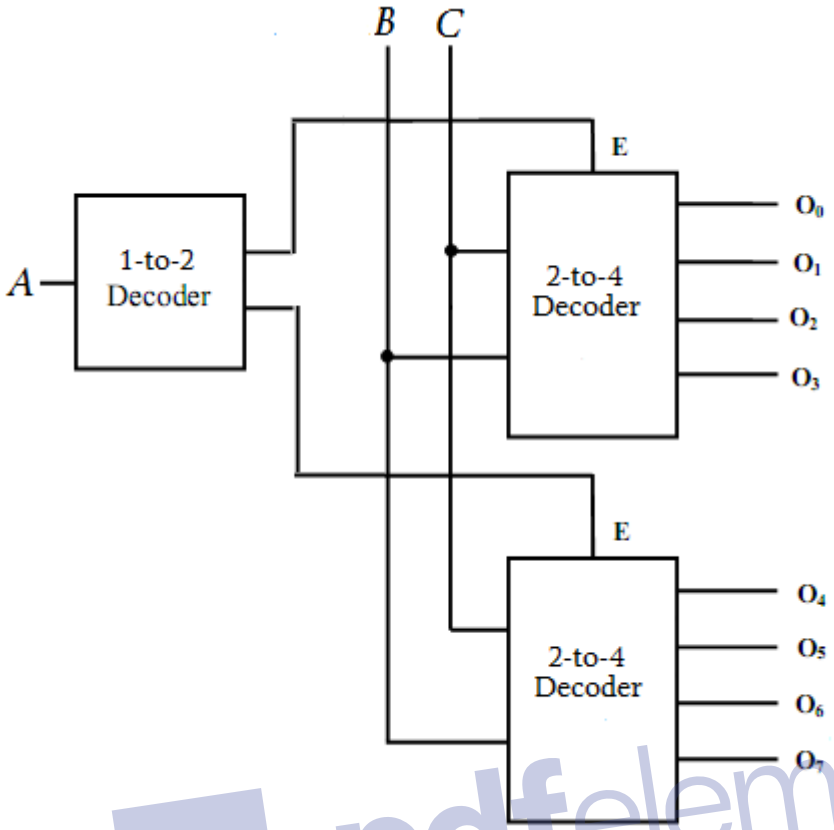
Solution

A	B	C	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



Example

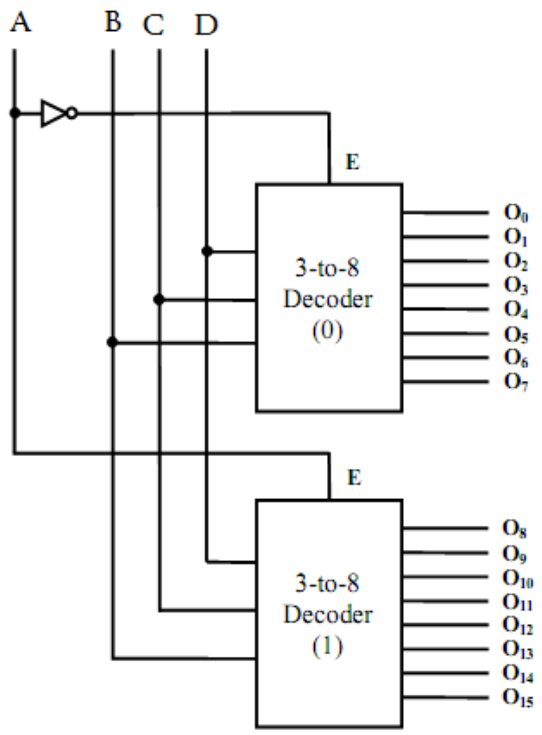
Construct a 3-to-8 decoder using 2-to-4 decoders with one 1-to-2 decoder



Example

Construct a 4-to-16 decoder using only 3-to-8 decoders with additional gates.

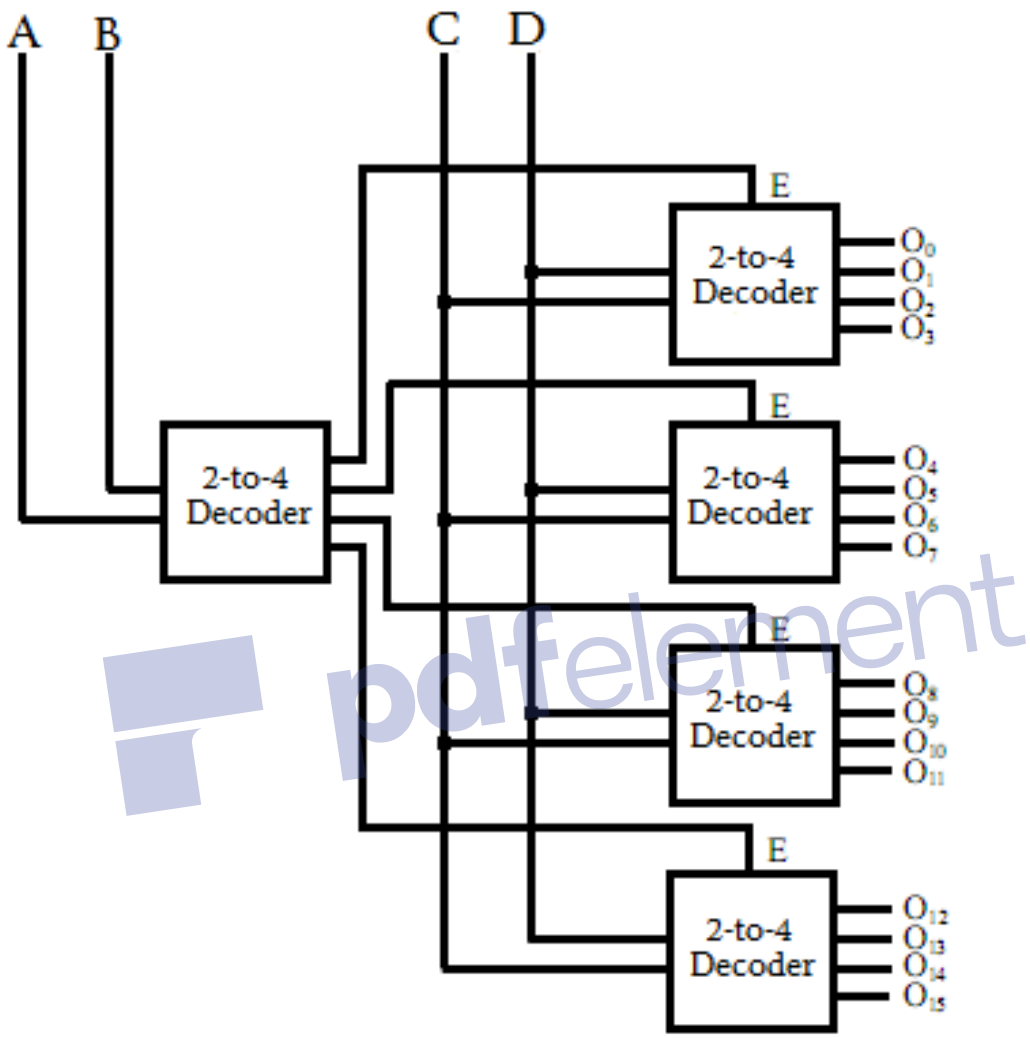
Solution



Example

Construct a 4-to-16 decoder using only 2-to-4 decoders.

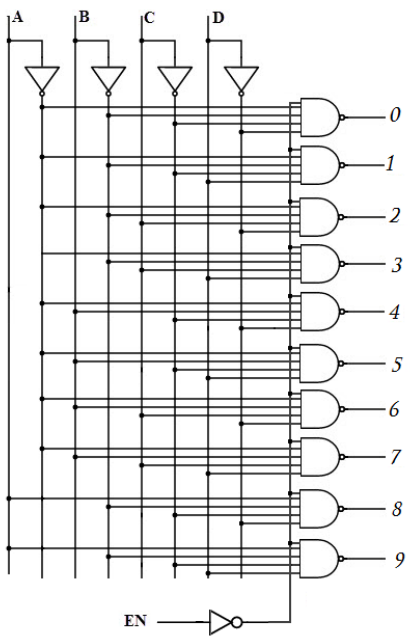
Solution



The BCD-to-Decimal Decoder

The BCD-to-decimal decoder converts each BCD code (8421 code) into one of ten possible decimal digit indications. It is frequently referred as a 4-line-to-10-line decoder or a 1-of-10 decoder. The method of implementation is the same as for the 4-of-16 decoder previously discussed, except that only ten decoding gates are required because the BCD code represents only the ten decimal digits 0 through 9. A list of the ten BCD codes and their corresponding decoding functions is given in below Table. Each of these decoding functions is implemented with NAND gates to provide active-LOW outputs. If an active-HIGH output is required, AND gates are used for decoding.

E	BCD Code				Decimal Output										Decoding Function
	A	B	C	D	0	1	2	3	4	5	6	7	8	9	
0	X	X	X	X	1	1	1	1	1	1	1	1	1	1	
1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	$\overline{E\bar{A}\bar{B}\bar{C}\bar{D}}$
1	0	0	0	1	1	0	1	1	1	1	1	1	1	1	$\overline{E\bar{A}\bar{B}\bar{C}D}$
1	0	0	1	0	1	1	0	1	1	1	1	1	1	1	$\overline{E\bar{A}\bar{B}C\bar{D}}$
1	0	0	1	1	1	1	1	0	1	1	1	1	1	1	$\overline{E\bar{A}\bar{B}CD}$
1	0	1	0	0	1	1	1	1	0	1	1	1	1	1	$\overline{E\bar{A}B\bar{C}\bar{D}}$
1	0	1	0	1	1	1	1	1	1	0	1	1	1	1	$\overline{E\bar{A}B\bar{C}D}$
1	0	1	1	0	1	1	1	1	1	1	0	1	1	1	$\overline{E\bar{A}BC\bar{D}}$
1	0	1	1	1	1	1	1	1	1	1	1	0	1	1	$\overline{E\bar{A}BCD}$
1	1	0	0	0	1	1	1	1	1	1	1	1	0	1	$\overline{EA\bar{B}\bar{C}\bar{D}}$
1	1	0	0	1	1	1	1	1	1	1	1	1	1	0	$\overline{EA\bar{B}\bar{C}D}$



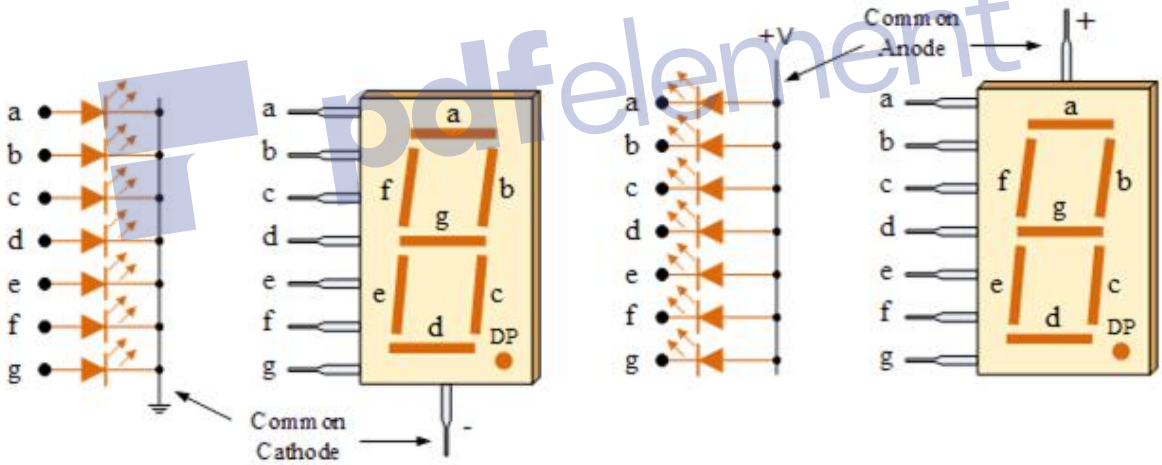
BCD to 7-Segment Display Decoders

7-segment LED (Light Emitting Diode) type display, provide a very convenient way of displaying information or digital data in the form of numbers, letters or even alpha-numerical characters.

A standard 7-segment LED display generally has 8 input connections, one for each LED segment and one that acts as a common terminal or connection for all the internal display segments. Some single displays have also have an additional input pin to display a decimal point in their lower right or left hand corner.

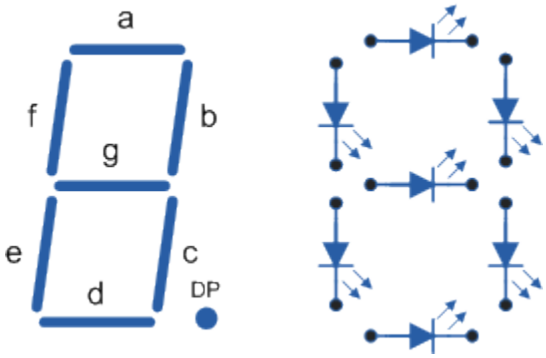
In electronics there are two important types of 7-segment LED digital display.

1. The Common Cathode Display (CCD) – In the common cathode display, all the cathode connections of the LED's are joined together to logic "0" or ground. The individual segments are illuminated by application of a "HIGH", logic "1" signal to the individual Anode terminals.
2. The Common Anode Display (CAD) – In the common anode display, all the anode connections of the LED's are joined together to logic "1" and the individual segments are illuminated by connecting the individual Cathode terminals to a "LOW", logic "0" signal.



Electrical connection of the individual diodes for a common cathode display and a common anode display and by illuminating each light emitting diode individually, they can be made to display a variety of numbers or characters.

7-Segment Display Format



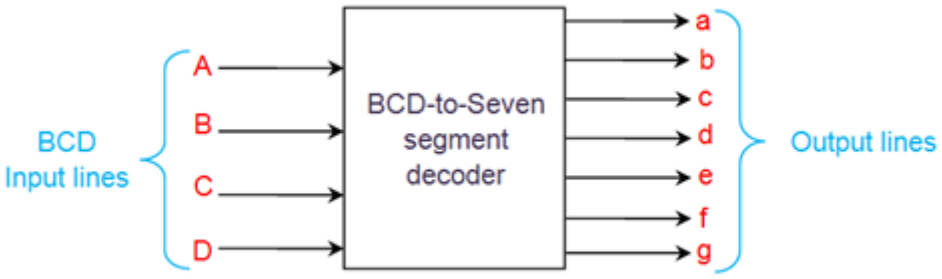
So in order to display the number 3 for example, segments a, b, c, d and g would need to be illuminated. If we wanted to display a different number or letter then a different set of segments would need to be illuminated. Then for a 7-segment display, we can produce a truth table giving the segments that need to be illuminated in order to produce the required character as shown below.

Table for a 7-segment display

Individual Segments							Display
A	b	c	d	e	f	g	
x	x	x	x	x	x		0
	x	x					1
x	x		x	x		x	2
x	x	x	x			x	3
	x	x			x	x	4
x		x	x		x	x	5
x		x	x	x	x	x	6
x	x	x					7
x	x	x	x	x	x	x	8
x	x	x	x		x	x	9

BCD to 7-Segment Decoder

BCD to seven segment decoder is a circuit used to convert the input BCD into a form suitable for the display. It has four input lines (A, B, C and D) and 7 output lines (a, b, c, d, e, f and g) as shown in Figure below.



BCD-to-Seven segment decoder

Truth table for BCD to seven segment decoder with common anode

Decimal digit	Input lines				Output lines							Display pattern
	A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	0	1	1	1	1	1	1	0	0
1	0	0	0	1	0	1	1	0	0	0	0	1
2	0	0	1	0	1	1	0	1	1	0	1	2
3	0	0	1	1	1	1	1	1	0	0	1	3
4	0	1	0	0	0	1	1	0	0	1	1	4
5	0	1	0	1	1	0	1	1	0	1	1	5
6	0	1	1	0	1	0	1	1	1	1	1	6
7	0	1	1	1	1	1	1	0	0	0	0	7
8	1	0	0	0	1	1	1	1	1	1	1	8
9	1	0	0	1	1	1	1	1	0	1	1	9

Segment	Logic Function
A	∑m(0,2,3,5,6,7,8,9)
B	∑m(0,1,2,3,4 ,7,8,9)
C	∑m(0,1,3,4,5,6,7,8,9)
D	∑m(0,2,3, 5,6,8,9)
E	∑m(0,2, 6,8)
F	∑m(0, 4,5,6, 8,9)
G	∑m(2,3,4,5,6, 8,9)

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	1	0	x	1
$\bar{C}D$	0	1	x	1
CD	1	1	x	x
$C\bar{D}$	1	1	x	x

$a = A + C + BD + \bar{B}\bar{D}$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	1	1	x	1
$\bar{C}D$	1	0	x	1
CD	1	1	x	x
$C\bar{D}$	1	0	x	x

$b = \bar{B} + CD + \bar{C}\bar{D}$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	1	1	x	1
$\bar{C}D$	1	1	x	1
CD	1	1	x	x
$C\bar{D}$	0	1	x	x

$c = \bar{C} + D + B$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	1	0	x	1
$\bar{C}D$	0	1	x	1
CD	1	0	x	x
$C\bar{D}$	1	1	x	x

$d = A + C\bar{D} + \bar{B}\bar{D} + \bar{B}C + B\bar{C}D$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	1	0	x	1
$\bar{C}D$	0	0	x	0
CD	0	0	x	x
$C\bar{D}$	1	1	x	x

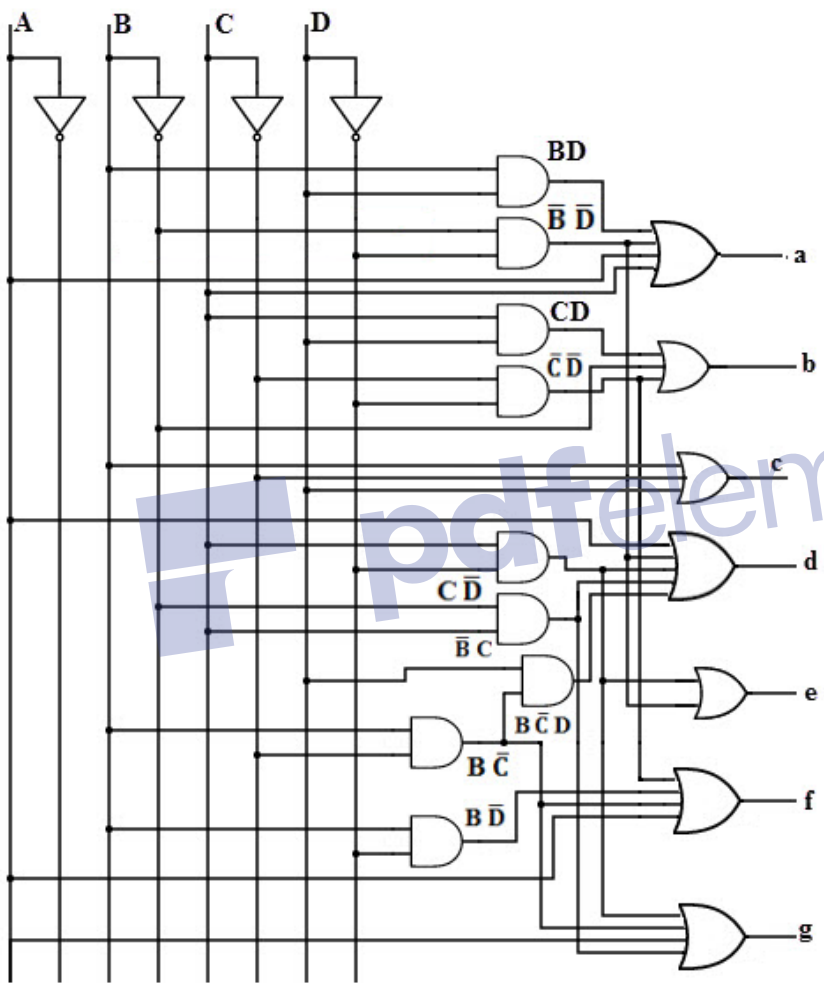
$e = C\bar{D} + \bar{B}\bar{D}$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	1	1	x	1
$\bar{C}D$	0	1	x	1
CD	0	0	x	x
$C\bar{D}$	0	1	x	x

$f = A + \bar{C}\bar{D} + \bar{B}\bar{D} + \bar{B}C + B\bar{C}D$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	0	1	x	1
$\bar{C}D$	0	1	x	1
CD	1	0	x	x
$C\bar{D}$	1	1	x	x

$$g = A + C\bar{D} + \bar{B}C + B\bar{C}$$



Note

1- When common-anode 7-segment display is used active-low outputs are required.
2- Other types of digital displays are:
   a- LCDs (Liquid Crystal Displays).
   b- The Dot Matrix.
3- A practical BCD-to-7segment decoder driver is the 7447

Decoder Application: Implementing Boolean Functions Using Decoders

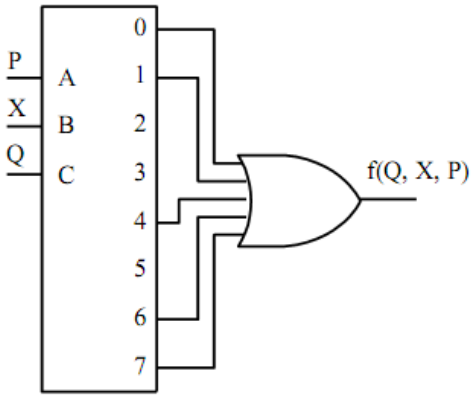
Any combinational circuit can be constructed using decoders and OR gates. The decoder generates the required minterms and an external OR gate is used to produce the sum of minterms.

Example

Implement the following Boolean Functions Using Decoders.

$F(Q,X,P) = \sum m(0,1,4,6,7)$

Solution



Example

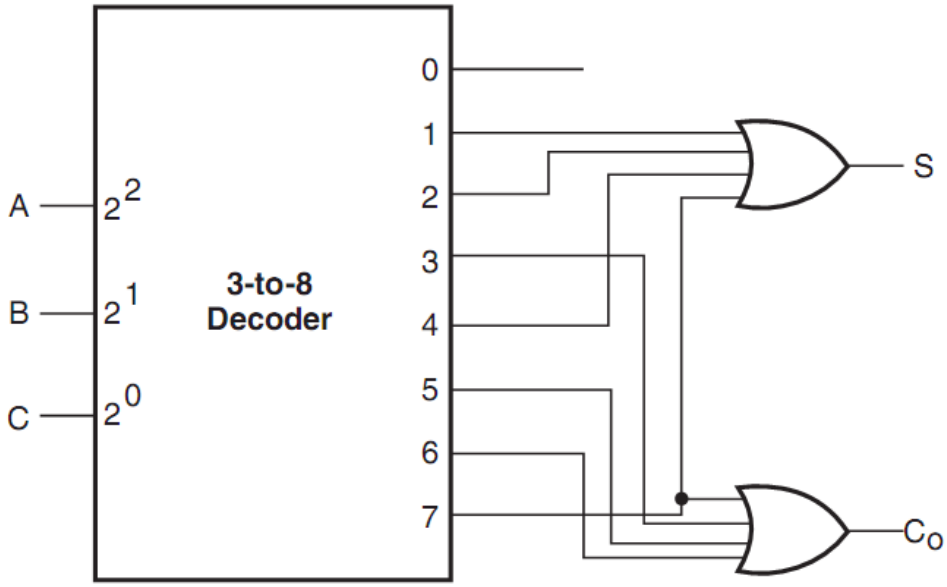
Implement a full adder circuit with a decoder and two OR gates.

Solution

Let

1- A, B, and C are inputs.
2- full adder equations are:
   - $S(A,B,C) = \sum m\ (1,2,4,7)$.
   - $C\ (A,B,C) = \sum m\ (3,5,6,7)$.

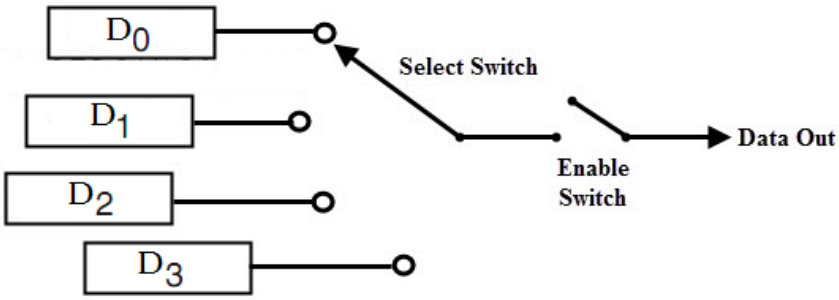
Since there are 3 inputs and a total of 8 minterms, we need a 3-to-8 decoder.



4.4Multiplexers (MUX) and Demultiplexer (DEMUX)

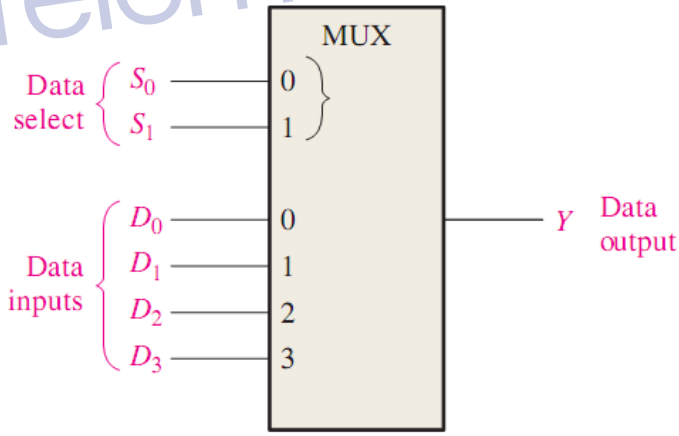
4.4.1Multiplexers (Data Selectors)

A multiplexer (MUX) is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination. Then, it has several data-input lines and a single output line. It also has data-select inputs that permit digital data on any one of the inputs to be switched to the output line. Multiplexers are also known as data selectors.



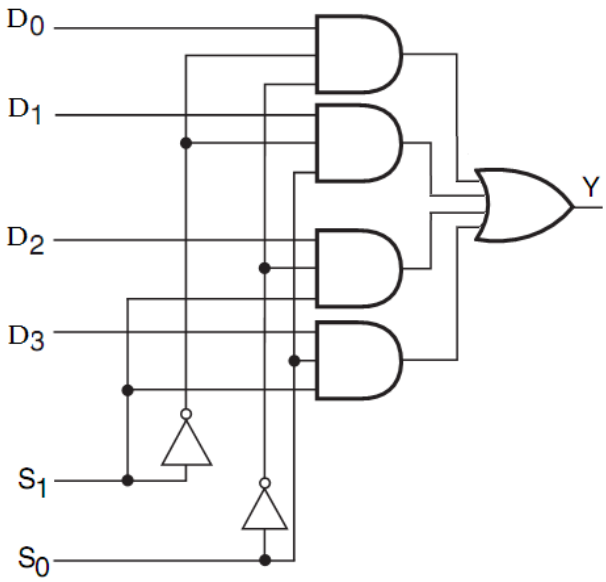
A logic symbol for a 4-input multiplexer (MUX without enable) is shown in Figure below. Notice that there are two data-select lines because with two select bits, any one of the four data-input lines can be selected.

Data Select		Output data Y
S_1	S_0	
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3



The logic expression for the output in terms of the input and the select inputs are:
The output is equal to D_0 only if $S_1 = 0$ *and* $S_0 = 0$: $Y = D_0\bar{S}_1\bar{S}_0$.
The output is equal to D_1 only if $S_1 = 0$ *and* $S_0 = 0$: $Y = D_1\bar{S}_1S_0$.
The output is equal to D_1 only if $S_1 = 0$ *and* $S_0 = 0$: $Y = D_2S_1\bar{S}_0$.
The output is equal to $D2$ only if $S_1 = 0$ *and* $S_0 = 0$: $Y = D_3S_1S_0$.
When these terms are ORed, the total expression for the data output is
$Y = D_0\bar{S}_1\bar{S}_0 + D_1\bar{S}_1S_0 + D_2S_1\bar{S}_0 + D_3S_1S_0$

The logic circuit for a 4-to-1 multiplexer without enable is:

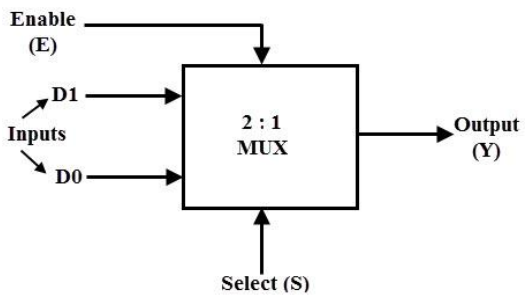


Example

Design 2-to-1 multiplexer with an enable input.

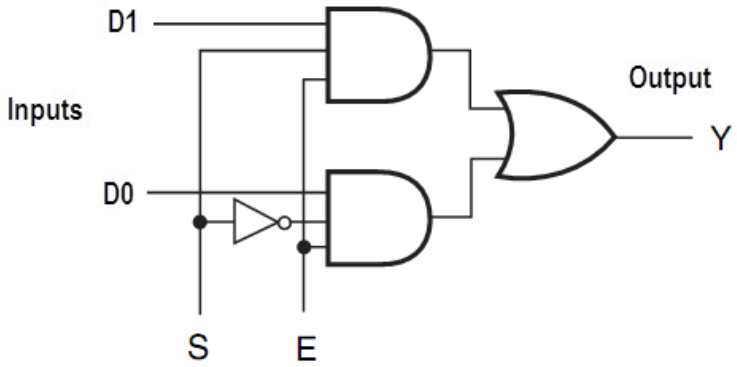
Solution

Enable	Data Select S	Output data Y
0	x	0
1	0	D_0
1	1	D_1

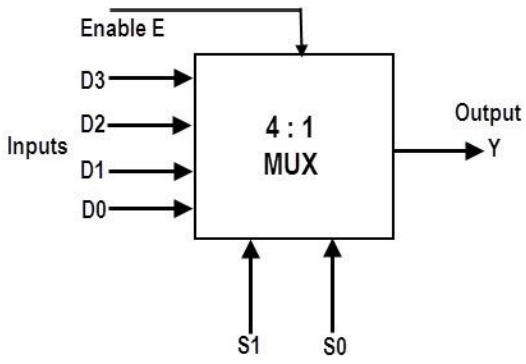


$$Y = ED_0\bar{S} + ED_1S$$

The logic circuit for a 2-to-1 multiplexer with enable is:



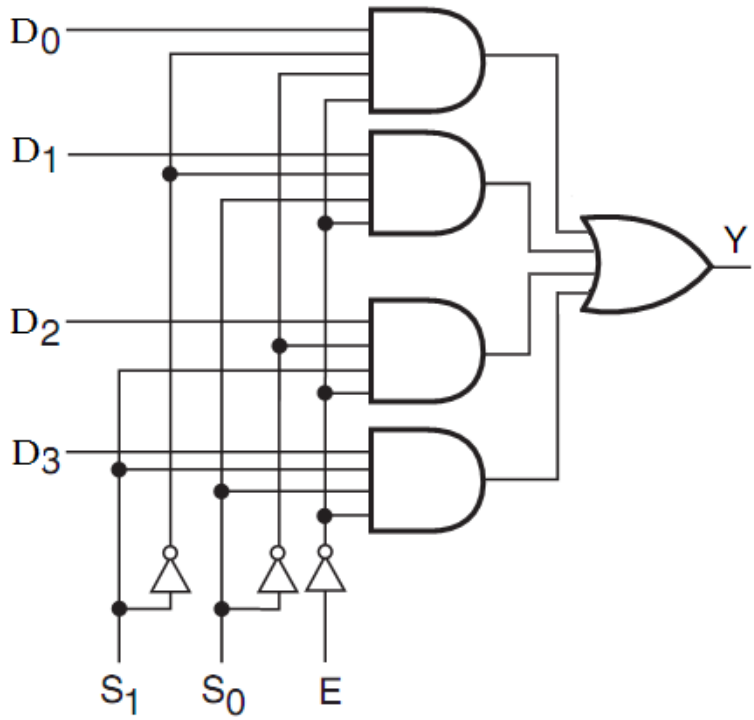
Example
Design 4-to-1 multiplexer with an enable input.
Solution



Enable E	Data Select		Output data Y
	S_1	S_0	
0	x	x	0
1	0	0	D_0
1	0	1	D_1
1	1	0	D_2
1	1	1	D_3

$$Y = ED_0\bar{S}_1\bar{S}_0 + ED_1\bar{S}_1S_0 + ED_2S_1\bar{S}_0 + ED_3S_1S_0$$

The logic circuit for a 4-to-1 multiplexer with enable is:



Implementing Boolean Functions with Multiplexers

The Boolean function may be implemented in 2^n to 1 multiplexer.

- If we have a Boolean function of n variables, we take $n-1$ of these variables and connect them to the selection lines of a multiplexer (let's say these are "select variables").
- The remaining single variable (MSB variable) of the function is used for the inputs of the multiplexer (let's say these are "input variable").
- Now form the implementation table:
  - First row lists all those minterms where "input variable" is complemented (say 0).
  - Second row lists all those minterms where "input variable" is in its normal form (say 1).
- The minterms are circled as per the given Boolean function. Now use the following steps to find out final multiplexer inputs.
  - If the 2 minterms in a column are not circled, 0 is placed to the corresponding multiplexer inputs.
  - If the 2 minterms in a column are circled, 1 is placed to the corresponding multiplexer inputs.
  - If the minterms in the second row is circled and the first row is not circled, apply second row of variable to the corresponding multiplexer inputs.
  - If the minterms in the first row is circled and not the second row, apply first row of the variable to the corresponding multiplexer inputs.

Example

Implement the following Boolean function using 8-to-1 multiplexer.

$Y(A,B,C,D) = \sum m(1,3,4,11,12,13,14,15)$

Solution

Total number of variable $n = 4\ (A,B,C,D)$

Number of select lines: $n-1 = 3\ (B,C,D)$

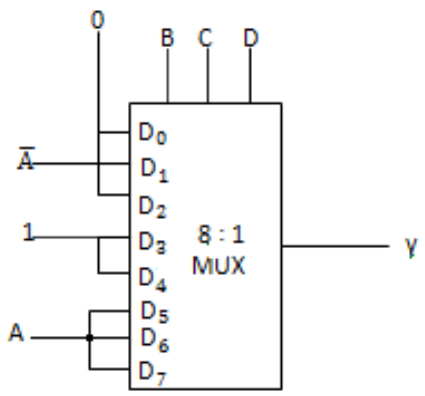
All the minterms are divided into 2 groups

The first group (0-7) minterms are entered in the first row (Variable A =0)

The second group (8-15) minterms are entered in the second row (Variable A= 1)

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	**1**	2	**3**	**4**	5	6	7
A	8	9	10	**11**	**12**	**13**	**14**	**15**
	0	$\bar{A}$	0	1	1	A	A	A

Circle the minterm number as per function.



Example

Implement the following Boolean function using 8-to-1multiplexer

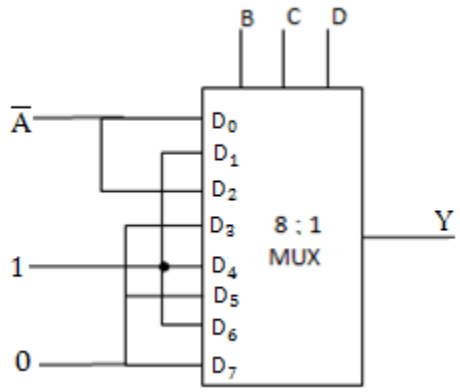
$Y(A,B,C,D) = \sum m(0,1,3,2,4,6,9,12,14)$

Solution

Total number of variable $n = 4\ (A,B,C,D)$

Select lines: $n-1 = 3\ (B,C,D)$

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	**0**	**1**	**2**	3	**4**	5	**6**	7
A	8	**9**	10	11	**12**	13	**14**	15
	$\bar{A}$	1	$\bar{A}$	0	1	0	1	0



Example

Implement the following Boolean function using 8-to-1multiplexer

$Y(A,B,C,D) = \prod M\ (0,3,5,6,8,9,10,12,14)$

Solution

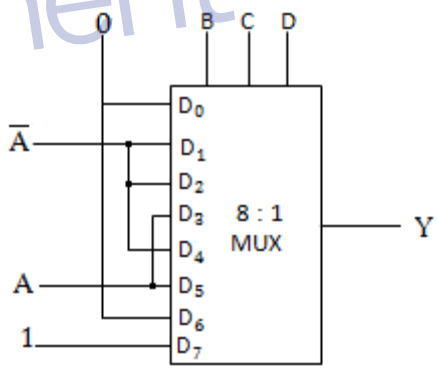
The given maxterms are inverted to obtain minterms. From minterms, we can implement the above Boolean function using 8-to-1 multiplexer

$Y(A,B,C,D) = \sum m(1,2,4,7,11,13,15)$

Total number of variable $n = 4\ (A,B,C,D)$

Select lines: $n-1 = 3\ (B,C,D)$

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	**1**	**2**	3	**4**	5	6	7
A	8	9	10	**11**	12	**13**	14	**15**
	0	$\bar{A}$	$\bar{A}$	A	$\bar{A}$	A	0	1



Example

Implement the following Boolean function using 8-to-1multiplexer

$Y(A,B,C,D) = \sum m(0,2,6,10,11,12,13) + \sum d(3,8,14)$

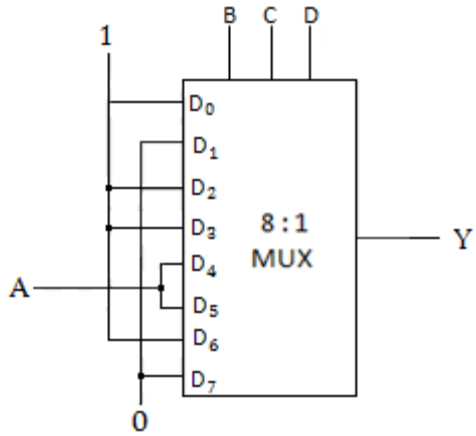
Solution

The Boolean function has three don't care conditions which can be treated as either 0's or 1's. We consider don't care conditions as1's.

Total number of variable $n = 4\ (A,B,C,D)$

Select lines: $n-1 = 3\ (B,C,D)$

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	**0**	1	**2**	**3**	4	5	**6**	7
A	**8**	9	**10**	**11**	**12**	**13**	**14**	15
	1	0	1	1	A	A	1	0



Example

Implement the following Boolean function using 4-to-1multiplexer

$F = \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$

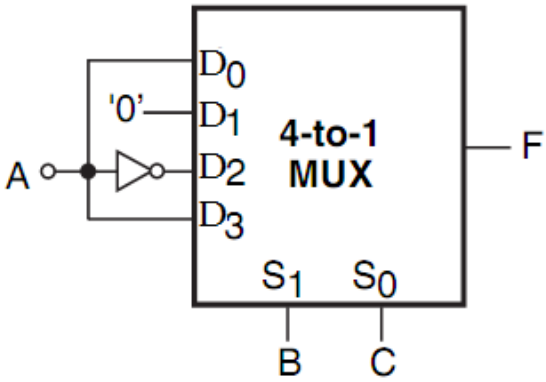
Solution

$F(A,B,C) = \sum m(2,4,7)$

Total number of variable $n = 3\ (A,B,C)$

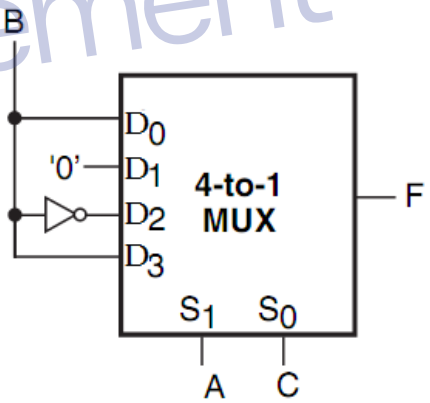
Select lines: $n - 1 = 2(B,C)$

	D_0	D_1	D_2	D_3
$\bar{A}$	0	1	**2**	3
A	**4**	5	6	**7**
	A	0	$\bar{A}$	A



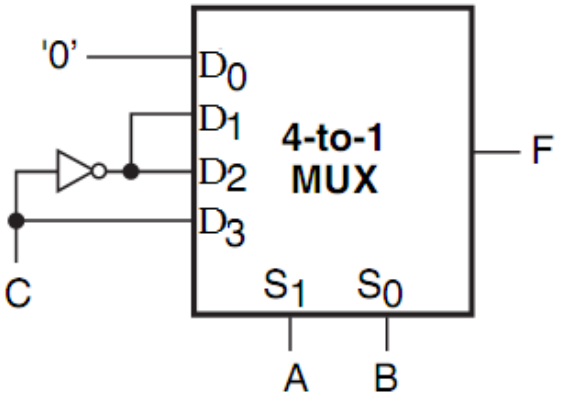
Or

	D_0	D_1	D_2	D_3
$\bar{B}$	0	1	**4**	5
B	**2**	3	6	**7**
	B	0	$\bar{B}$	B



Or

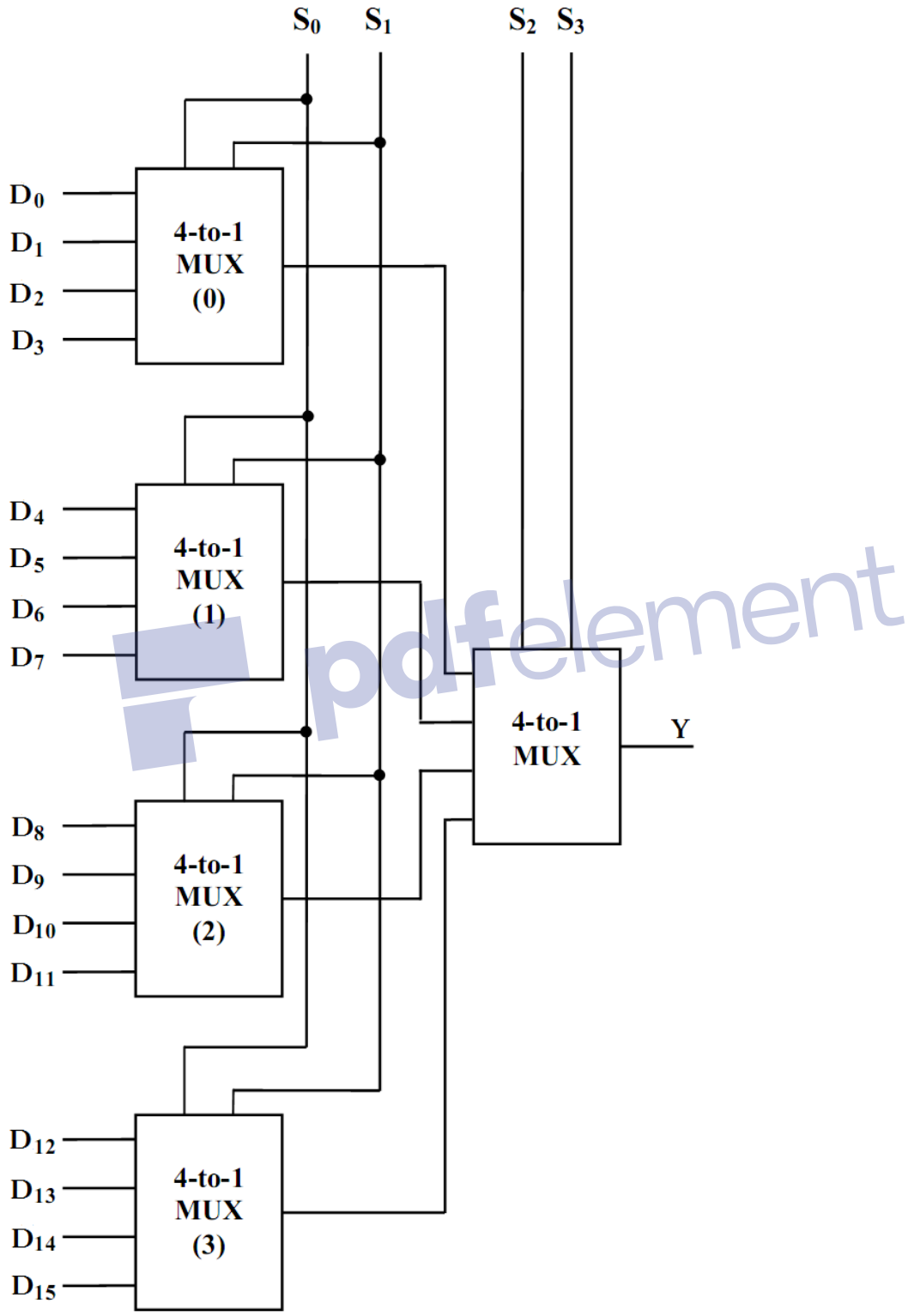
	D_0	D_1	D_2	D_3
$\bar{C}$	0	**2**	**4**	6
C	1	3	5	**7**
	0	$\bar{C}$	$\bar{C}$	C



Example

Construct a 16-to-1 multiplexer using only 4-to-1 multiplexer.

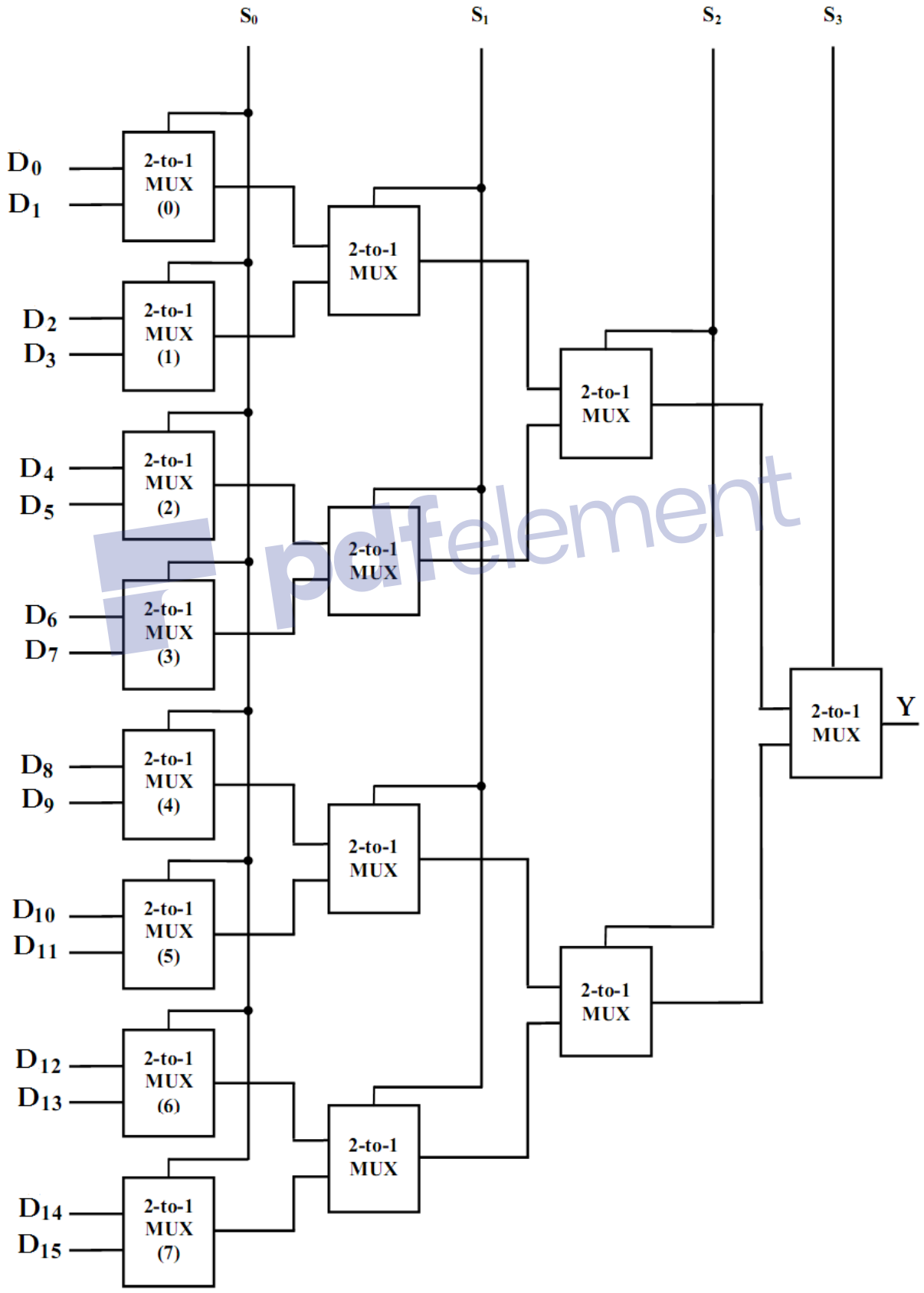
Solution



Example

Construct a 16-to-1 multiplexer using only 2-to-1 multiplexer.

Solution

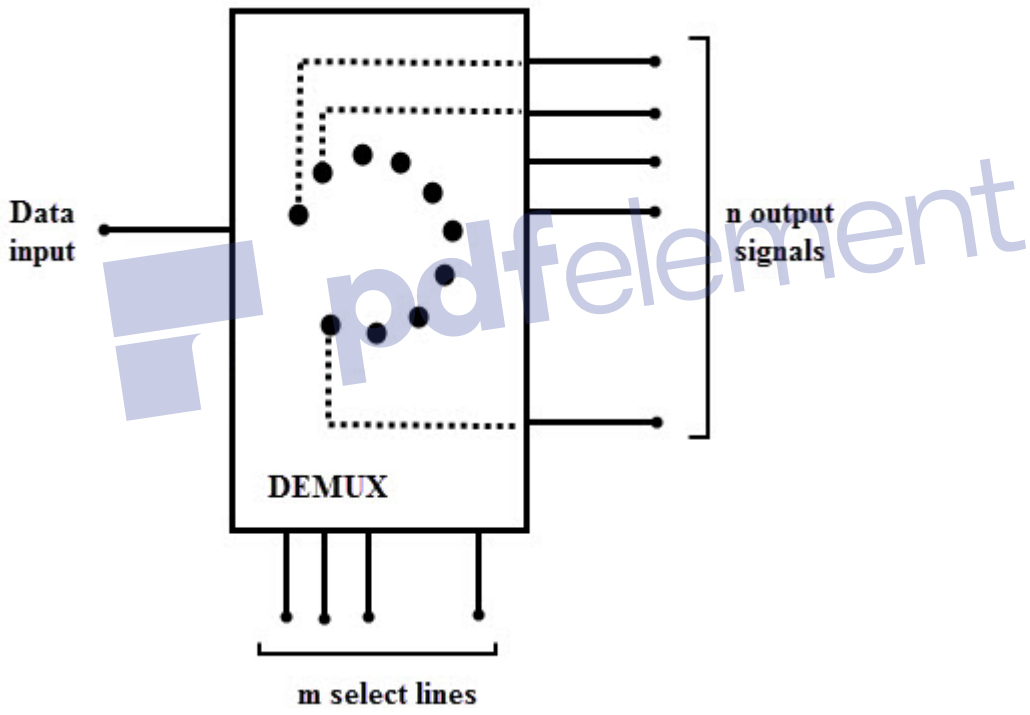


4.4.2Demultiplexer (DEMUX)

A demultiplexer (DEMUX) basically reverses the multiplexing function. It takes digital information from one line and distributes it to a given number of output lines. The demultiplexer is also known as a data distributor.

A demultiplexer is a 1-to-N device where as the multiplexer is an N-to-1 device. The figure below shows the block diagram of a demultiplexer or simply a DEMUX.

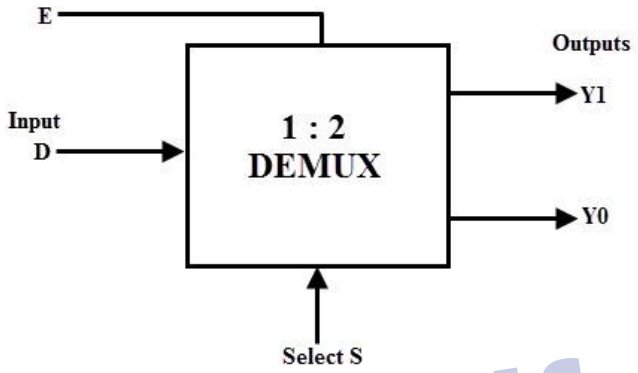
It consists of 1 input line, n output lines and m select lines. In this, m selection lines are required to produce 2m possible output lines (consider $2^m = n$). For example, a 1-to-4 demultiplexer requires 2 select lines to control the 4 output lines.



There are several types of demultiplexers based on the output configurations such as 1:4, 1:8 and 1:16.

1-to-2 Demultiplexer

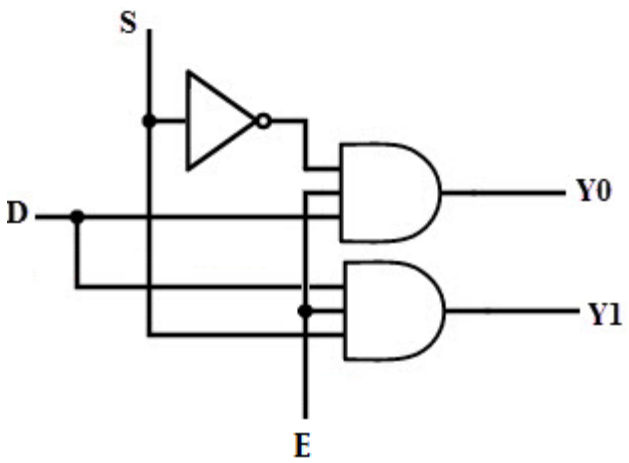
A 1-to-2 demultiplexer with enable consists of one input line, two output lines, one input enable and one select line. The signal on the select line helps to switch the input to one of the two outputs. The figure below shows the block diagram of a 1-to-2 demultiplexer with additional enable input. In the figure, there are only two possible ways to connect the input to output lines, thus only one select signal is enough to do the demultiplexing operation. When the select input is low, then the input will be passed to Y0 and if the select input is high then the input will be passed to Y1.



The truth table of a 1-to-2 demultiplexer is shown below.

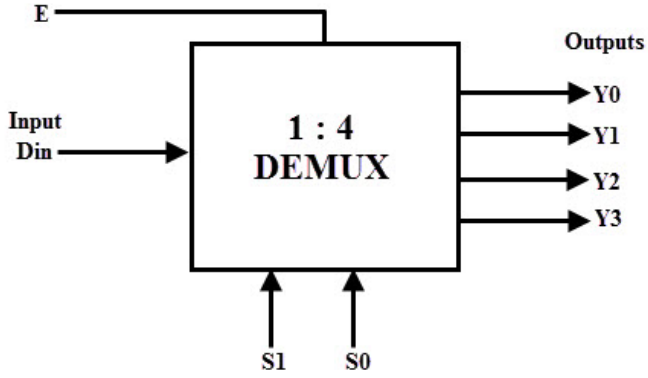
E	Select	Input	Output	
	S	D	Y_0	Y_1
0	X	D	0	0
1	0	D	D	0
1	1	D	0	D

$Y_0 = E\bar{S}D$
$Y_1 = ESD$



1-to-4 Demultiplexer

The block diagram of 1:4 DEMUX with additional enable is shown below.



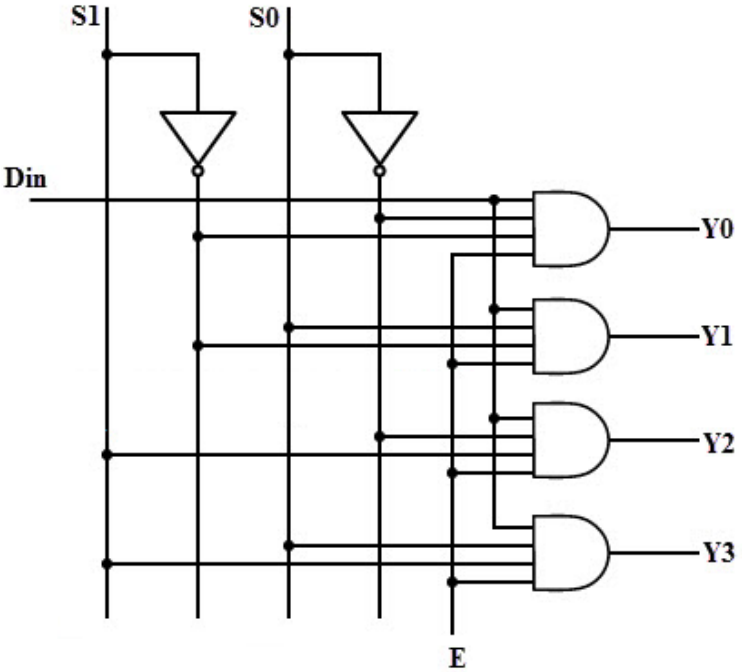
The truth table of this type of demultiplexer is given below:

Enable E	Select Inputs		Input	Output			
	S_1	S_0	D	Y_0	Y_1	Y_2	Y_3
0	x	x	D	0	0	0	0
1	0	0	D	D	0	0	0
1	0	1	D	0	D	0	0
1	1	0	D	0	0	D	0
1	1	1	D	0	0	0	D

The output logic can be expressed as min terms and are given below.

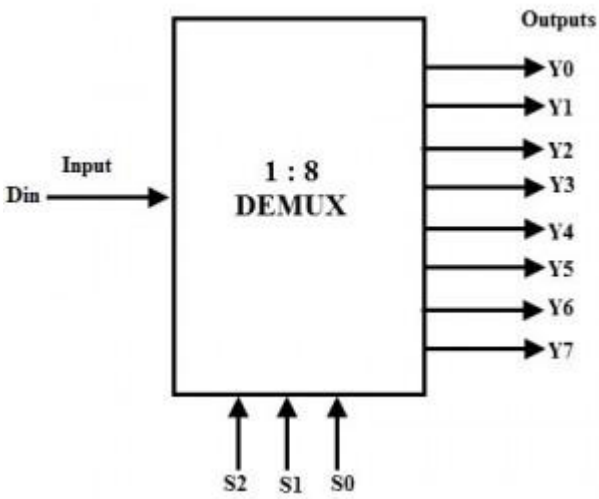
$Y_0 = E\bar{S}_1\bar{S}_0 D$, $Y_1 = E\,\bar{S}_1 S_0 D$, $Y_2 = E\,S_1\bar{S}_0 D$, $Y_3 = E\,S_1 S_0 D$

Where: D is the input data, Y_0 to Y_3 are outputs lines and S_0 & S_1 are select lines.



1-to-8 Demultiplexer

The below figure shows the block diagram of a 1-to-8 demultiplexer without enable



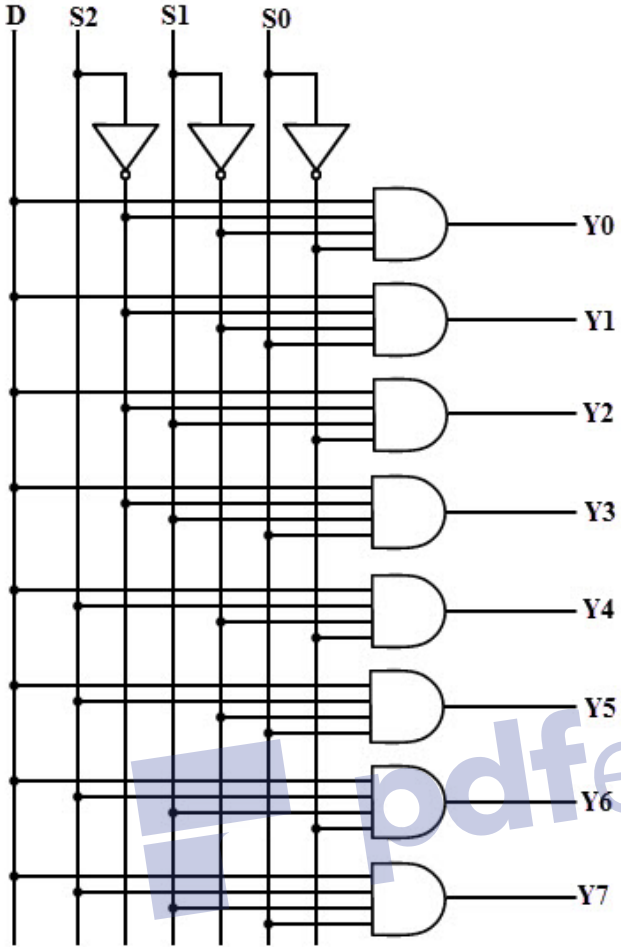
The truth table for this type of demultiplexer is shown below.

Select Inputs			Input	Output							
S_2	S_2	S_0	D	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	0	0	D	D	0	0	0	0	0	0	0
0	0	1	D	0	D	0	0	0	0	0	0
0	1	0	D	0	0	D	0	0	0	0	0
0	1	1	D	0	0	0	D	0	0	0	0
1	0	0	D	0	0	0	0	D	0	0	0
1	0	1	D	0	0	0	0	0	D	0	0
1	1	0	D	0	0	0	0	0	0	D	0
1	1	1	D	0	0	0	0	0	0	0	D

The Boolean expressions for all the outputs can be written as follows.

$Y_0 = \bar{S}_2\bar{S}_1\bar{S}_0\ D$	$Y_4 = S_2\bar{S}_1\bar{S}_0\ D$
$Y_1 = \bar{S}_2\ \bar{S}_1 S_0 D$	$Y_5 = S_2\ \bar{S}_1 S_0 D$
$Y_2 = \bar{S}_2\ S_1\bar{S}_0 D$	$Y_6 = S_2\ S_1\bar{S}_0 D$
$Y_3 = \bar{S}_2\ S_1 S_0 D$	$Y_7 = S_2\ S_1 S_0 D$

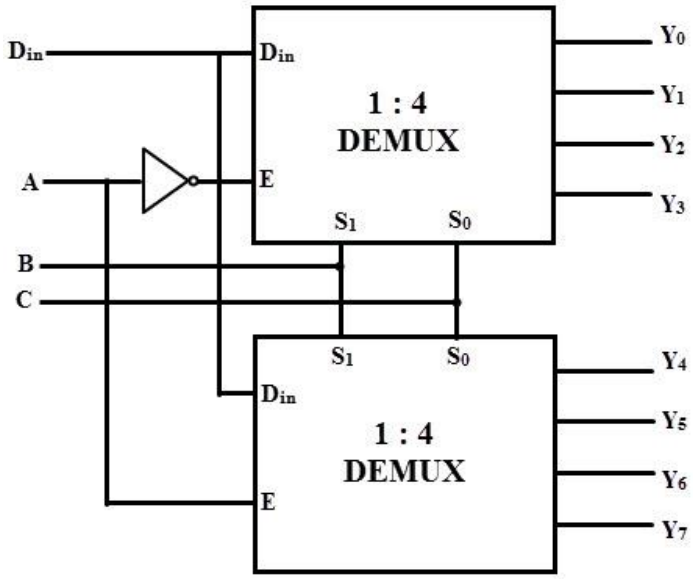
From these obtained equations, the logic diagram of this demultiplexer as shown in below figure.



Example

Construct a 1-to-8 DEMUX using Two 1-to- 4 Demultiplexers.

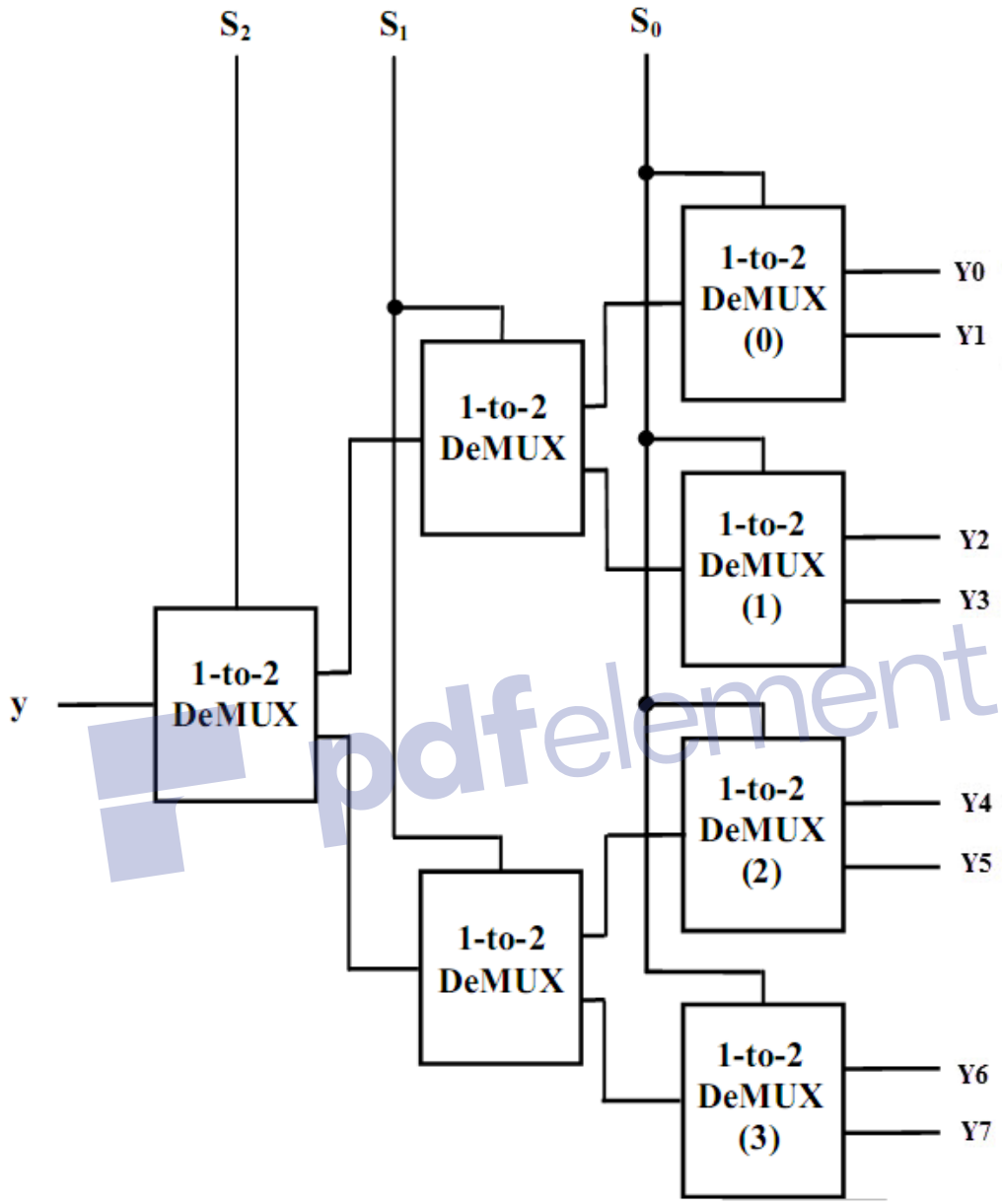
Solution



Example

Construct a 1-to-8 DEMUX using only 1-to- 2 Demultiplexers.

Solution



Implementation of Full Subtractor Using 1-to-8 DEMUX

As similar to the multiplexers, demultiplexers are also used for Boolean function implementation as well as combinational circuit design. We can design the demultiplexer to produce any truth table output by correspondingly controlling the select lines.

The truth table below shows the output of a full subtractor.

Inputs			outputs	
Minuend A	Subtrahend B	Borrow In B_{in}	Difference D	Borrow Out B_o
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

From the above table, the full subtractor output D can be written as

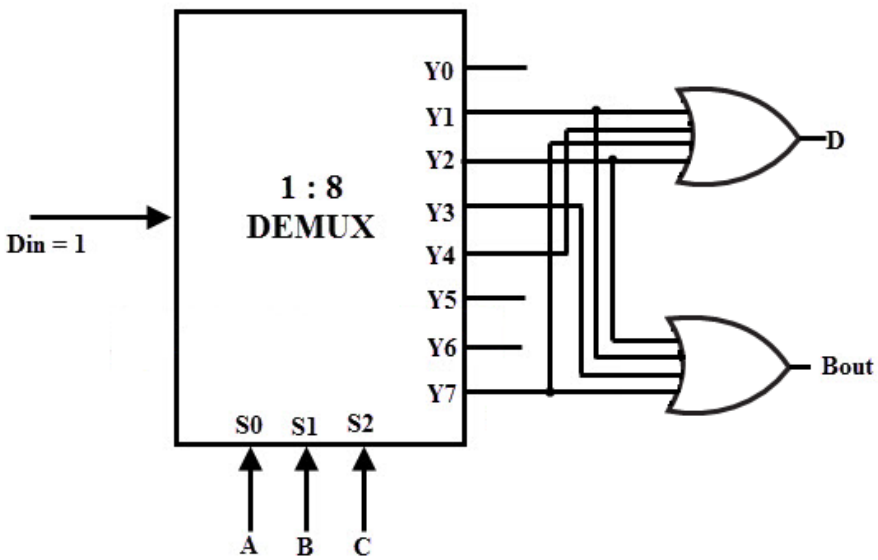
$D = f(A,B,C) = \sum m(1,2,4,7)$

And the borrow output can be expressed as

$B_{out} = F(A,B,C) = \sum m(1,2,3,7)$

From these Boolean functions, a demultiplexer for producing full subtractor output can be built by properly configuring the 1-to-8 DEMUX such that with input ***D = 1*** it gives the minterms at the output.

And by logically ORing these minterms, the outputs of difference and borrow can be obtained as shown in figure.



Code Conversion

Binary-to-Gray and Gray-to-Binary Conversion

We will now see how XOR gates can be used for these conversions. Figure1 show a 4-bit binary-to-gray code conversion, and figure2 illustrates 4-bit gray-to-binary converter.

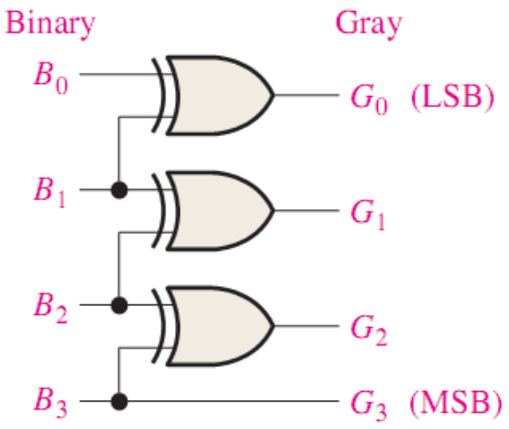


Figure (1): 4-bit binary-to-Gray conversion logic

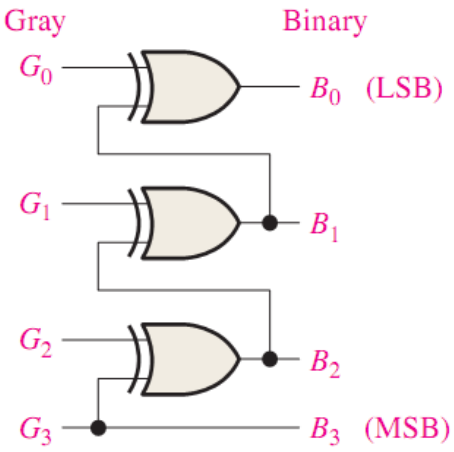


Figure (2):4-bit Gray-to-binary conversion logic.

5- Sequential circuits (Latches, Flip-Flops, and Timers)

In the same way that gates are the building blocks of combinatorial circuits, latches and *flip-flops* are the building blocks of sequential circuits. Latches can be built from gates, and flip-flops can be built from latches. This fact will make it somewhat easier to understand latches and flip-flops.

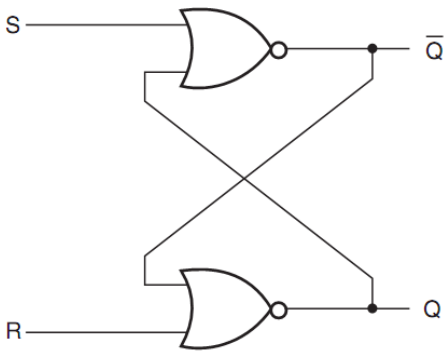
5.1 Latches and Flip Flops

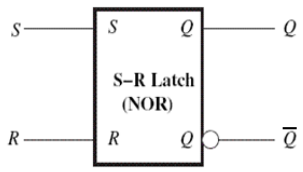
Latches and flip flops are the basic elements and these are used to store information. One flip flop and latch can store one bit of data. The latch checks input continuously and changes the output whenever there is a change in input. But, flip flop is a combination of latch and clock that continuously checks input and changes the output time adjusted by the clock. In this article, we are going to look at the operations of the numerous latches and flip-flops.

Both Latches and flip flops are circuit elements wherein the output not only depends on the current inputs, but also depends on the previous input and outputs. The main difference between the latch and flip flop is that a flip flop has a clock signal, whereas a latch does not. Basically, there are four types of latches and flip flops: SR, D, JK and T. The major differences between these types of flip flops and latches are the number of i/ps they have and how they change the states.

5.1.1 The S-R Latch

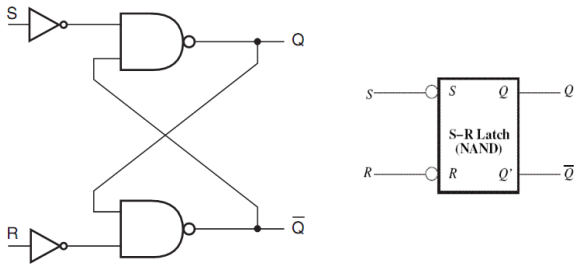
There are two types of S-R Latch which are $(S-R\ )NAND$ latch $and\ (S-R\ )NOR$ latch. The diagrams below show the logic symbol and logic gate representation of S-R NOR gates.





Truth table for S-R NOR latch (active-HIGH input)				
Inputs		Outputs		Comments
S	***R***	***Q***	$\overline{Q}$	
0	0	N.C	NC	No change. Latch remains in present state.
1	0	1	0	Latch SET.
0	1	0	1	Latch RESET.
1	1	?	?	Invalid state

The diagrams below show the logic symbol and logic gate representation of S-R NAND gates.



Truth table for an active-LOW input latch with NAND gates.				
Inputs		Outputs		Comments
S	***R***	***Q***	$\overline{Q}$	
0	0	NC	NC	No change. Latch remains in present state.
1	0	1	0	Latch SET.
0	1	0	1	Latch RESET.
1	1	?	?	Invalid state

5.2 Flip Flops

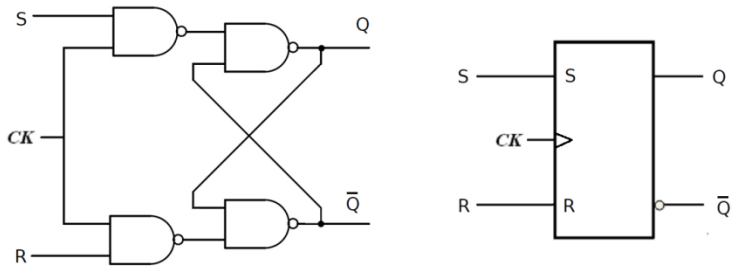
A flip flops is a bistable logic circuit which has two stable states. It's capable of residing in either of these two states (SET or RESET) until a new clock activation trigger is applied.

1- Edge-Triggered Flip-Flops

An edge-triggered flip-flop changes states either at the positive edge (rising edge) or at the negative edge (falling edge) of the clock pulse on the control input and is sensitive to its inputs only at this transition of the clock.

A- The edge-triggered S-R flip-flop:

The logic symbol and logic circuit for the SET-RESET flip-flop are shown below:



It has the inputs (S and R) and the clock input terminal. The outputs are Q and its complement$\overline{Q}$.

As illustrated in the truth table below, the output is fixed (unchanged) when the input has the state (S=0 , R=0). The output in SET or RESET when the

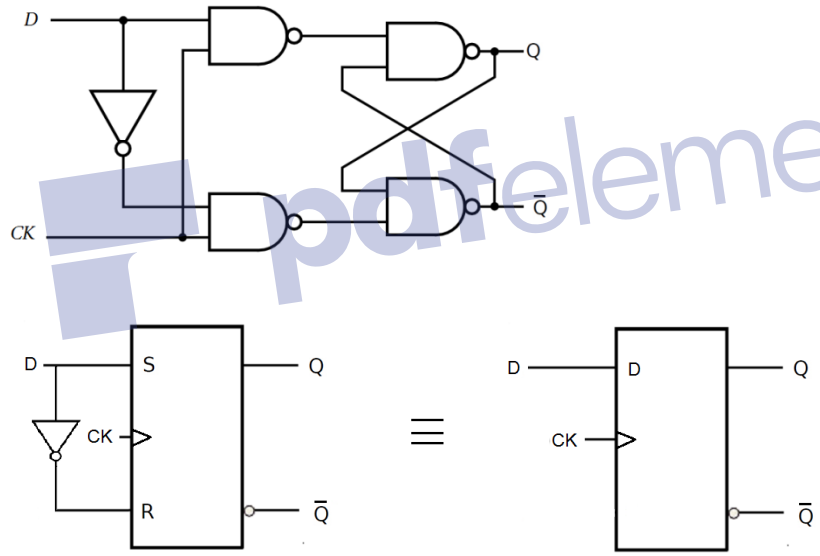
input has the states (S=1 , R=0) or (S=0 , R=1), respectively. Finally, the output is an **invalid** state when (S=1 , R=1)

Truth table for S-R flip-flop					
Inputs			Outputs		Comments
S	***R***	**CK**	***Q***	$\overline{Q}$	
0	0	↑	Q_o	$\overline{Q}_o$	No Change.
0	1	↑	0	1	RESET.
1	0	↑	1	0	SET.
1	1	↑	?	?	Invalid

Where ↑ is the positive edge of a clock pulse, and Q_o and $\overline{Q}_o$ are old value of Q and $\overline{Q}$.

B- The edge-triggered D flip-flop:

The addition of an inverter to an S-R flip-flop creates a D flip-flop as shown below:



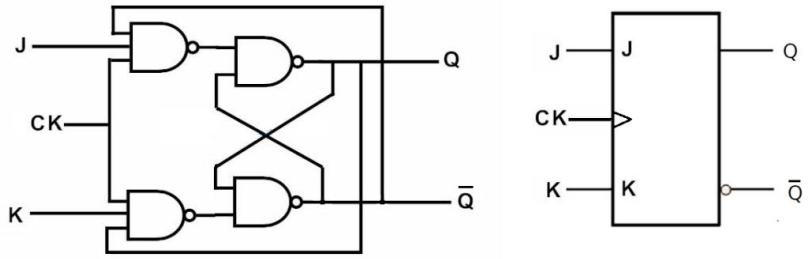
Notice that this flip-flop has only one input in addition to the clock. If D is high (1) when a clock pulse is applied, then the flip-flop will SET. If D is low (0) when a clock pulse is applied, the flip-flop will RESET, as shown in the truth table.

Truth table for D flip-flop				
Inputs		Outputs		Comments
D	CK	***Q***	$\overline{Q}$	
0	↑	0	1	RESET (store 0)
1	↑	1	0	SET (store 1)

The 7474 IC is a dual edge triggered D flip-flop
The 7476 IC is a dual edge triggered JK flip-flop

C- The edge- triggered JK flip-flop:

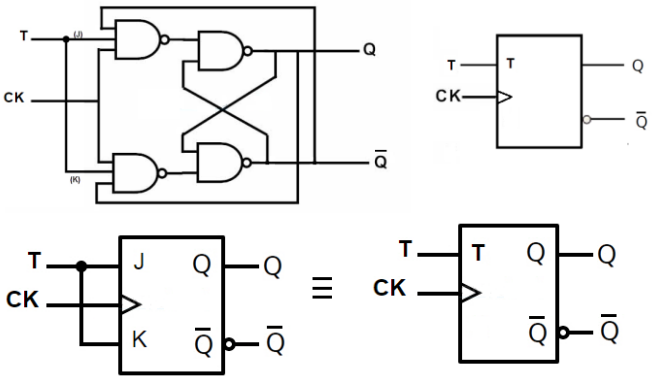
The functioning of the JK flip-flop is identical to that of the S-R flip-flop in SET, RESET and (no-change) condition of operation. The difference is that the JK flip-flop has no invalid state. The following truth table summarizes the operation of an edge triggered JK flip-flop.



Truth table for J-K flip-flop					
Inputs			Outputs		Comments
J	***K***	CK	***Q***	$\overline{Q}$	
0	0	↑	Q_o	$\overline{Q}_o$	No Change.
0	1	↑	0	1	RESET.
1	0	↑	1	0	SET.
1	1	↑	$\overline{Q}_o$	Q_o	Toggle

D- The edge- triggered T flip-flop:

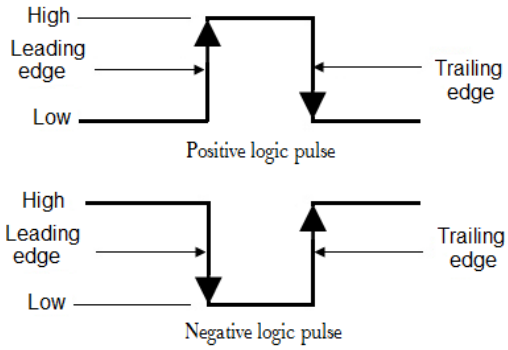
The J input and K input of the JK flip – flop are connected together and provided with the T input. The logic circuit of a T flip – flop constructed from a JK flip – flop is shown below.



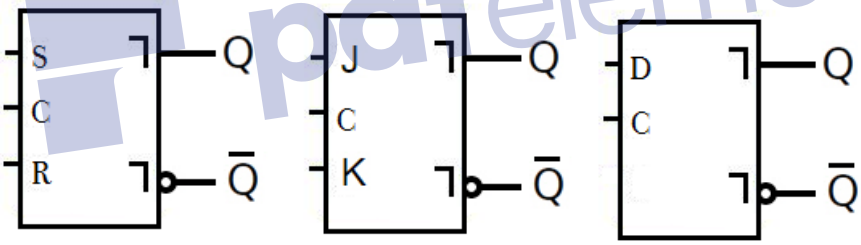
Truth table for D flip-flop				
Inputs		Outputs		Comments
T	CK	***Q***	$\overline{Q}$	
0	↑	Q_o	$\overline{Q}_o$	No Change.
1	↑	$\overline{Q}_o$	Q_o	Toggle

E- Pulse-triggered (master-slave) flip-flops

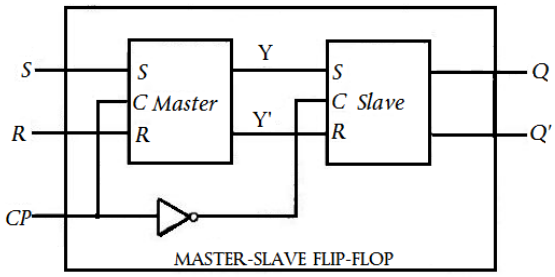
The term pulse-triggered means that data are entered into the flip-flop on the leading edge of the clock pulse, but the output does not reflect the input state until the trailing edge of the clock pulse. Therefore, the data must not while the clock pulse is HIGH.

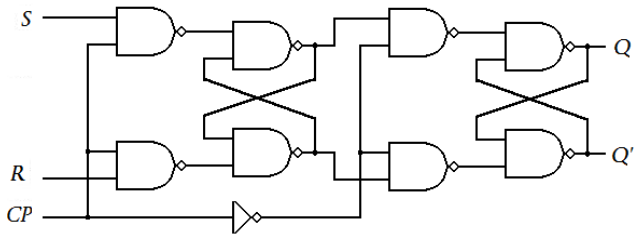


The logic symbols of pulse triggered (master-slave) flip-flops are given below:
The three basic types of pulse-triggered flip-flops are S-R, J-K and D. Their logic symbols are shown below.



The truth tables for the above pulse-triggered flip-flops are all the same as that for the edge-triggered flip-flops, except for the way they are clocked. These flip-flops are also called Master-Slave flip-flops simply because their internal construction is divided into two sections. The slave section is basically the same as the master section except that it is clocked on the inverted clock pulse and is controlled by the outputs of the master section rather than by the external inputs. The logic diagram for a basic master-slave S-R flip-flop is shown below.



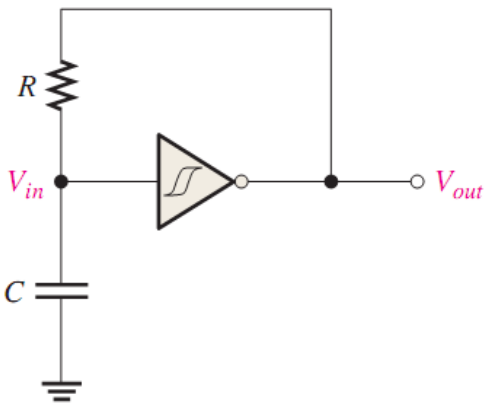


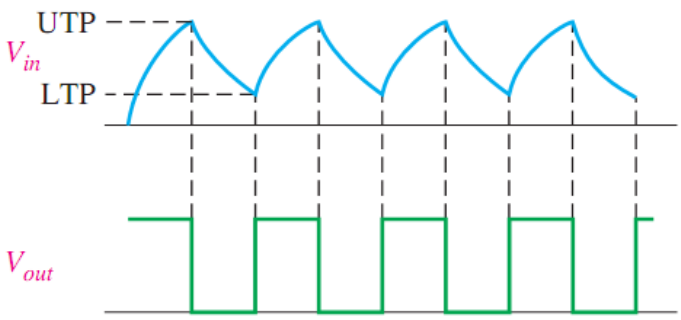
5.3 CLOCK GENERATOR CIRCUITS

Flip-flops have two stable states; therefore, we can say that they are bistable multivibrators. One-shots have one stable state, and so we call them monostable multivibrators. A third type of multivibrator has no stable states; it is called an astable or free-running multivibrator. This type of logic circuit switches back and forth (oscillates) between two unstable output states. It is useful for generating clock signals for synchronous digital circuits. Several types of astable multivibrators are in common use. We will present three of them without any attempt to analyze their operation. They are presented here so that you can construct a clock generator circuit if needed for a project or for testing digital circuits in the lab.

1-The Astable Multivibrator

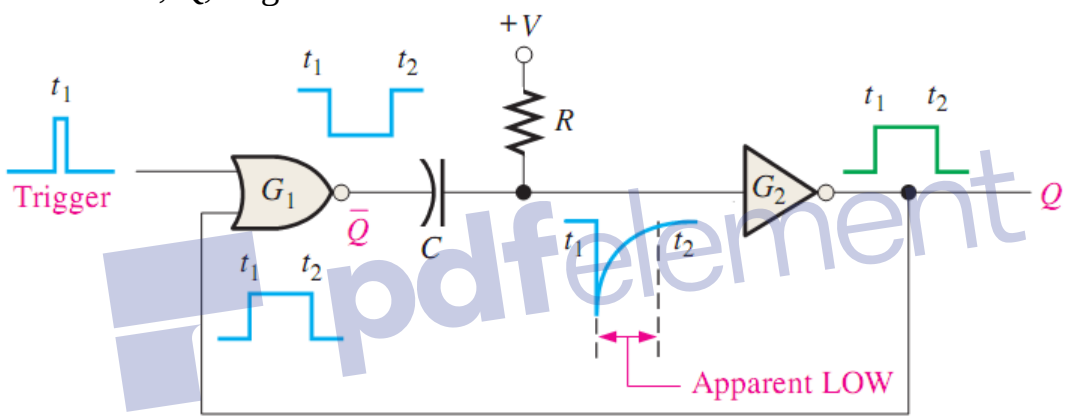
An astable multivibrator is a device that has no stable states; it changes back and forth (oscillates) between two unstable states without any external triggering. The resulting out-put is typically a square wave that is used as a clock signal in many types of sequential logic circuits. Astable multivibrators are also known as pulse oscillators. Figure below shows a simple form of astable multivibrator using an inverter with hysteresis (Schmitt trigger) and an RC circuit connected in a feedback arrangement. When power is first applied, the capacitor has no charge; so the input to the Schmitt trigger inverter is LOW and the output is HIGH. The capacitor charges through R until the inverter input voltage reaches the upper trigger point (UTP. At this point, the inverter output goes LOW, causing the capacitor to discharge back through R. When the inverter input voltage decreases to the lower trigger point (LTP), its output goes HIGH and the capacitor charges again. This charging/discharging cycle continues to repeat as long as power is applied to the circuit, and the resulting output is a pulse waveform, as indicated.





2-Monostable Multivibrator (One-Shots)

The monostable multivibrator or one-shot, is a device with only one stable state. A monostable multivibrator is normally in its stable state and will change to its unstable state only when triggered. Once it is triggered, the monostable multivibrator remains in its unstable state for a predetermined length of time and then automatically returns to its stable state. The time that the device stays in its unstable state determines the pulse width of its output.
Figure below shows a basic monostable multivibrator (one-shot) that is composed of a logic gate and an inverter. When a pulse is applied to the trigger input, the output of gate G1 goes LOW. This HIGH-to-LOW transition is coupled through the capacitor to the input of inverter G2. The apparent LOW on G2 makes its output go HIGH. This HIGH is connected back into G1, keeping its output LOW. Up to this point the trigger pulse has caused the output of the monostable multivibrator, Q, to go HIGH.



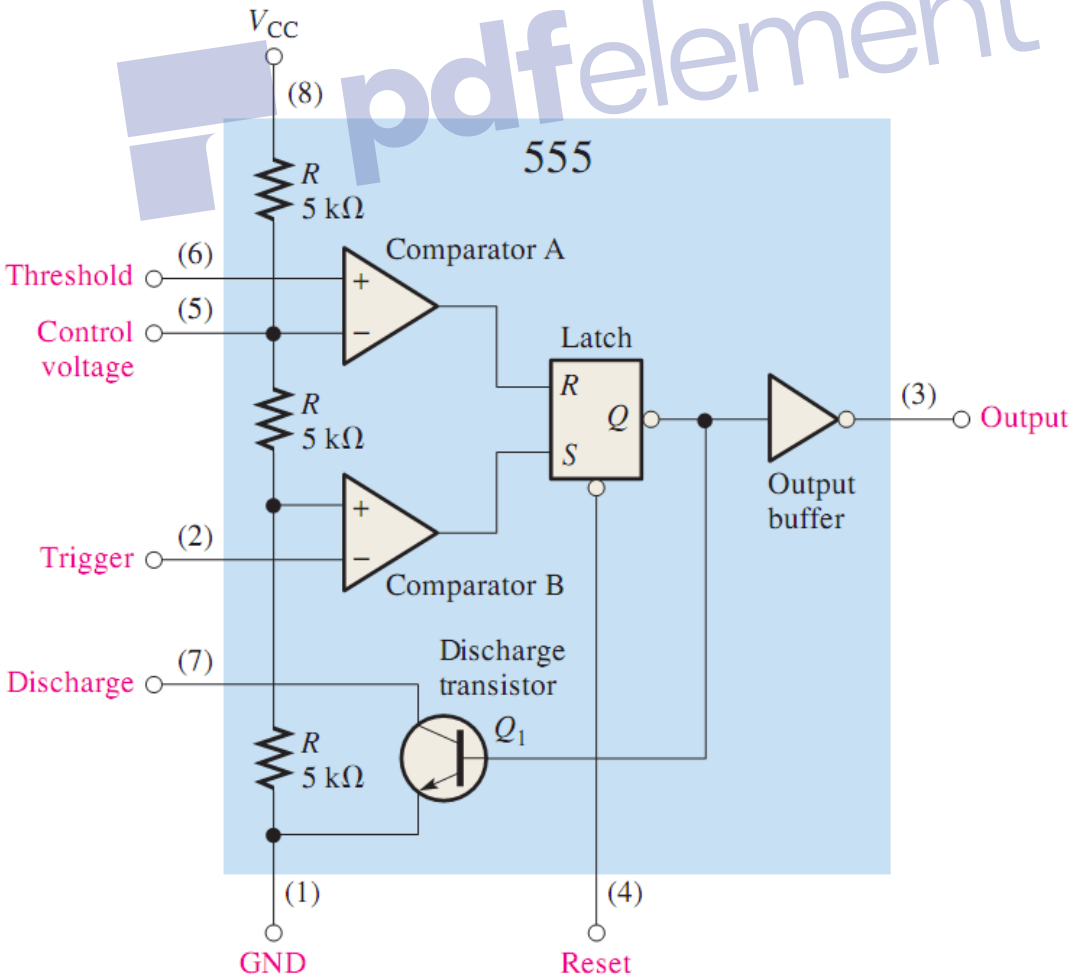
The capacitor immediately begins to charge through R toward the high voltage level. The rate at which it charges is determined by the RC time constant. When the capacitor charges to a certain level, which appears as a HIGH to G2, the output goes back LOW. To summarize, the output of inverter G2 goes HIGH in response to the trigger input. It remains HIGH for a time set by the RC time constant. At the end of this time, it goes LOW. A single narrow trigger pulse produces a single output pulse whose time duration is controlled by the RC time constant.

3-The 555 Timer:

The 555 timer is a versatile and widely used IC device because it can be configured in two different modes as either a monostable multivibrator (one-shot) or as an astable multivibrator (pulse oscillator).

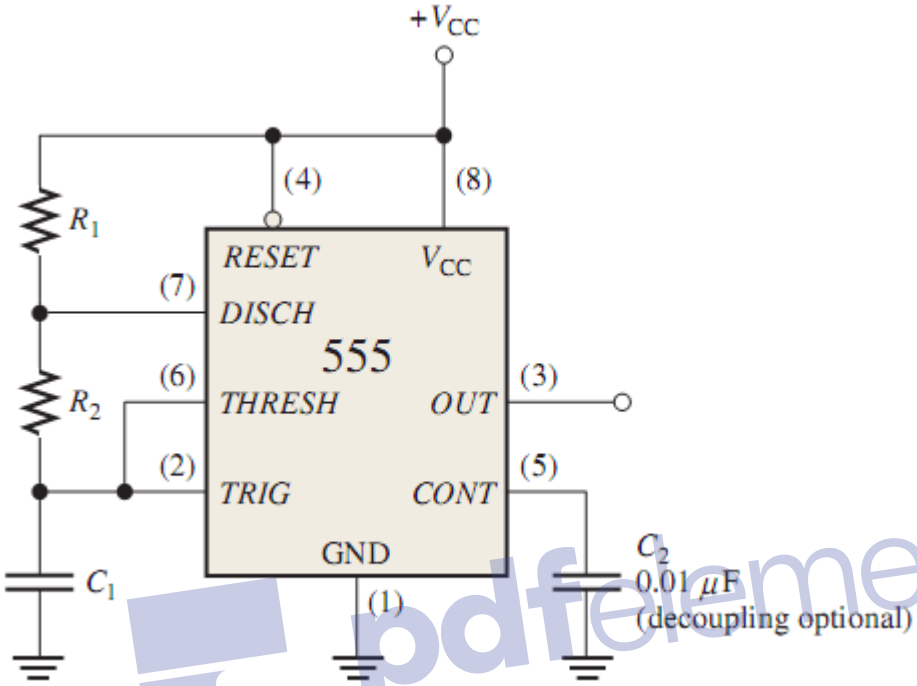
The 555 Timer Operations:

A functional diagram showing the internal components of a 555 timer is shown in Figure below. The comparators are devices whose outputs are HIGH when the voltage on the positive (+) input is greater than the voltage on the negative (-) input and LOW when the - input voltage is greater than the + input voltage. The voltage divider consisting of three 5kΩ resistors provides a trigger level of $\frac{1}{3} V_{CC}$ and a threshold level of $\frac{2}{3} V_{CC}$. The control voltage input (pin 5) can be used to externally adjust the trigger and threshold levels to other values if necessary. When the normally HIGH trigger input momentarily goes below $\frac{1}{3} V_{CC}$, the output of comparator B switches from LOW to HIGH and sets the S-R latch, causing the output (pin 3) to go HIGH and turning the discharge transistor Q_1 off. The output will stay HIGH until the normally LOW threshold input goes above $\frac{2}{3} V_{CC}$ and causes the output of comparator A to switch from LOW to HIGH. This resets the latch, causing the output to go back LOW and turning the discharge transistor on. The external reset input can be used to reset the latch independent of the threshold circuit. The trigger and threshold inputs (pins 2 and 6) are controlled by external components connected to produce either monostable or astable action.



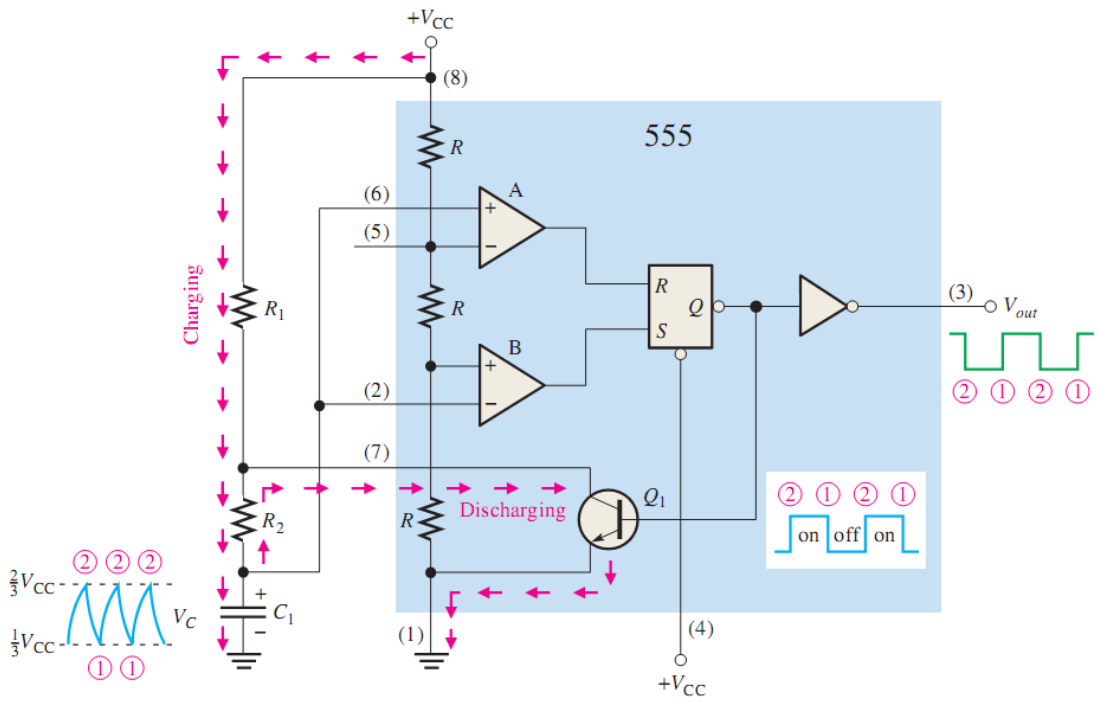
A-The 555 Timer as an Astable Multivibrator

A 555 timer connected to operate as an astable multivibrator is shown in Figure below. Notice that the threshold input (THRESH) is now connected to the trigger input (TRIG). The external components R1, R2, and C1 form the timing network that sets the frequency of oscillation. The 0.01 µF capacitor, C2, connected to the control (CONT) input is strictly for decoupling and has no effect on the operation; in some cases it can be left off.



Initially, when the power is turned on, the capacitor (C1) is uncharged and thus the trigger voltage (pin 2) is at 0 V. This causes the output of comparator B to be HIGH and the output of comparator A to be LOW, forcing the output of the latch, and thus the base of Q1, LOW and keeping the transistor off. Now, C1 begins charging through R1 and R2, as indicated in Figure below. When the capacitor voltage reaches

$\frac{1}{3} V_{CC}$, comparator B switches to its LOW output state; and when the capacitor voltage reaches $\frac{2}{3} V_{CC}$, comparator A switches to its HIGH output state. This resets the latch, causing the base of Q1 to go HIGH and turning on the transistor. This sequence creates a discharge path for the capacitor through R2 and the transistor, as indicated. The capacitor now begins to discharge, causing comparator A to go LOW. At the point where the capacitor discharges down to $\frac{1}{3} V_{CC}$, comparator B switches HIGH; this sets the latch, making the base of Q1 LOW and turning off the transistor. Another charging cycle begins, and the entire process repeats.



$$f = \frac{1.44}{(R_1 + 2R_2)C_1}$$

$t_H = 0.7(R_1 + R_2)C_1$ Where: t_H Is the time that the output is HIGH.

$t_L = 0.7R_2C_1$ Where: t_H Is the time that the output is HIGH.

The period, T, of the output waveform is the sum of t_H and t_L

$$T = t_H + t_L = 0.7(R_1 + 2R_2)C_1$$

$$f = \frac{1}{T} = \frac{1.44}{(R_1 + 2R_2)C_1}$$

Finally, the duty cycle is

$$Duty\ cycle = \frac{t_H}{T} = \frac{t_H}{t_H + t_L}$$

$$Duty\ cycle = \left(\frac{R_1 + R_2}{R_1 + 2R_2}\right)100\%$$

$$Duty\ cycle = \left(\frac{R_1}{R_1 + R_2}\right)100\%$$

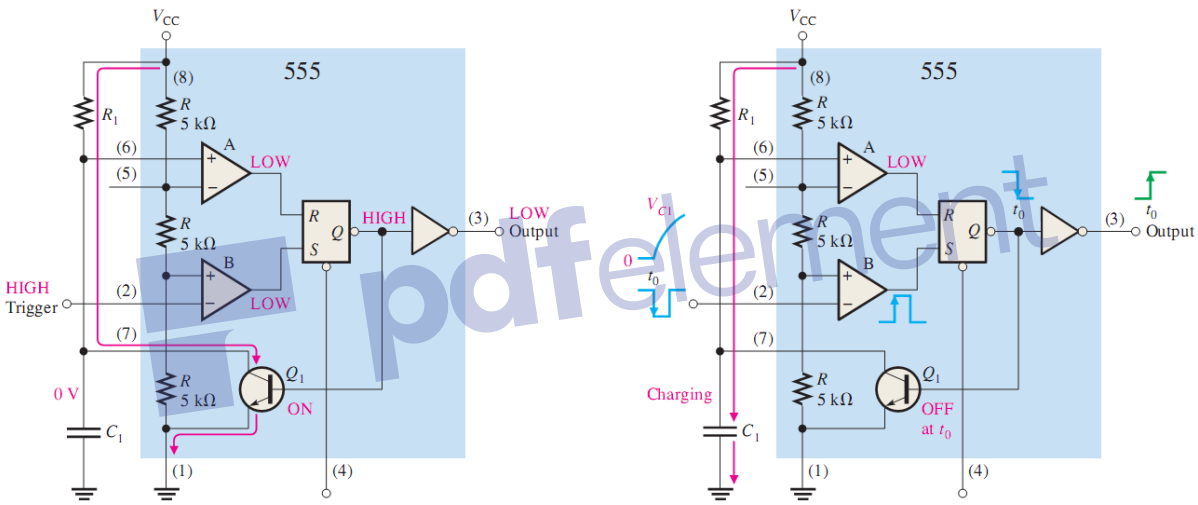
B-The 555 Timer as a monostable

An external resistor and capacitor connected as shown in Figure below are used to set up the 555 timer as a monostable. The pulse width of the output is determined by the time constant of R1 and C1 according to the following formula:

$$t_W = 1.1R_1C_1$$

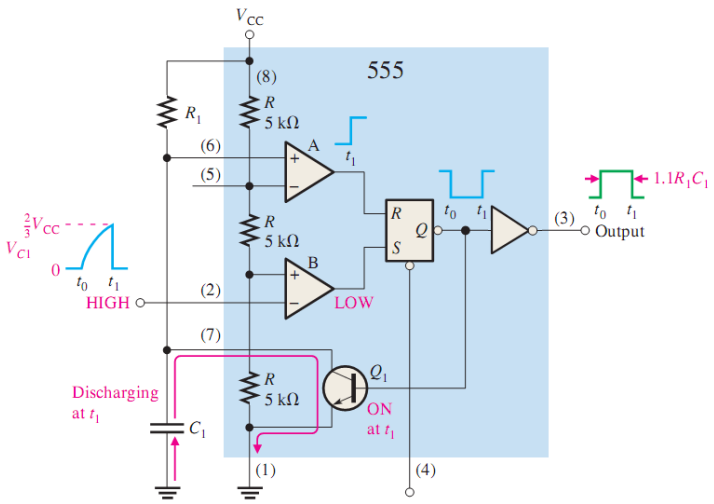
The control voltage input is not used and is connected to a decoupling capacitor C2 to prevent noise from affecting the trigger and threshold levels.

Before a trigger pulse is applied, the output is LOW and the discharge transistor Q1 is on, keeping C1 discharged as shown in Figure below part (a). When a negative-going trigger pulse is applied at t_0, the output goes HIGH and the discharge transistor turns off, allowing capacitor C1 to begin charging through R1 as shown in part (b). When C1 charges to $\frac{1}{3}V_{CC}$, the output goes back LOW at t1 and Q1 turns on immediately, discharging C1 as shown in part (c). As you can see, the charging rate of C1 determines how long the output is HIGH.



(a) Prior to triggering. (The current path is indicated by the red arrow.)

(b) When triggered

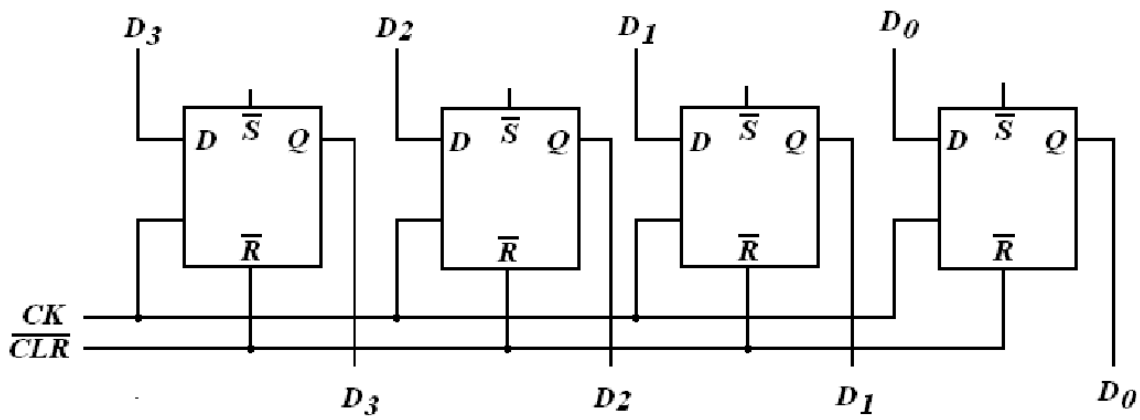


(c) At end of charging interval

5.4Basic flip-flop Applications:

1- Parallel data storage (Simple memory)

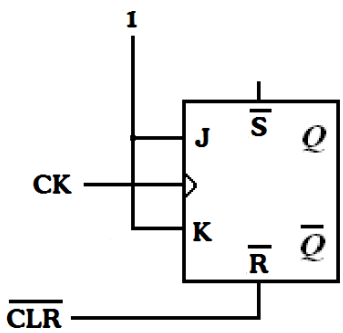
In order to store a 4-bit binary word, we can use the arrangement below:

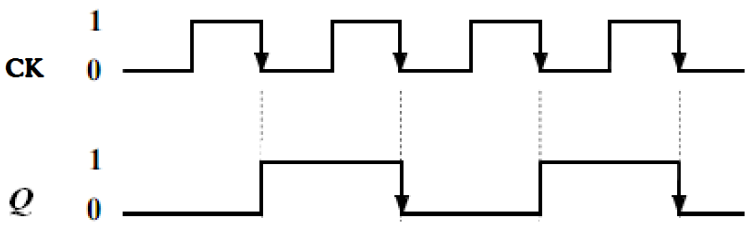


- The 4-bit register (flip-flop) is first cleared by putting $\overline{CLR} = 0$
- When CK is ↑ , the 4-bit word applied on D_0, D_1, D_2, D_3 is stored in the flip-flops.
- The stored data can be read from the outputsQ_0, Q_1, Q_2, Q_3.

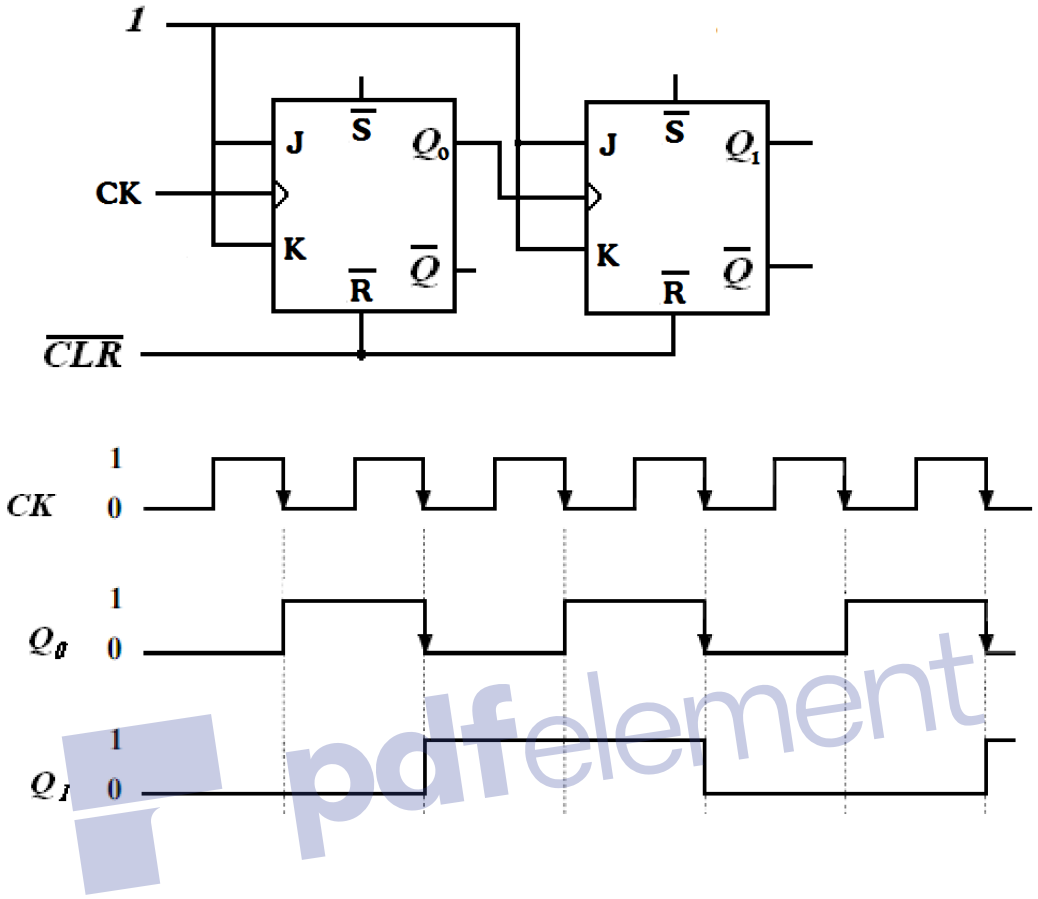
2- Frequency Division

The frequency of the clock signal is divided by 2 at the output (Q) of a JK flip-flop connected in the toggling condition ($J = 1, K = 1$)





Further division of a clock frequency can be achieved by using the output of one flip-flop as the clock input to a second flip-flop, as shown below:



- The frequency of $Q_0 = \frac{1}{2} \times original\ clock\ frequency$
- The frequency of $Q_1 = \frac{1}{4} \times original\ clock\ frequency$

By connecting flip-flops in this way, a frequency is division of 2^n is a achieved, where n is the number of flip-flops. For example three flip-flops divide the clock frequency by $2^3 = 8$; four flip-flops divide the clock frequency by $2^4 = 16$.

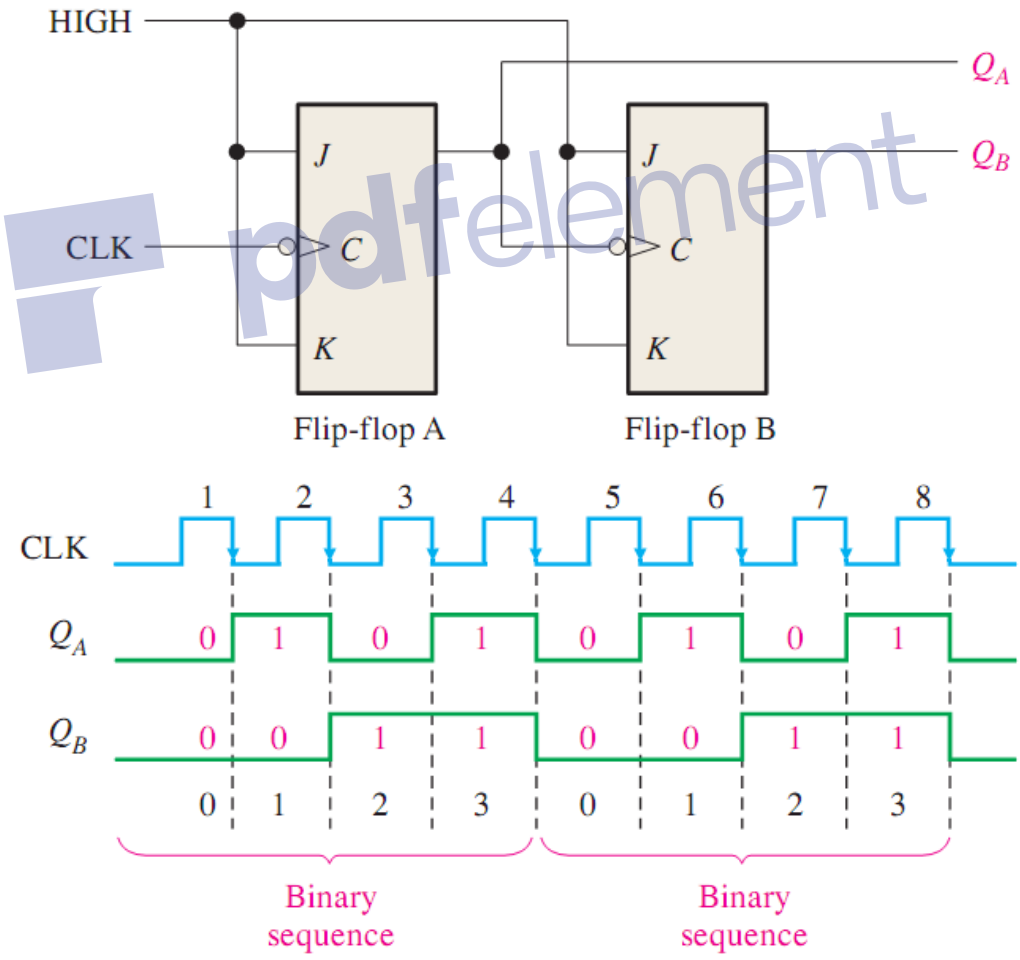
3- Counters

1. Asynchronous counter(Ripple counters)

An asynchronous counter is a sequential logic system in which the clock is applied at one end of the counter. With respect to counter operation, asynchronous means that the flip-flops within the counter are not made to change states at exactly the same time, because the clock pulses are not connected directly to the CK input of each flip-flop in the counter. Asynchronous counters are commonly referred to as ripple counters since the flip-flops are triggered one after the other separated by same delay time. Thus the effect of an input clock pulse "ripples" through the counter to reach the last flip-flop.

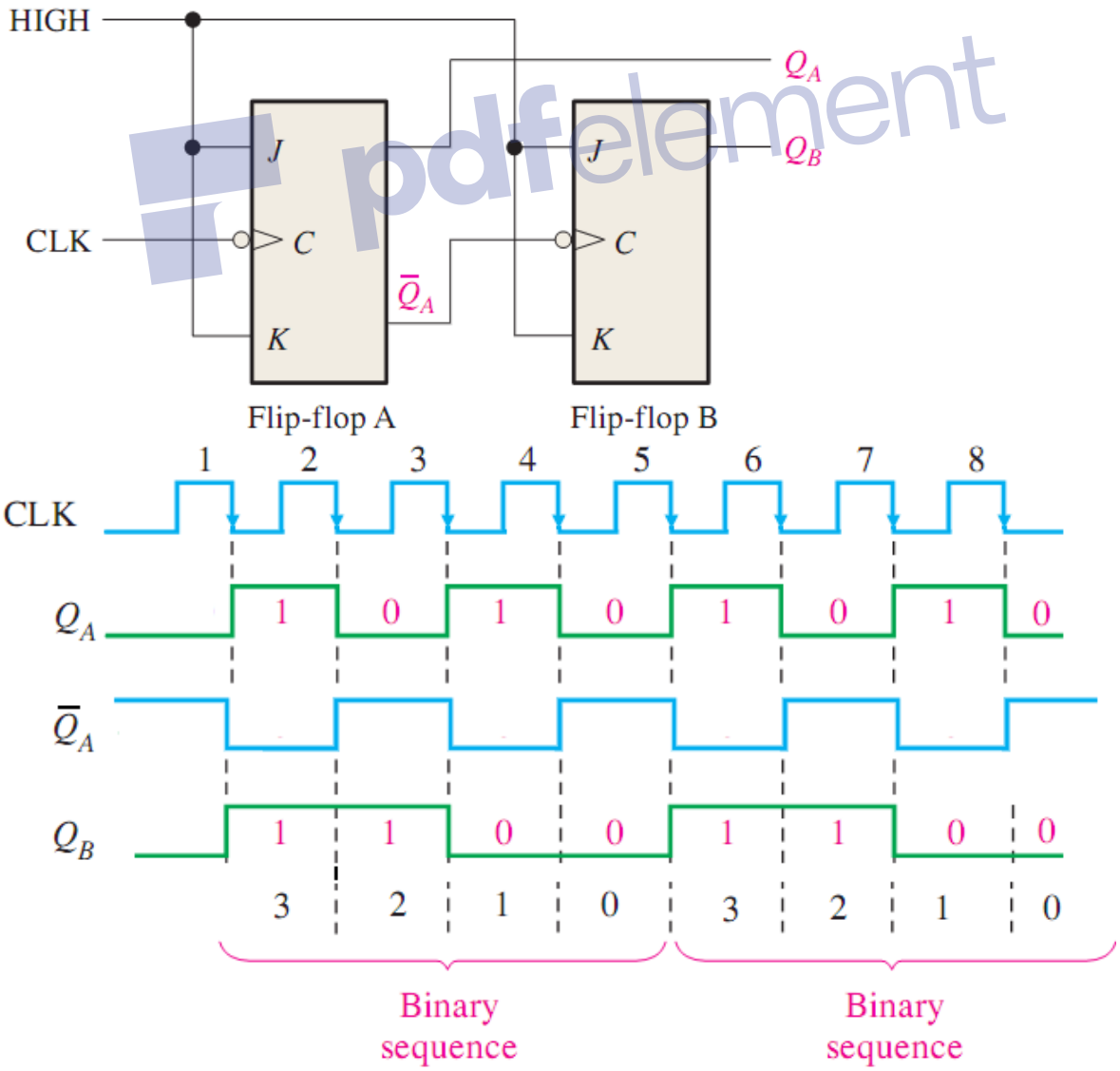
a) Two-Bit Asynchronous Binary Counter.

The following figure shows a 2-bit asynchronous binary counter.



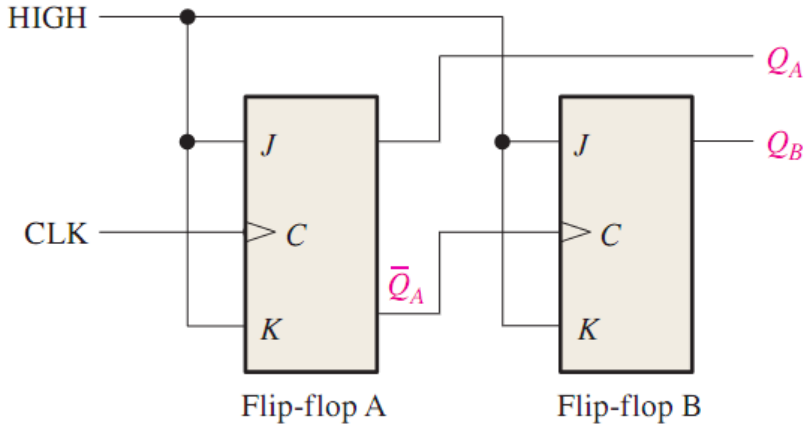
- The JK flip-flops are connected for toggle operation (J=1 , K=1).
- Assuming that the flip-flops are initially RESET.
- When the 1st falling (-ve going) edge of the clock CK comes, Q_A toggles to become HIGH. This has no effect on flip-flop B.
- When the 2nd falling (-ve going) edge of the clock CK comes, Q_A toggles to become LOW. This falling edge of Q_A is connected to the clock input of flip-flop B, therefore, Q_B will toggle to become HIGH, and so on
- The counter will complete a cycle each four clock pulse, and then recycles to the original state.
- The number of states is given by2^n, where n is the number of flip-flops ($2^2 = 4$, 0,1,2,3).

Note that:
The counter described above is an up-counter, i.e., it starts counting from 00-to11 (regardless whether the flip-flops are initially SET or RESET).
If it's required to implement a down-counter, we may connect the output $\bar{Q}_A$ to the clock input of flip-flop B, as shown below:

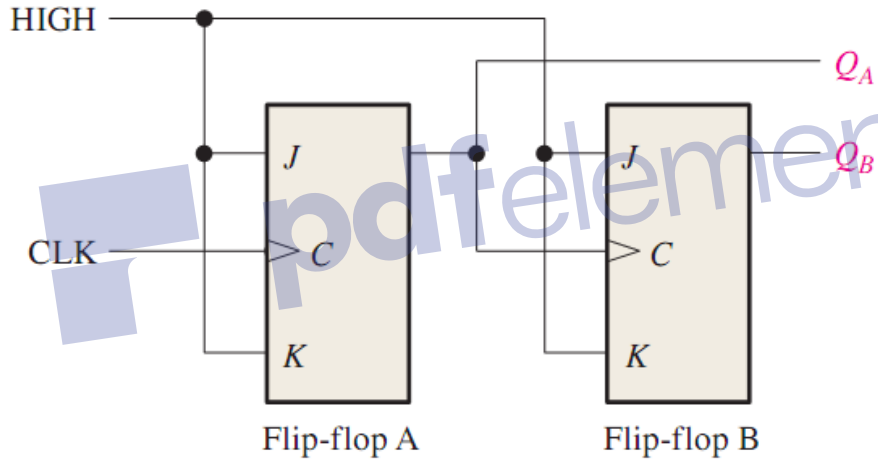


If we wish to implement ripple (asynchronous) counters using positive edge triggered flip-flops, we must note that:

- For an up-counter, the output $\bar{Q}_A$ is connected to the clock input of flip-flop B, as shown in figure below:



- For a down-counter, the output Q_A is connected to the clock input of flip-flop B, as shown in figure below:

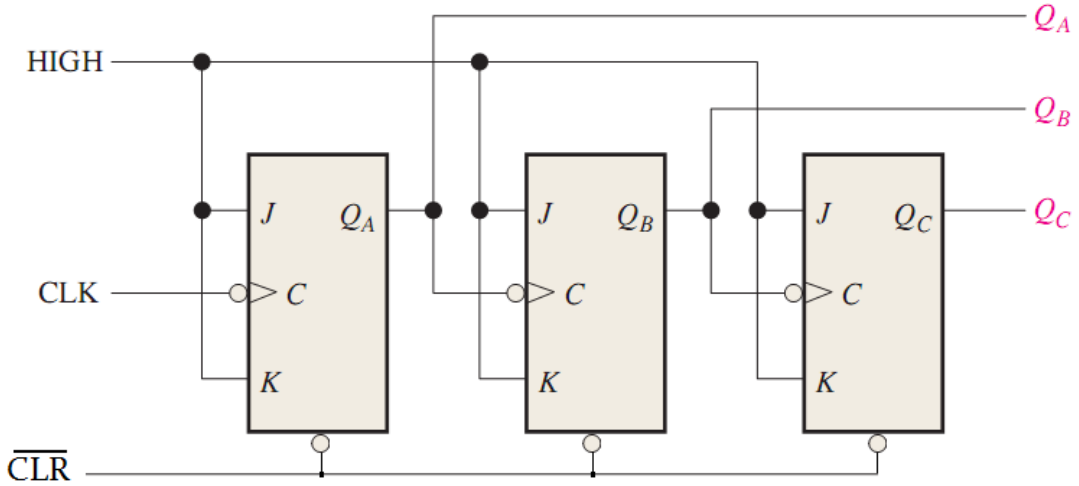


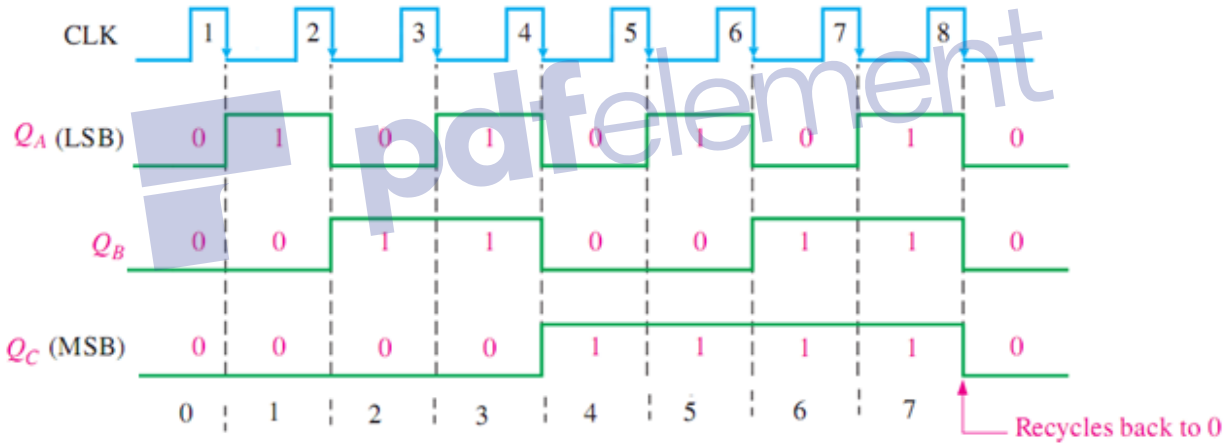
$$\text{upper counter}\begin{cases} +ve\ edge\ clock : \bar{Q}_A \Rightarrow clock\ \ flip-flop\ B \\ -ve\ edge\ clock : Q_A \Rightarrow clock\ \ flip-flop\ B \end{cases}$$

$$\text{down counter}\begin{cases} +ve\ edge\ clock : Q_A \Rightarrow clock\ \ flip-flop\ B \\ -ve\ edge\ clock : \bar{Q}_A \Rightarrow clock\ \ flip-flop\ B \end{cases}$$

b) Three-Bit Asynchronous Binary Counter.

- Here we have three flip-flops, and therefore, eight different states.
- The same basic operation principles of a 2-bit counter are connect here.
- The logic diagram of a 3-bit ripple up counter together with its timing diagram are shown in the figure below:





The output is the number $Q_C Q_B Q_A$

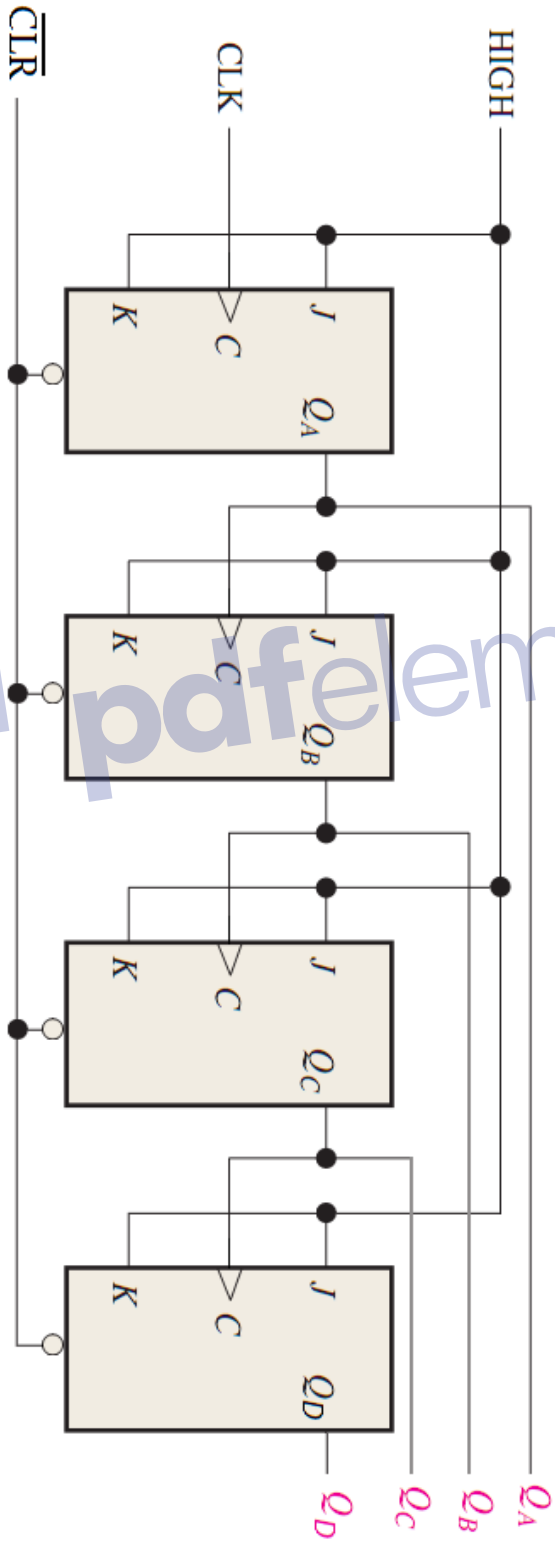
CK pulse	0	1	2	3	4	5	6	7	8	9→
Output	000	001	010	011	100	101	110	111	000	001→

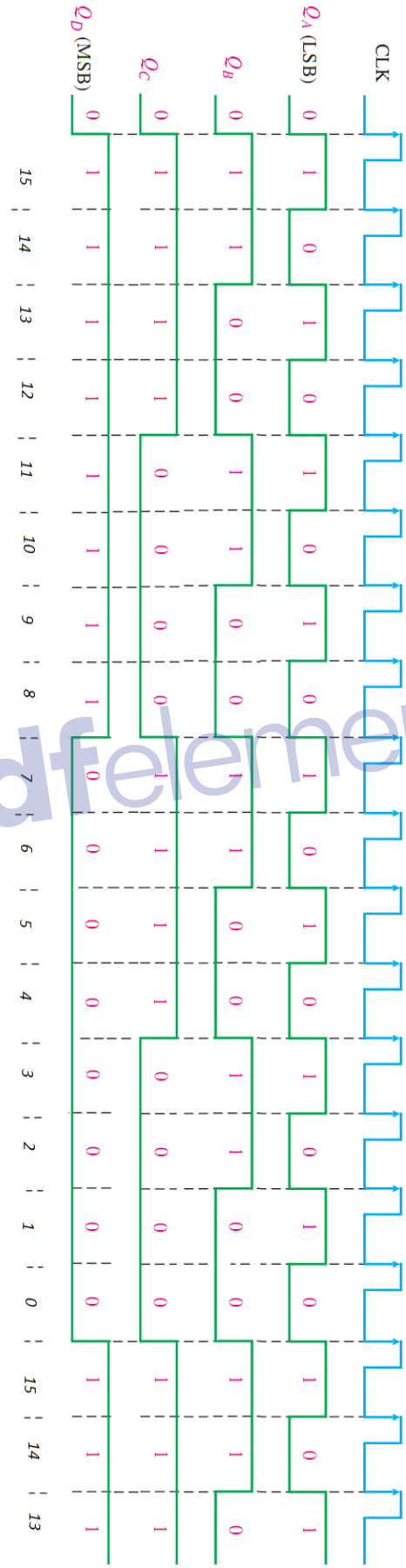
➲ H.W

1. Design a 3-bit asynchronous binary down counter with +ve edge clock.
2. Design a 3-bit asynchronous binary up counter with +ve edge clock.
3. Design a 3-bit asynchronous binary down counter with -ve edge clock.

c) Four-Bit Asynchronous Binary Counter.

The logic timing diagrams for a 4-bit asynchronous down counter using (+ve edge clock) JK flip-flops are shown below:



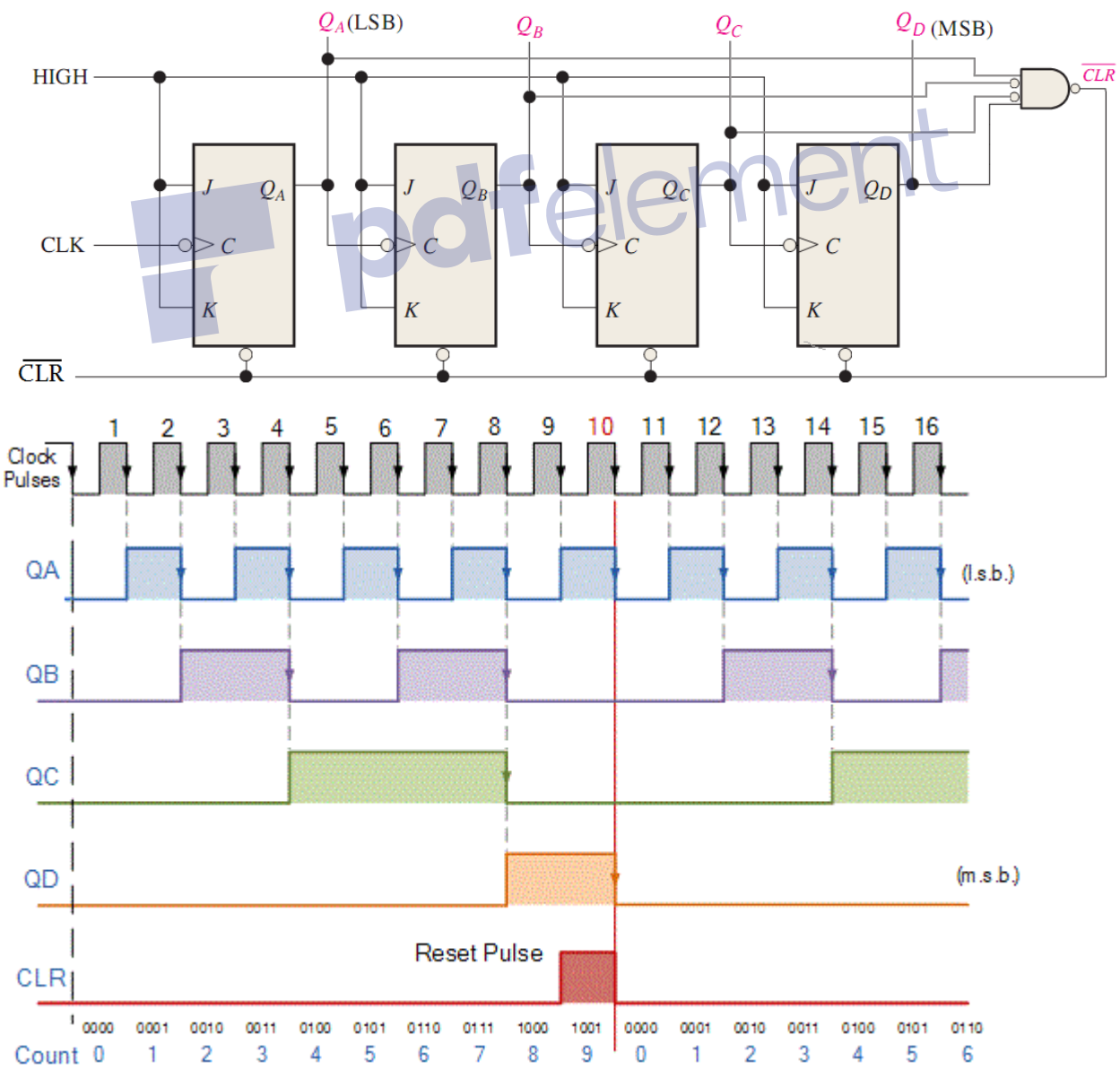


- H.W

1) Design a 4-bit asynchronous binary up counter with -ve edge clock.
2) Design a 4-bit asynchronous binary down counter with -ve edge clock.
3) Design a 4-bit asynchronous binary up counter with +ve edge clock.

d) Asynchronous Decade Counter

- Regular binary counters have 2^n maximum possible the number of flip-flops in the counter.
- Counters can also be designed to have a number of states in their sequence less than2^n. The resulting sequence is called a truncated sequence.
- Counters with ten states n their sequence are called decade counters. A decade counter with a sequence of 0 to 9 (0000 to 1001) is a BCD decade counter because its ten states sequence is the BCD code.
- To do that it is necessary to force the counter to recycle before completing all of its normal state. For example, the BCD decade counter must recycle back to the 0000 state after 1001 state.
- A logic circuit (NAND) must be added such that its output is LOW when the code 1001 appears on the Qs of the counter, in order to bring the counter back to the 0000 state using the $\overline{CLR}$ line as shown in figure below:



2. synchronous counters:

The term synchronous, as applied to counter operations, means that the counter is clocked such that each flip-flop in the counter is triggered at the same time. This is accomplished by connecting the clock line to each of the counter. Unlike asynchronous counters, synchronous counters have different arrangements for the J and K inputs in order to achieve a binary sequence.

A procedure for the design of synchronous counters:

1- Determine the type and the number of flip-flops needed.
2- Write a truth table containing the present state and the next state according to required sequence.
3- Find an expression for each flip-flop input using the k-map according to the type of flip-flops used.
4- Implement these expressions with combinational logic and combine with flip-flops.

Example:

Design a 2-bit synchronous up counter using edge-triggered JK flip-flops (modulus 4 or mod 4 or divide by 4).

Solution

Here we need 2JK flip-flops.

CK	Present state		Next state		Output			
	Q_{in}		$Q_{in}+1$		Q_A		Q_B	
	Q_A	Q_B	Q_A	Q_B	J	K	J	K
0	0	0	0	1	0	x	1	x
1	0	1	1	0	1	x	x	1
2	1	0	1	1	x	0	1	x
3	1	1	0	0	x	1	x	1

Now, the state transition table of a JK flip-flops is:

$Q_{in} \rightarrow Q_{in}+1$	J	K
$0 \rightarrow 0$	0	x
$0 \rightarrow 1$	1	x
$1 \rightarrow 0$	x	1
$1 \rightarrow 1$	x	0

$$0 \rightarrow 0 \begin{cases} \text{Reset } J=0, K=1 \\ \text{No change } J=0, K=0 \end{cases} \Rightarrow J=0, K=x$$

$$0 \rightarrow 1 \begin{cases} \text{Set } J=1, K=0 \\ \text{Toggle } J=1, K=1 \end{cases} \Rightarrow J=1, K=x$$

$$1 \rightarrow 0 \begin{cases} \text{Reset } J=0, K=1 \\ \text{Toggle } J=1, K=1 \end{cases} \Rightarrow J=x, K=1$$

$$1 \rightarrow 1 \begin{cases} \text{Set } J=1, K=0 \\ \text{No change } J=0, K=0 \end{cases} \Rightarrow J=x, K=0$$

1- For FFA, use the k-map to find J_A and K_A :

	$\bar{A}$	A
$\bar{B}$	0	x
B	1	x

$J_A = B = Q_B$

	$\bar{A}$	A
$\bar{B}$	x	0
B	x	1

$K_A = B = Q_B$

2- For FFB, use the k-map to find J_B and K_B :

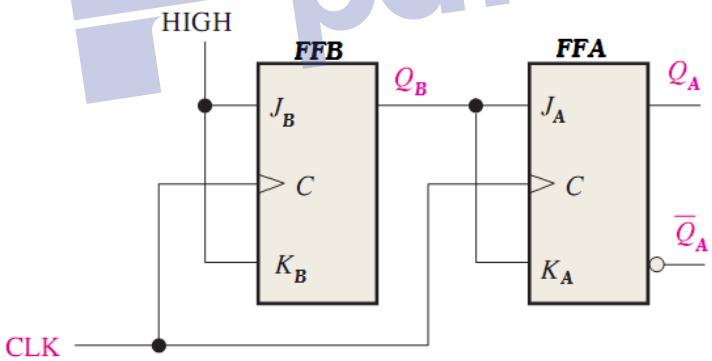
	$\bar{A}$	A
$\bar{B}$	1	1
B	x	x

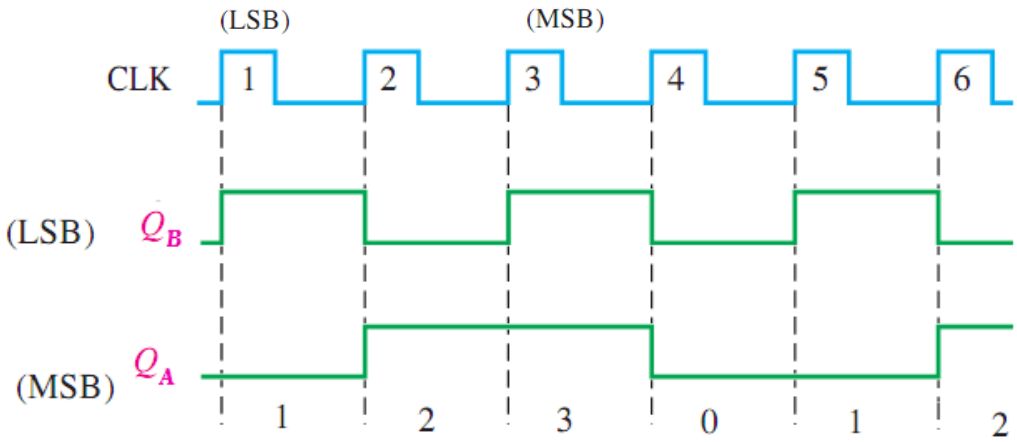
$J_B = 1$

	$\bar{A}$	A
$\bar{B}$	x	x
B	1	1

$K_B = 1$

∴The logic diagram of the counter is:





Note that the design is independent from the way of flip-flop triggering.

Example:
Design a 2-bit synchronous up counter using edge-triggered D flip-flops.
Solution
The state transition table of D flip-flop is:

$Q_{in} \rightarrow Q_{in}+1$	D
$0 \rightarrow 0$	0
$0 \rightarrow 1$	1
$1 \rightarrow 0$	0
$1 \rightarrow 1$	1

CK	Present state Q_{in}		Next state $Q_{in}+1$		Out put	
	Q_A	Q_B	Q_A	Q_B	Q_A	Q_B
0	0	0	0	1	0	1
1	0	1	1	0	1	0
2	1	0	1	1	1	1
3	1	1	0	0	0	0

1- For FFA, use the k-map to find D_A :

	$\bar{A}$	A
$\bar{B}$	0	1
B	1	0

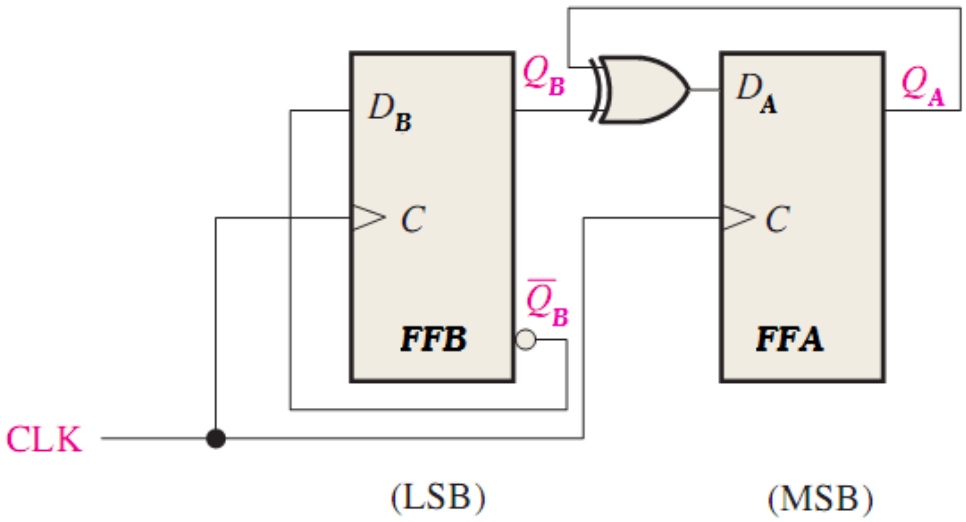
$D_A = \bar{A}B + A\bar{B}$
$= Q_A \oplus Q_B$

2- For FFB, use the k-map to find D_B:

	$\bar{A}$	A
$\bar{B}$	1	1
B	0	0

$D_B = \bar{B} = \bar{Q}_B$

∴The logic diagram of the counter is:



Example:
Design a mod-4(divide by 4) (2-bit) binary synchronous down- counter using JK flip-flops.
Solution
As we saw before, two flip-flops are needed.

CK	Present state		Next state		Input of F.FS			
	Q_{in}		$Q_{in}+1$		Q_A		Q_B	
	Q_A	Q_B	Q_A	Q_B	J	K	J	K
0	0	0	1	1	1	x	1	x
1	0	1	0	0	0	x	x	1
2	1	0	0	1	x	1	1	x
3	1	1	1	0	x	0	x	1

$Q_{in} \rightarrow Q_{in}+1$	J	K
$0 \rightarrow 0$	0	x
$0 \rightarrow 1$	1	x
$1 \rightarrow 0$	x	1
$1 \rightarrow 1$	x	0

1- For FFA, use the k-map to find J_A and K_A :

	$\bar{A}$	A
$\bar{B}$	1	x
B	0	x

$J_A = \bar{B} = \bar{Q}_B$

	$\bar{A}$	A
$\bar{B}$	x	1
B	x	0

$K_A = B = Q_B$

2- For FFB, use the k-map to find J_B and K_B :

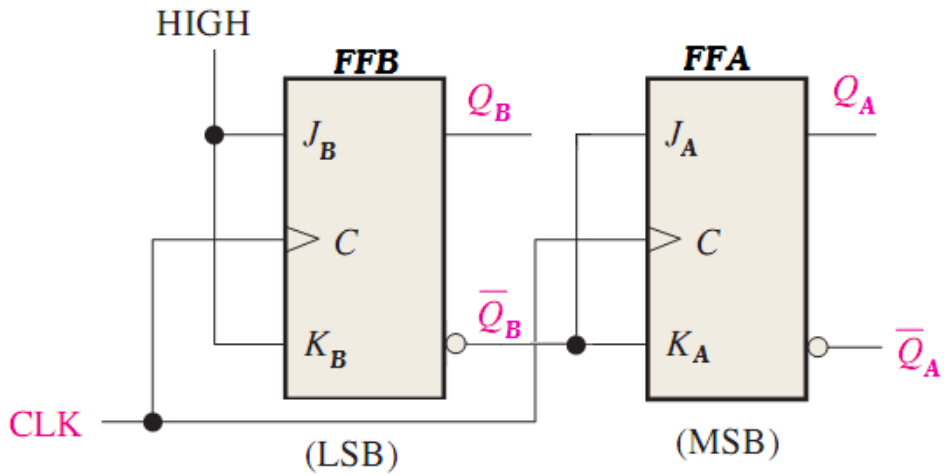
	$\bar{A}$	A
$\bar{B}$	1	1
B	x	x

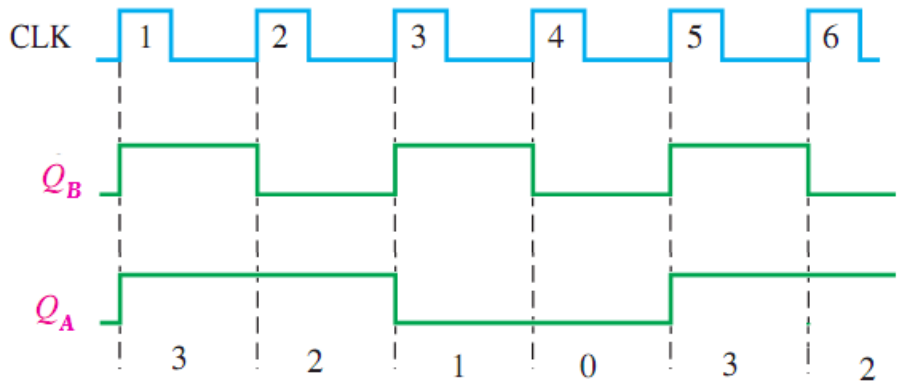
$J_B = 1$

	$\bar{A}$	A
$\bar{B}$	x	x
B	1	1

$K_B = 1$

∴The logic diagram of the counter is:





Example:
Design a 3-bit synchronous up counter using edge-triggered JK flip-flops.
Solution
Here we need 3JK flip-flops.

CK	Present state			Next state			Input of F.FS					
	Q_{in}			$Q_{in}+1$			Q_A		Q_B		Q_C	
	Q_A	Q_B	Q_C	Q_A	Q_B	Q_C	J	K	J	K	J	K
0	0	0	0	0	0	1	0	x	0	x	1	x
1	0	0	1	0	1	0	0	x	1	x	x	1
2	0	1	0	0	1	1	0	x	x	0	1	x
3	0	1	1	1	0	0	1	x	x	1	x	1
4	1	0	0	1	0	1	x	0	0	x	1	x
5	1	0	1	1	1	0	x	0	1	x	x	1
6	1	1	0	1	1	1	x	0	x	0	1	x
7	1	1	1	0	0	0	x	1	x	1	x	1

$Q_{in} \rightarrow Q_{in}+1$	J	K
$0 \rightarrow 0$	0	x
$0 \rightarrow 1$	1	x
$1 \rightarrow 0$	x	1
$1 \rightarrow 1$	x	0

1- For FFA, use the k-map to find J_A and K_A :

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	0	0	x	x
C	0	1	x	x

$J_A = Q_B Q_C$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	x	x	0	0
C	x	x	1	0

$K_A = Q_B Q_C$

2- For FFB, use the k-map to find J_B and K_B :

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	0	x	x	0
C	1	x	x	1

$J_B = Q_C$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	0	x	x	0
C	1	x	x	1

$K_B = Q_C$

3- For FFC, use the k-map to find J_C and K_C :

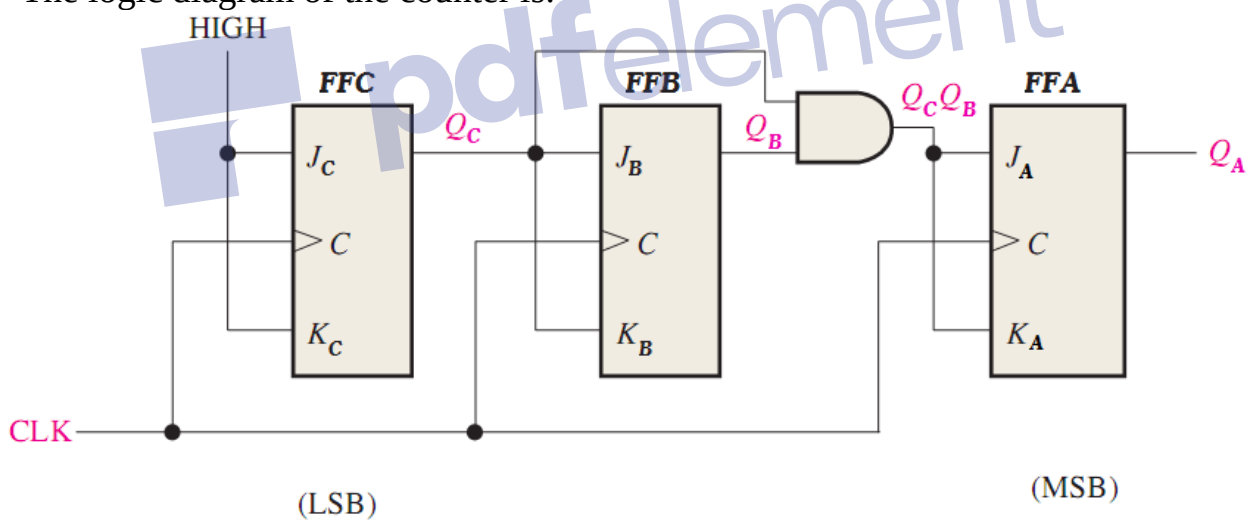
	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	1	1	1	1
C	x	x	x	x

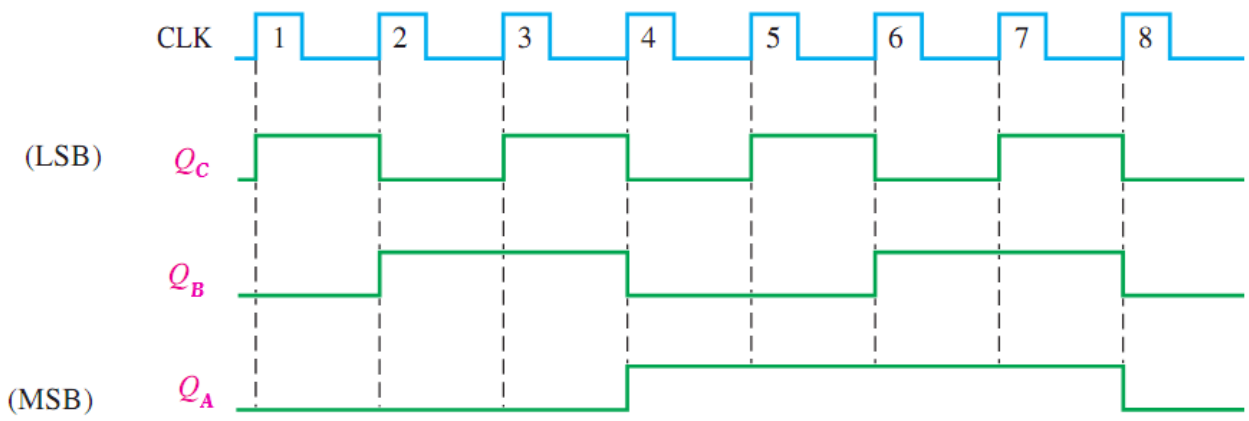
$J_C = 1$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	x	x	x	x
C	1	1	1	1

$K_C = 1$

∴The logic diagram of the counter is:



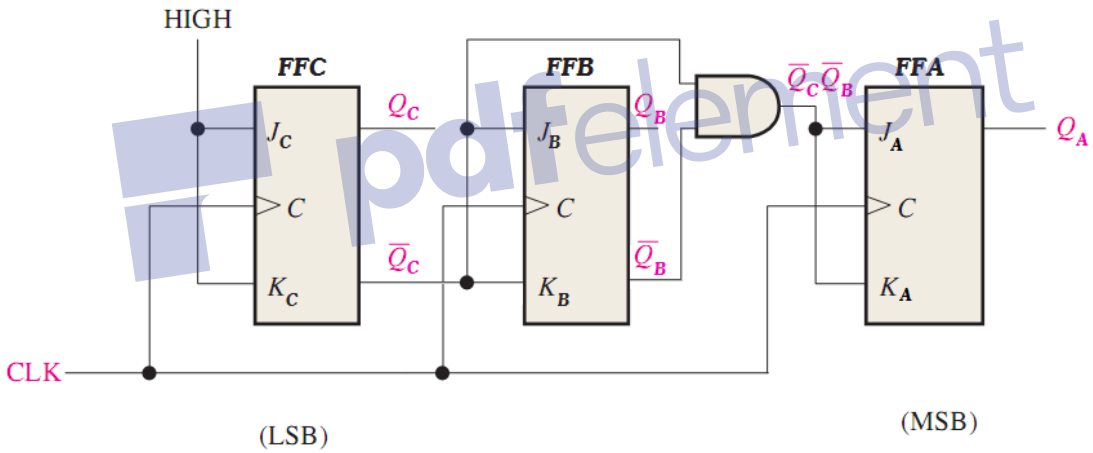


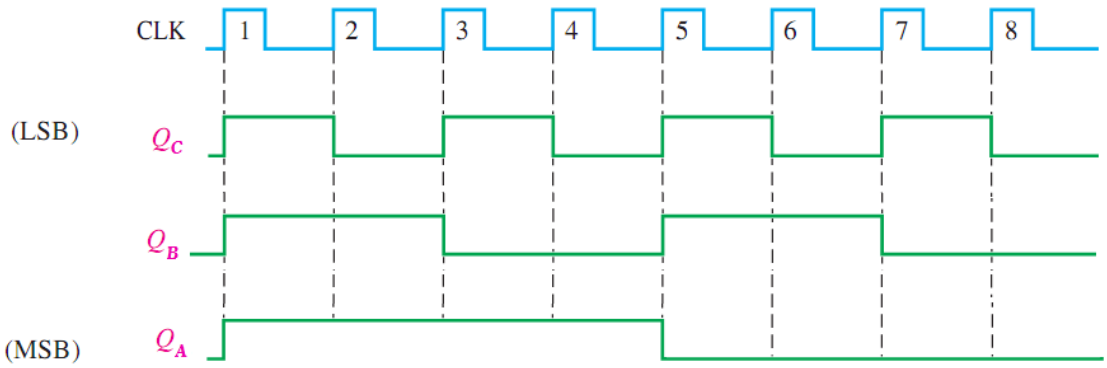
Example:
Design a 3-bit synchronous down counter using edge-triggered JK flip-flops.
Solution
Here we need 3JK flip-flops.

CK	Present state Q_{in}			Next state $Q_{in}+1$		
	Q_A	Q_B	Q_C	Q_A	Q_B	Q_C
0	0	0	0	1	1	1
1	0	0	1	0	0	0
2	0	1	0	0	0	1
3	0	1	1	0	1	0
4	1	0	0	0	1	1
5	1	0	1	1	0	0
6	1	1	0	1	0	1
7	1	1	1	1	1	0

$J_A = \bar{Q}_B\bar{Q}_C,$	$K_A = \bar{Q}_B\bar{Q}_C,$	$J_B = \bar{Q}_C,$	$K_B = \bar{Q}_C,$	$J_C = 1,$	$K_C = 1$

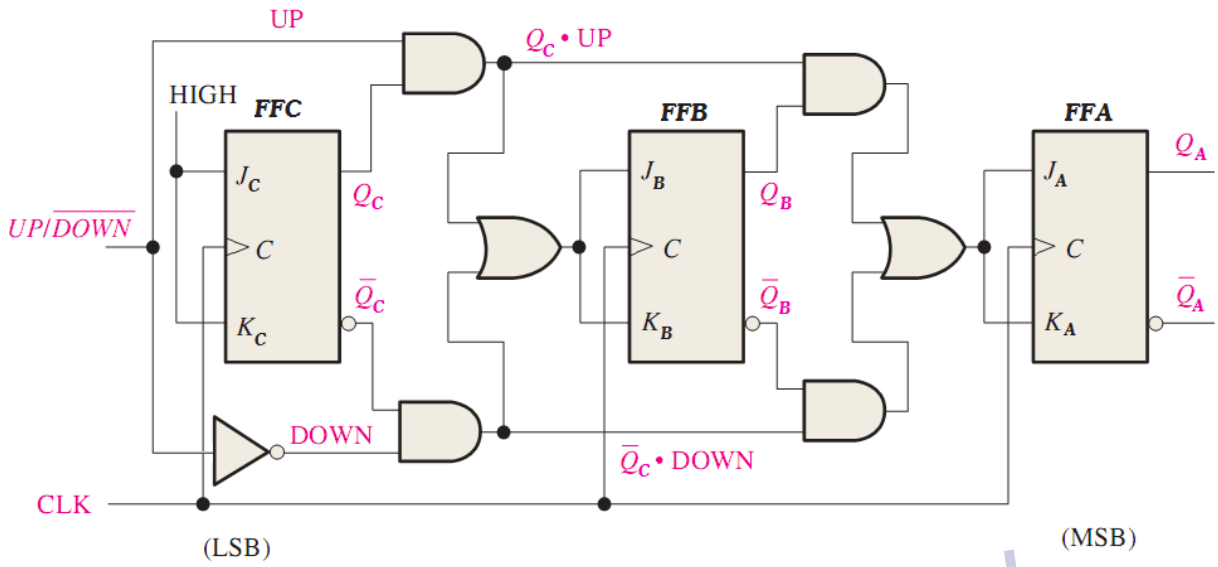
∴The logic diagram of the counter is:





➲ H.W.:
Design a mod 8 synchronous counter using edge triggered D flip-flops:
a- An up-counter.
b- A down counter.

Example:
Draw a logic diagram for an up/down 3-bit synchronous counter using edge-triggered JK flip-flops.
Solution



Example:
Design a BCD (mod-10) synchronous up counter using edge-triggered JK flip-flops.
Solution

CK	Present state				Next state				Input of F.FS							
	Q_{in}				$Q_{in}+1$				Q_A		Q_B		Q_C		Q_D	
	Q_A	Q_B	Q_C	Q_D	Q_A	Q_B	Q_C	Q_D	J	K	J	K	J	K	J	K
0	0	0	0	0	0	0	0	1	0	x	0	x	0	x	1	x
1	0	0	0	1	0	0	1	0	0	x	0	x	1	x	x	1
2	0	0	1	0	0	0	1	1	0	x	0	x	x	0	1	x
3	0	0	1	1	0	1	0	0	0	x	1	x	x	1	x	1
4	0	1	0	0	0	1	0	1	0	x	x	0	0	x	1	x
5	0	1	0	1	0	1	1	0	0	x	x	0	1	x	x	1
6	0	1	1	0	0	1	1	1	0	x	x	0	x	0	1	x
7	0	1	1	1	1	0	0	0	1	x	x	1	x	1	x	1
8	1	0	0	0	1	0	0	1	x	0	x	0	0	x	1	x
9	1	0	0	1	0	0	0	0	x	1	x	0	0	x	x	1

1- For FFA, use the k-map to find J_A and K_A :

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	0	0	x	x
$\bar{C}D$	0	0	x	x
CD	0	1	x	x
$C\bar{D}$	0	0	x	x

$J_A = BCD = Q_B Q_C Q_D$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	x	x	x	0
$\bar{C}D$	x	x	x	1
CD	x	x	x	x
$C\bar{D}$	x	x	x	x

$K_A = D = Q_D$

2- For FFB, use the k-map to find J_B and K_B :

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	0	x	x	x
$\bar{C}D$	0	x	x	x
CD	1	x	x	x
$C\bar{D}$	0	x	x	x

$J_B = CD = Q_C Q_D$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	x	0	x	0
$\bar{C}D$	x	0	x	0
CD	x	1	x	x
$C\bar{D}$	x	0	x	x

$K_B = CD = Q_C Q_D$

3- For FFC, use the k-map to find J_C and K_C :

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	0	0	x	0
$\bar{C}D$	1	1	x	0
CD	x	x	x	x
$C\bar{D}$	x	x	x	x

$J_C = \bar{A}D = \bar{Q}_A Q_D$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	x	x	x	x
$\bar{C}D$	x	x	x	x
CD	1	1	x	x
$C\bar{D}$	0	0	x	x

$K_C = D = Q_D$

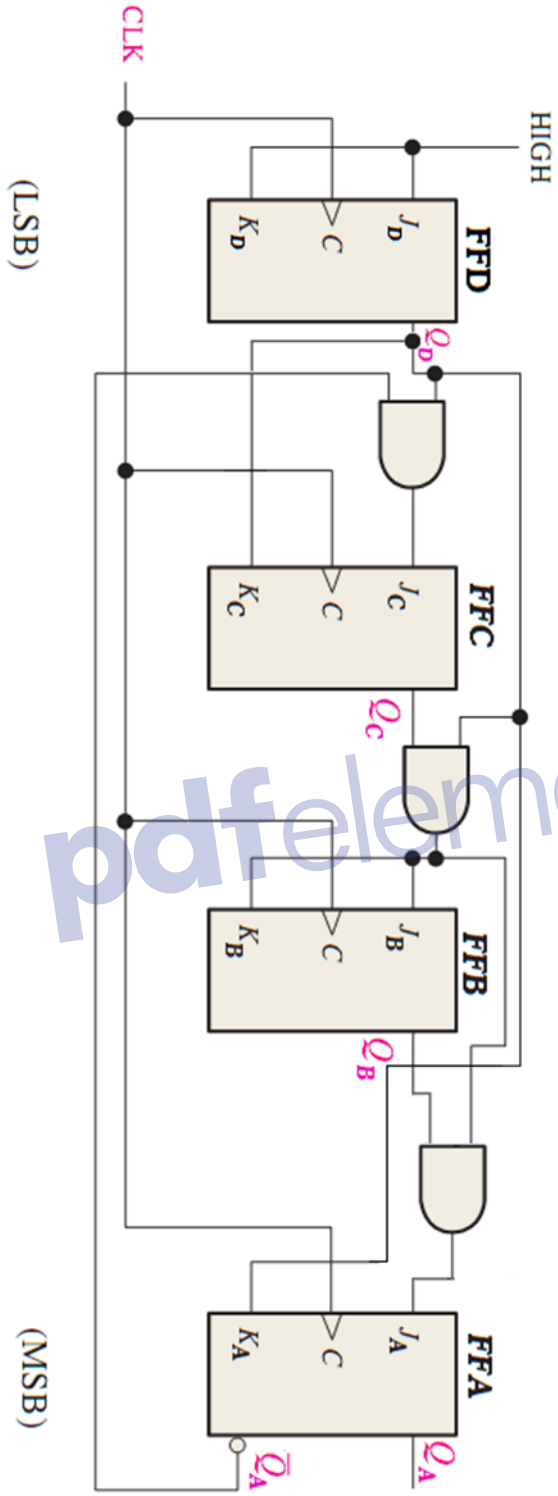
4- For FFD, use the k-map to find J_D and K_D :

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	1	1	x	1
$\bar{C}D$	x	x	x	x
CD	x	x	x	x
$C\bar{D}$	1	1	x	x

$J_D = 1$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	x	x	x	x
$\bar{C}D$	1	1	x	1
CD	1	1	x	x
$C\bar{D}$	x	x	x	x

$K_D = 1$



Example:
Design a synchronous 3-bit up counter with a Gray code sequence using JK flip-flops.
Solution
Here we need 3JK flip-flops.

CK	Present state Q_{in}			Next state $Q_{in}+1$			Input of F.FS					
							Q_A		Q_B		Q_C	
	Q_A	Q_B	Q_C	Q_A	Q_B	Q_C	J	K	J	K	J	K
0	0	0	0	0	0	1	0	x	0	x	1	x
1	0	0	1	0	1	1	0	x	1	x	x	0
2	0	1	1	0	1	0	0	x	x	0	x	1
3	0	1	0	1	1	0	1	x	x	0	0	x
4	1	1	0	1	1	1	x	0	x	0	1	x
5	1	1	1	1	0	1	x	0	x	1	x	0
6	1	0	1	1	0	0	x	0	0	x	x	1
7	1	0	0	0	0	0	x	1	0	x	0	x

1- For FFA, use the k-map to find J_A and K_A :

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	0	0	x	X
C	0	1	x	X

$$J_A = Q_B Q_C$$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	x	x	0	0
C	x	x	1	0

$$K_A = Q_B Q_C$$

2- For FFB, use the k-map to find J_B and K_B :

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	0	x	0	X
C	1	x	0	X

$$J_B = \bar{Q}_A Q_C$$

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	x	0	x	0
C	x	0	x	1

$$K_B = Q_A Q_C$$

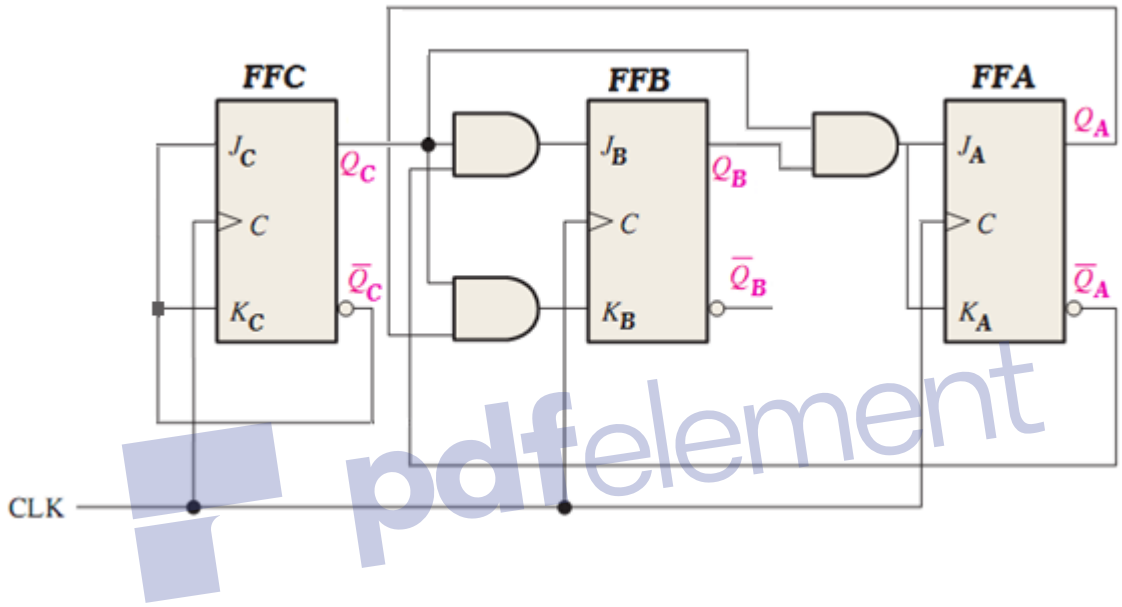
3- For FFC, use the k-map to find J_C and K_C :

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	1	x	x	1
C	x	0	0	x

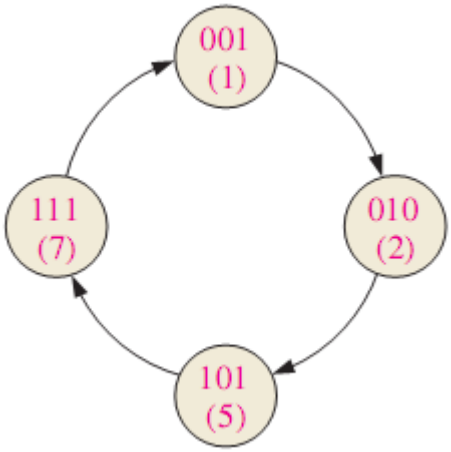
	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	x	1	1	x
C	0	x	x	0

$$J_C = \bar{Q}_C \qquad K_C = \bar{Q}_C$$

The implementation of the counter is shown in Figure below:



Example:
Design a counter with the irregular binary count sequence shown in the state diagram of Figure below. Use D flip-flops.



Solution
We have only four states, a 3-bit counter is require 3 flip-flops to implement this sequence because the maximum binary count is seven.

CK	Present state Q_{in}			Next state $Q_{in}+1$			Input of F.FS		
	Q_A	Q_B	Q_C	Q_A	Q_B	Q_C	Q_A	Q_B	Q_C
1	0	0	1	0	1	0	0	1	0
2	0	1	0	1	0	1	1	0	1
5	1	0	1	1	1	1	1	1	1
7	1	1	1	0	0	1	0	0	1

1- For FFA, use the k-map to find D_A :

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	x	1	x	x
C	0	x	0	1

$$D_A = \bar{C} + A\bar{B} = \bar{Q}_C + Q_A\bar{Q}_B$$

2- For FFB, use the k-map to find D_B:

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	x	0	x	x
C	1	x	0	1

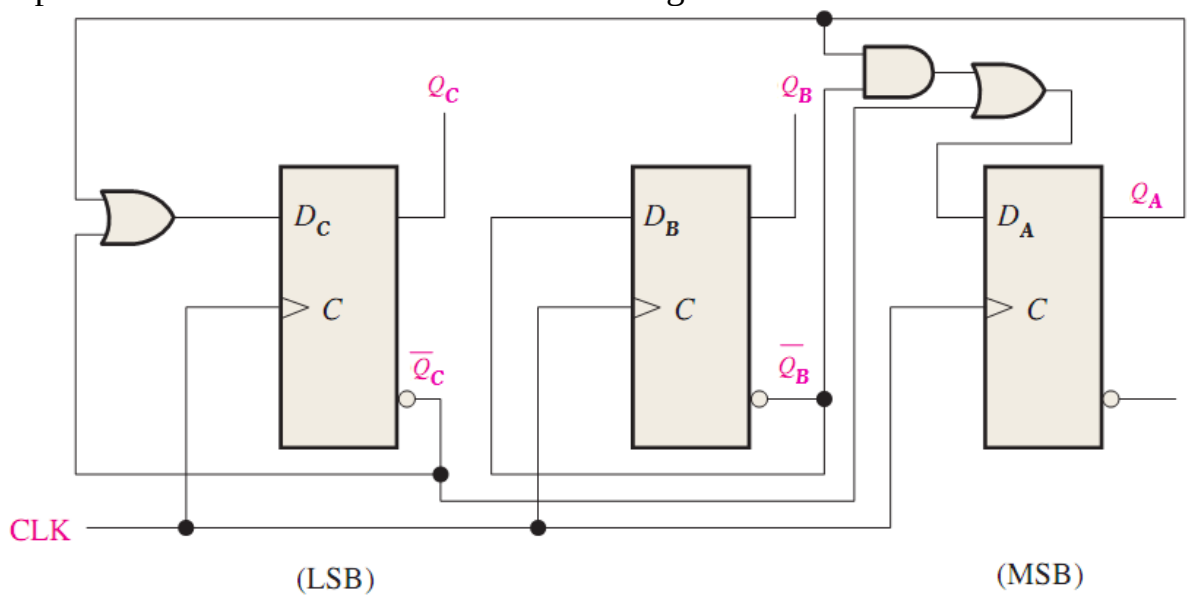
$$D_B = \bar{B} = \bar{Q}_B$$

3- For FFC, use the k-map to find D_C:

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	x	1	x	x
C	0	x	1	1

$$D_C = \bar{C} + A = \bar{Q}_C + Q_A$$

The implementation of the counter is shown in Figure below:



Example:
Design an up/down 3-bit synchronous counter using T flip-flops.
Solution
Here we need 3T flip-flops.

M	Present state Q_{in}			Next state $Q_{in}+1$			Input of F.FS		
							Q_A	Q_B	Q_C
	Q_A	Q_B	Q_C	Q_A	Q_B	Q_C	T	T	T
0	0	0	0	0	0	1	0	0	1
0	0	0	1	0	1	0	0	1	1
0	0	1	0	0	1	1	0	0	1
0	0	1	1	1	0	0	1	1	1
0	1	0	0	1	0	1	0	0	1
0	1	0	1	1	1	0	0	1	1
0	1	1	0	1	1	1	0	0	1
0	1	1	1	0	0	0	1	1	1
1	0	0	0	1	1	1	1	1	1
1	0	0	1	0	0	0	0	0	1
1	0	1	0	0	0	1	0	1	1
1	0	1	1	0	1	0	0	0	1
1	1	0	0	0	1	1	1	1	1
1	1	0	1	1	0	0	0	0	1
1	1	1	0	1	0	1	0	1	1
1	1	1	1	1	1	0	0	0	1

The state transition table of T flip-flop is:

$Q_{in} \rightarrow Q_{in}+1$	T
$0 \rightarrow 0$	0
$0 \rightarrow 1$	1
$1 \rightarrow 0$	1
$1 \rightarrow 1$	0

1- For FFA, use the k-map to find T_A

	$\bar{M}\bar{A}$	$\bar{M}A$	MA	$M\bar{A}$
$\bar{B}\bar{C}$	0	0	1	1
$\bar{B}C$	0	0	0	0
BC	1	1	0	0
$B\bar{C}$	0	0	0	0

$$T_A = M\bar{B}\bar{C} + \bar{M}BC = M\bar{Q}_B\bar{Q}_C + \bar{M}Q_BQ_C$$

2- For FFB, use the k-map to find T_B:

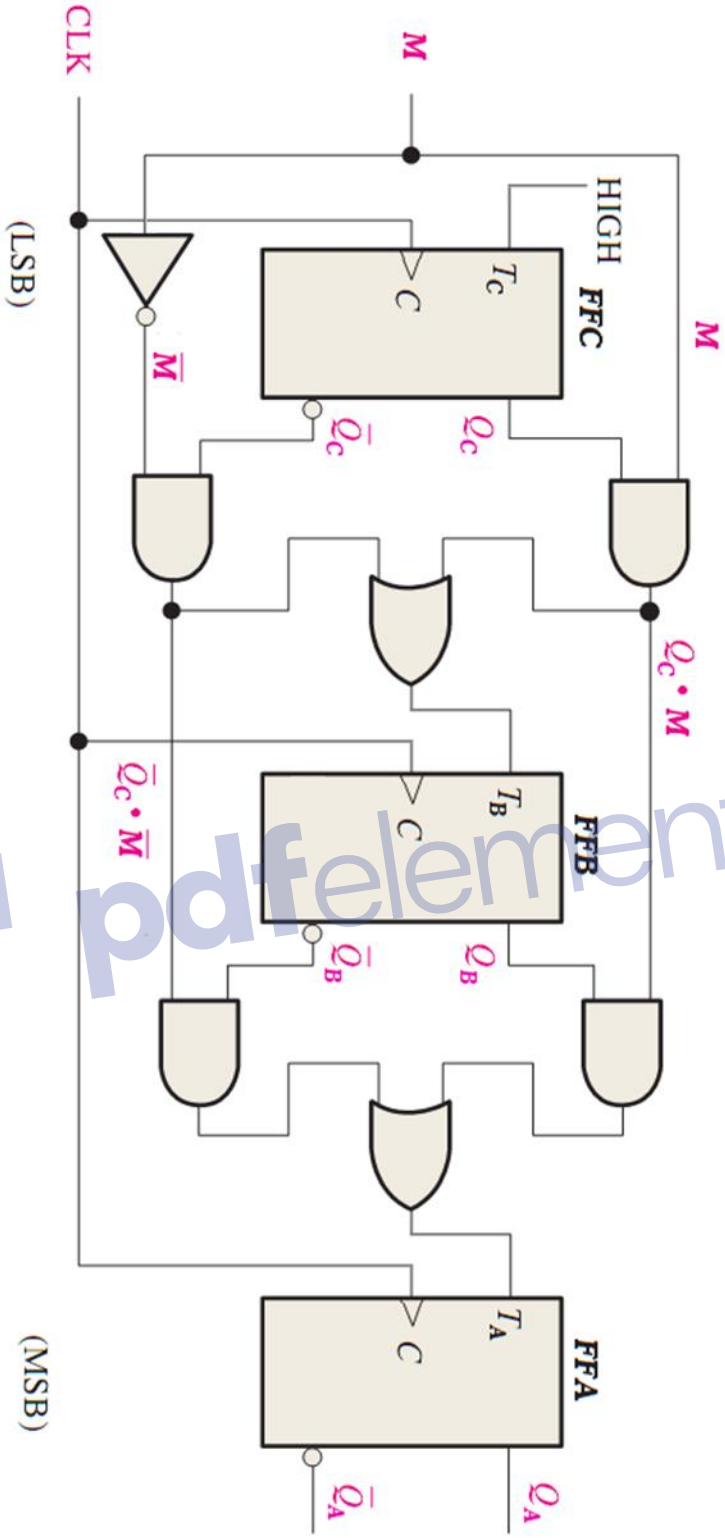
	$\bar{M}\bar{A}$	$\bar{M}A$	MA	$M\bar{A}$
$\bar{B}\bar{C}$	0	0	1	1
$\bar{B}C$	1	1	0	0
BC	1	1	0	0
$B\bar{C}$	0	0	1	1

$$T_B = M\bar{C} + \bar{M}C = M\bar{Q}_C + \bar{M}Q_C$$

3- For FFC, use the k-map to find T_C:

	$\bar{M}\bar{A}$	$\bar{M}A$	MA	$M\bar{A}$
$\bar{B}\bar{C}$	1	1	1	1
$\bar{B}C$	1	1	1	1
BC	1	1	1	1
$B\bar{C}$	1	1	1	1

$$T_C = 1$$



4- Shift register

Shift registers are a type of sequential logic circuit, mainly for storage of digital data. They are a group of flip-flops connected in a chain so that the output from one flip-flop becomes the input of the next flip-flop. Most of the registers possess no characteristic internal sequence of states. All flip-flops are driven by a common clock, and all are set or reset simultaneously.
The basic types of shift registers are studied, such as Serial In - Serial Out, Serial In - Parallel Out, Parallel In – Serial Out, Parallel In - Parallel Out, and bidirectional shift registers.

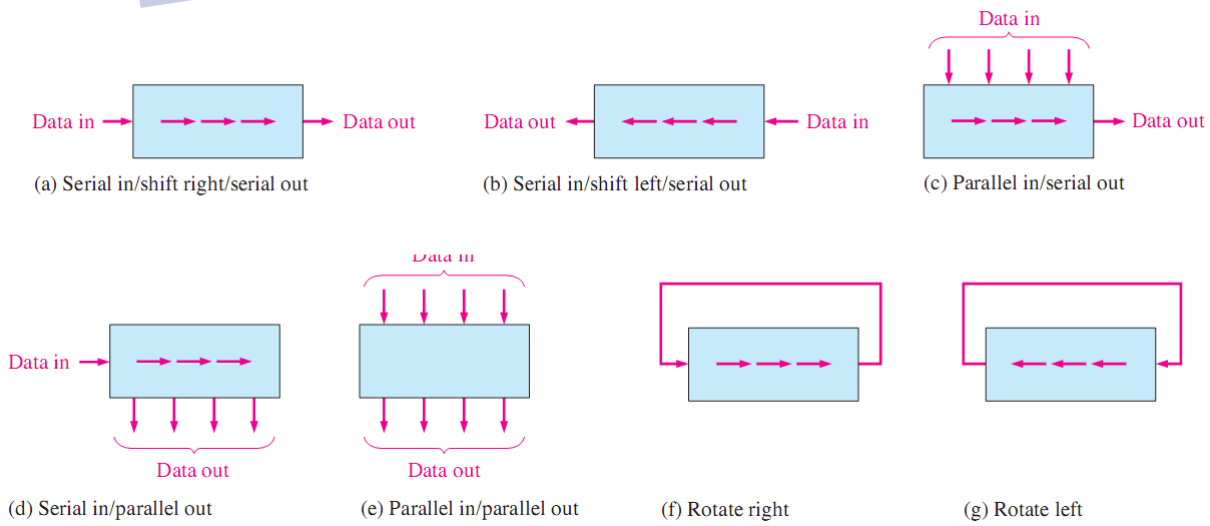
Register:

- A set of n flip-flops.
- Each flip-flop stores one bit.
- Two basic functions: data storage and data movement.

Shift Register:

- A register that allows each of the flip-flops to pass the stored information to its adjacent neighbour.

The figure below shows the basic data movement in shift registers.

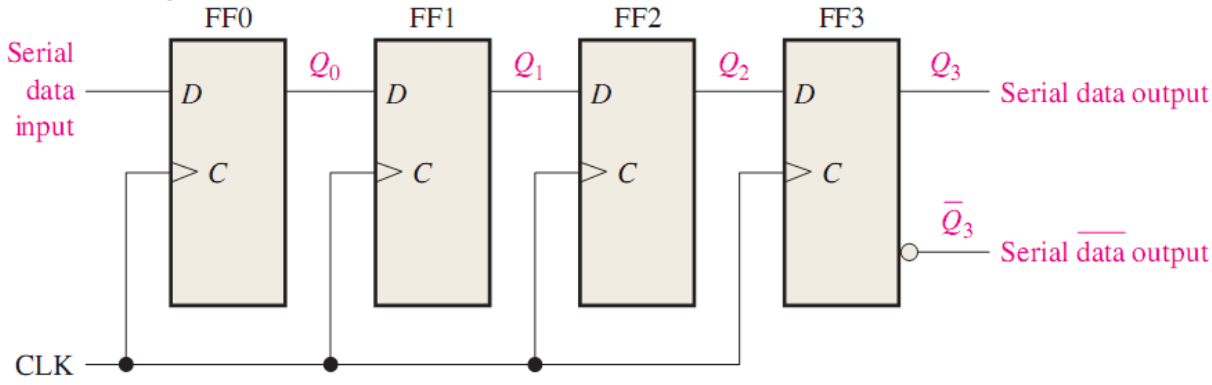


Basic data movement in shift registers. (Four bits are used for illustration. The bits move in the direction of the arrows.)

A. *Serial In - Serial Out Shift Registers*

The serial in/serial out shift register accepts data serially – that is, one bit at a time on a single line. It produces the stored information on its output also in serial form.

A basic four-bit shift register can be constructed using four D flip-flops, as shown in figure below:



The operation of the circuit is as follows:

- The register is first cleared, forcing all four outputs to zero.
- The input data is then applied sequentially to the D input of the first flip-flop on the left (FF0).
- During each clock pulse, one bit is transmitted from left to right.

Assume a data word to be 1010. The least significant bit of the data has to be shifted through the register from FF0 to FF3. In order to get the data out of the register, they must be shifted out serially. This can be done destructively or non-destructively. For destructive readout, the original data is lost and at the end of the read cycle, all flip-flops are reset to zero.

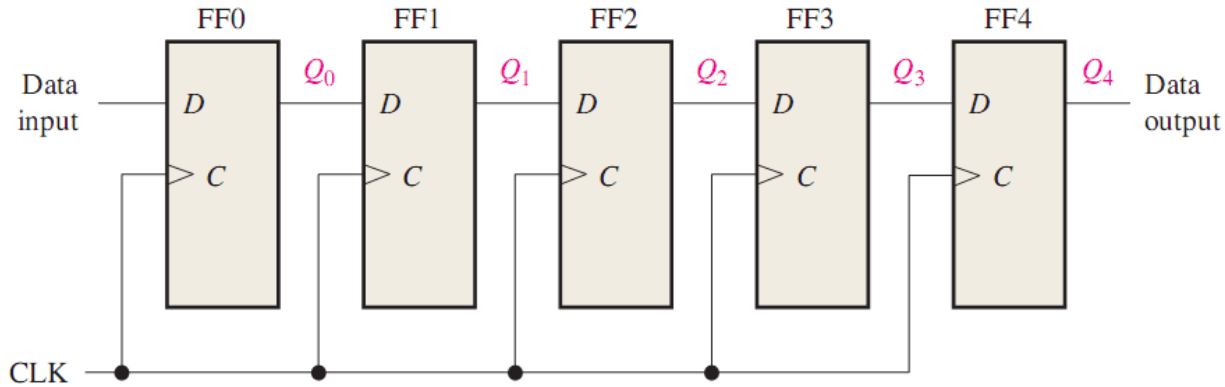
The following table shows shifting a 4-bit code into the shift register.

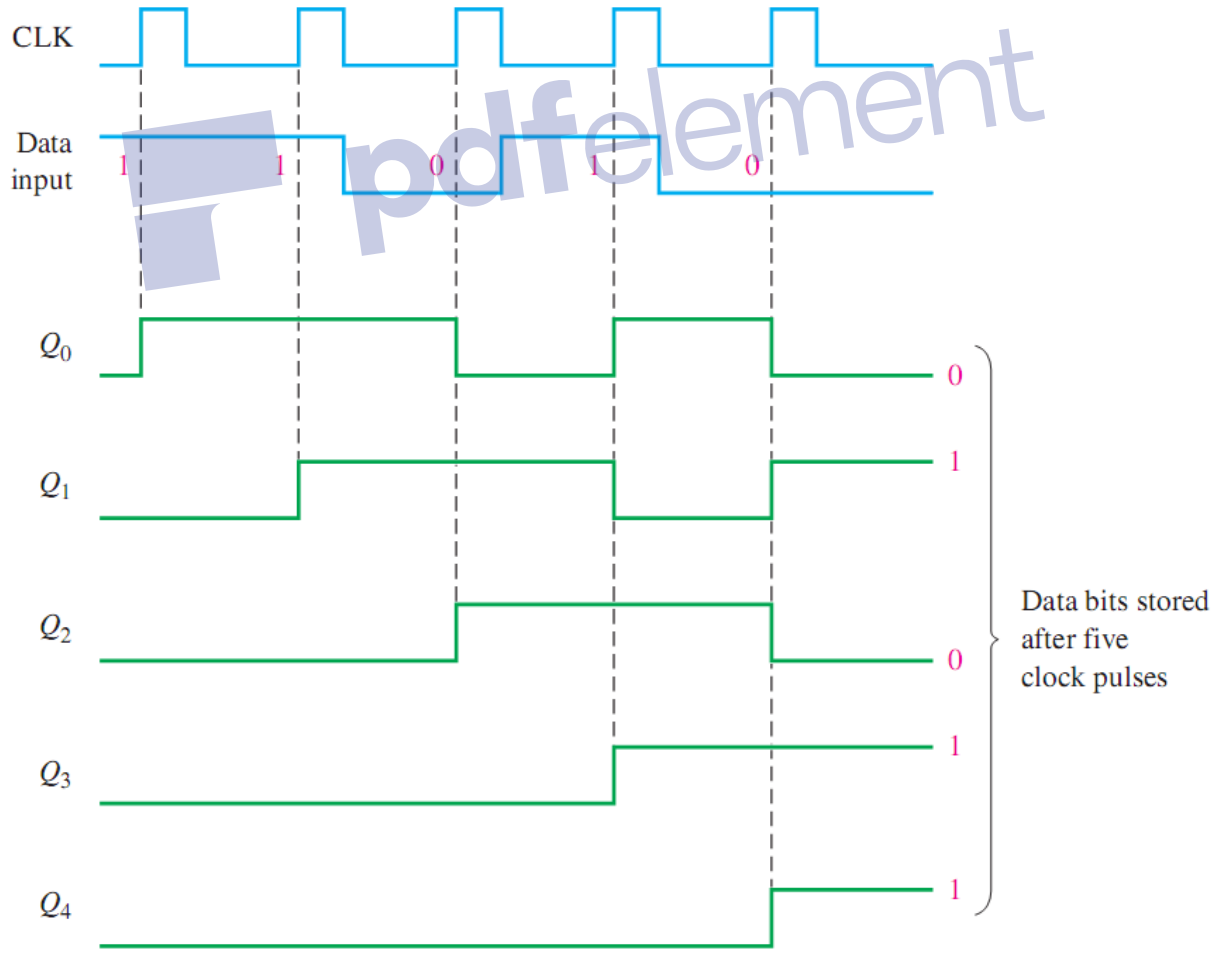
CLK	FF0(Q_0)	FF1(Q_1)	FF2(Q_2)	FF3(Q_3)
Initial	**0**	**0**	**0**	**0**
1	**0**	**0**	**0**	**0**
2	**1**	**0**	**0**	**0**
3	**0**	**1**	**0**	**0**
4	**1**	**0**	**1**	**0**

The following table shows shifting a 4-bit code out of the shift register

CLK	FF0(Q_0)	FF1(Q_1)	FF2(Q_2)	FF3(Q_3)
Initial	**1**	**0**	**1**	**0**
5	**0**	**1**	**0**	**1**
6	**0**	**0**	**1**	**0**
7	**0**	**0**	**0**	**1**
8	**0**	**0**	**0**	**0**

Example:
Show the states of the 5-bit register in Figure below for the specified data input and clock waveforms. Assume that the register is initially cleared (all 0s).
Solution
The first data bit (1) is entered into the register on the first clock pulse and then shifted from left to right as the remaining bits are entered and shifted. The register contains Q4Q3Q2Q1Q0 = 11010 after five clock pulses.

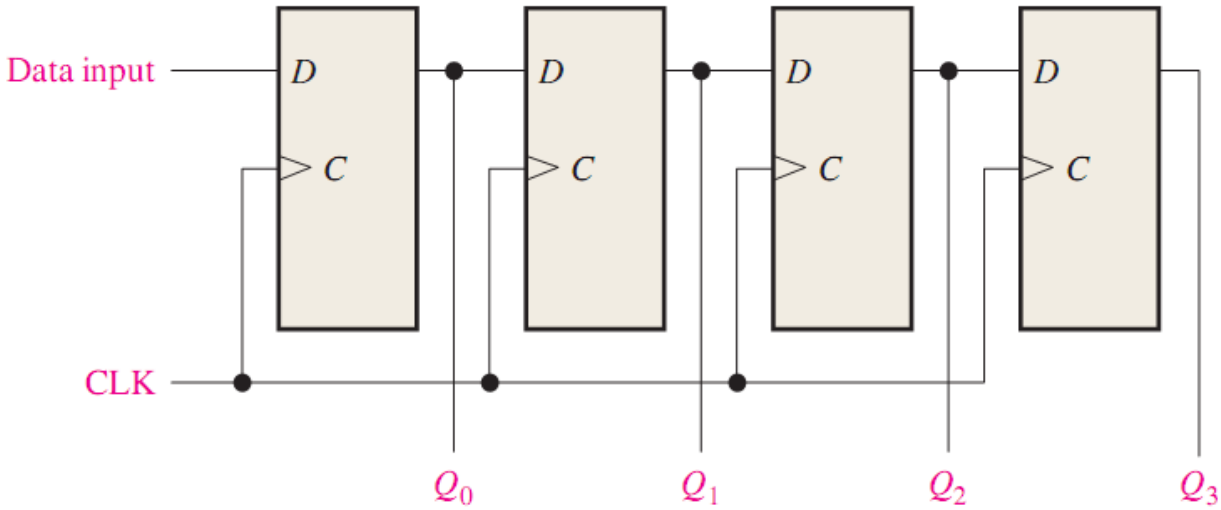


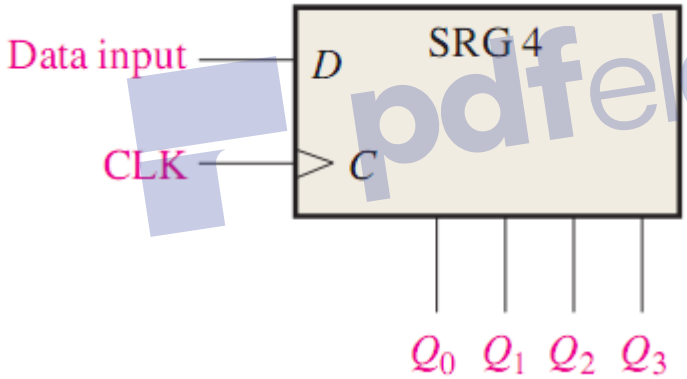


B. *Serial In/Parallel out Shift Registers*

For this kind of register, data bits are entered serially in the same manner as discussed in the last section. The difference is the way in which the data bits are taken out of the register. Once the data are stored, each bit appears on its respective output line, and all bits are available simultaneously.

A construction of a four-bit serial in - parallel out register is shown below.

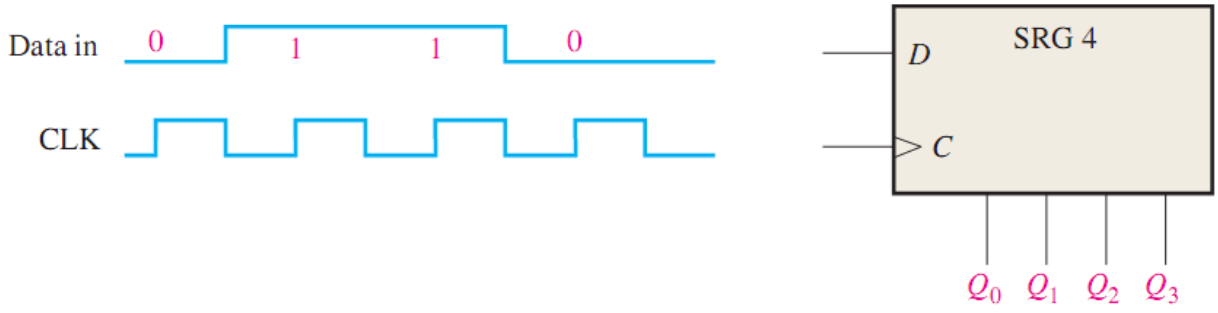




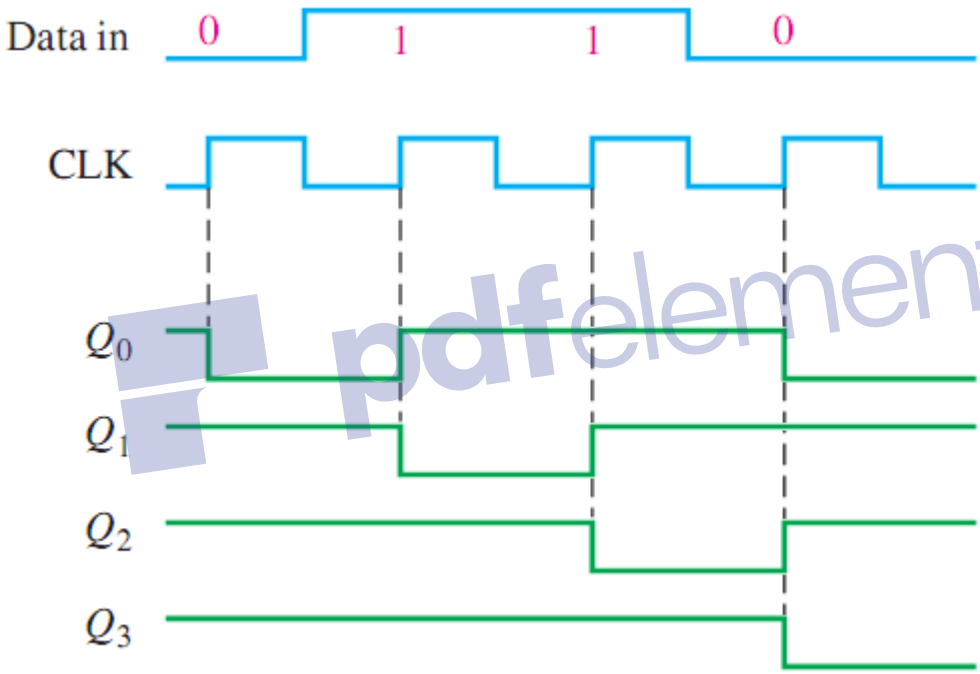
In the table below, we can see how the four-bit binary number 1001 is shifted to the Q outputs of the register.

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

Example:
Show the states of the 4-bit register (SRG 4) for the data input and clock waveforms in Figure below. The register initially contains all 1s.



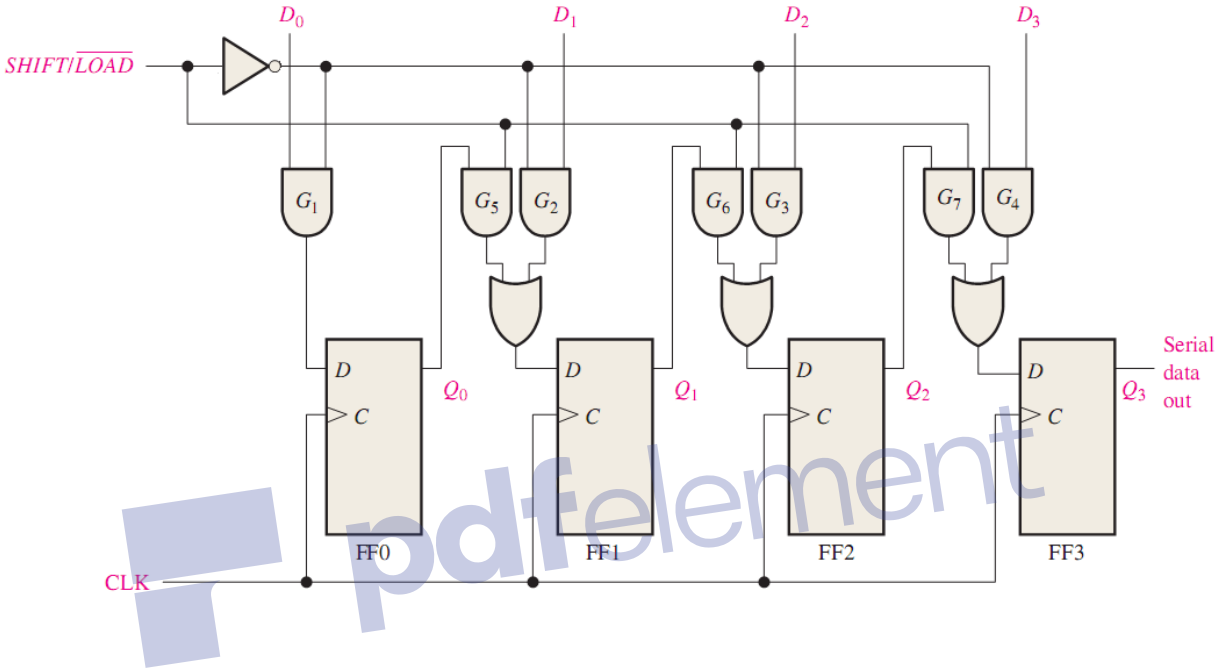
Solution

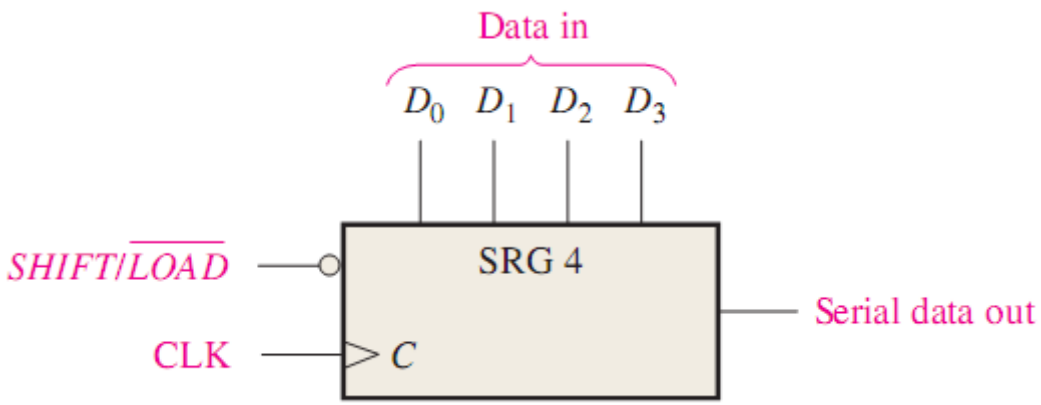


The register contains 0110 after four clock pulses.

C. *Parallel In - Serial out Shift Registers*

For a register with parallel data inputs, the bits are entered simultaneously into their respective stages on parallel lines rather than on a bit-by-bit basis on one line as with serial data inputs. The serial output is the same as in serial in/serial out shift registers, once the data are completely stored in the register. For parallel data, multiple bits are transferred at one time.
A logic diagram for a four-bit parallel in - serial out shift register is shown below.



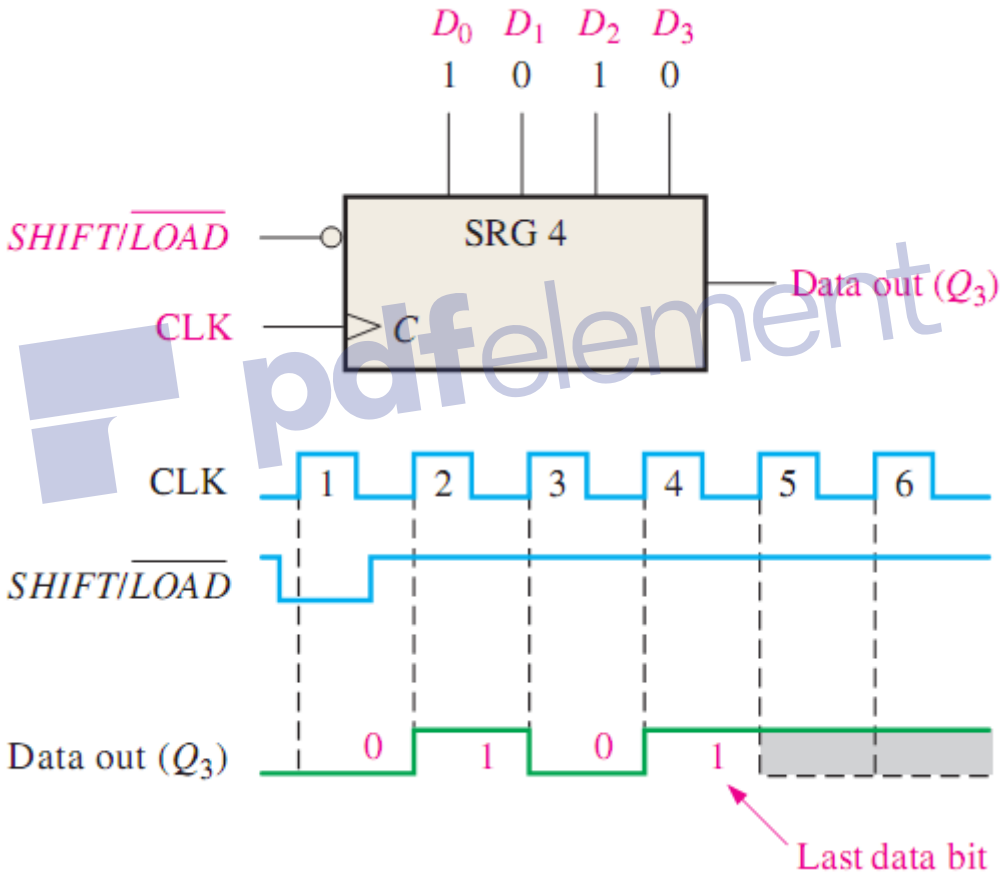


Logic symbol

The circuit uses D flip-flops and gates for entering data to the register. D0, D1, D2 and D3 are the parallel inputs, where D0 is the MSB and D3 is the LSB. To load data in, the mode control line is taken to LOW ($\overline{LOAD}$) and the data is clocked in. The data can be shifted when the mode control line is HIGH as SHIFT is active high. The register performs right shift operation on the application of a clock pulse.

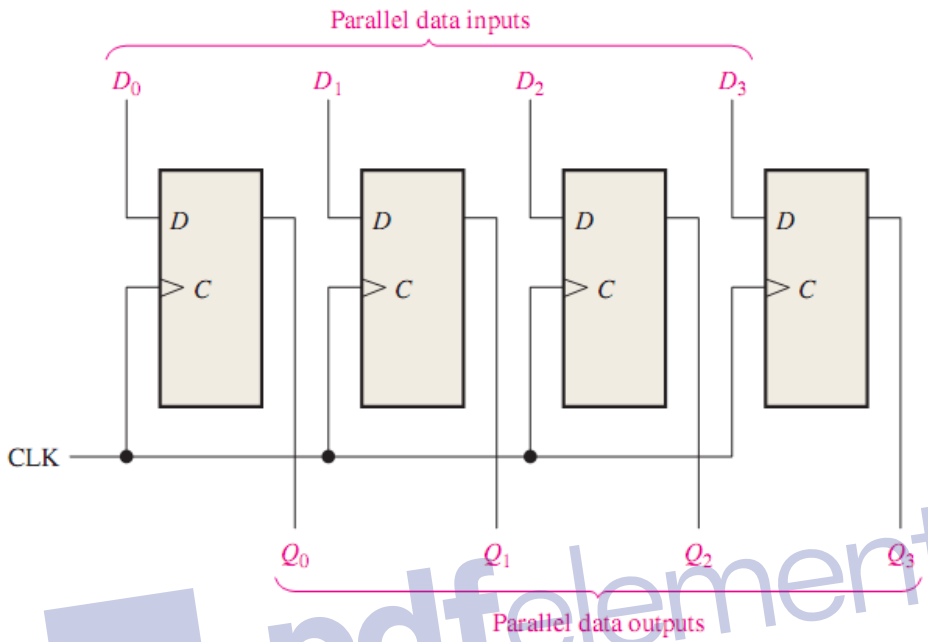
Example:
Show the data-output waveform for a 4-bit register with the parallel input data and the clock and SHIFT/LOAD waveforms given in Figure below.

Solution
On clock pulse 1, the parallel data (D0D1D2D3 = 1010) are loaded into the register, making Q3 a 0. On clock pulse 2 the 1 from Q2 is shifted onto Q3; on clock pulse 3 the 0 is shifted onto Q3; on clock pulse 4 the last data bit (1) is shifted onto Q3; and on clock pulse 5, all data bits have been shifted out, and only 1s remain in the register (assuming the D0 input remains a 1).



D. *Parallel In/Parallel out Shift Registers*

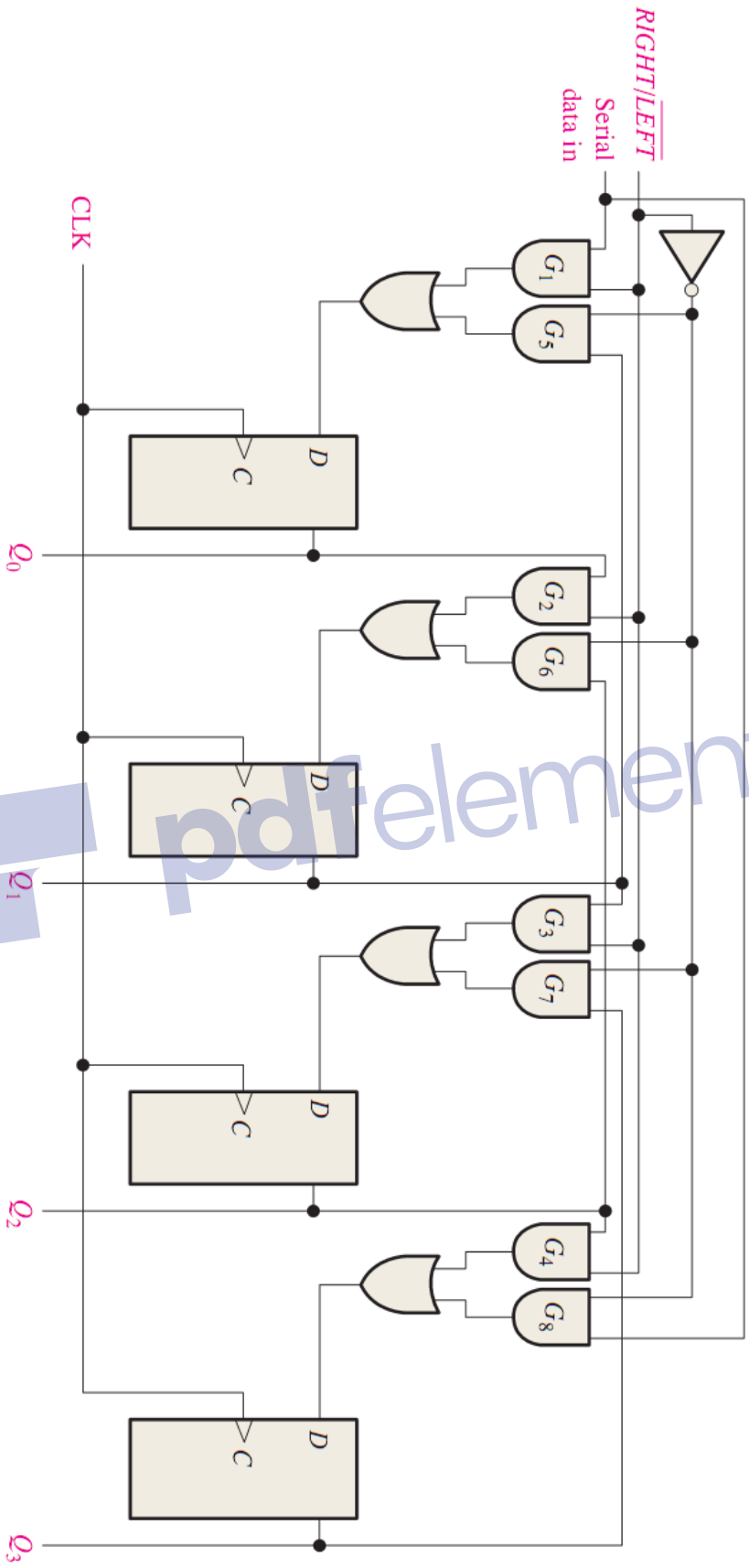
For parallel in - parallel out shift registers, all data bits appear on the parallel outputs immediately following the simultaneous entry of the data bits. The following circuit is a four-bit parallel in - parallel out shift register constructed by D flip-flops.



The D's are the parallel inputs and the Q's are the parallel outputs. Once the register is clocked, all the data at the D inputs appear at the corresponding Q outputs simultaneously.

E. *Bidirectional Shift Registers*

The registers discussed so far involved only right shift operations. Each right shift operation has the effect of successively dividing the binary number by two. If the operation is reversed (left shift), this has the effect of multiplying the number by two. With suitable gating arrangement a serial shift register can perform both operations. A bidirectional, or reversible, shift register is one in which the data can be shift either left or right. A four-bit bidirectional shift register using D flip-flops is shown below.

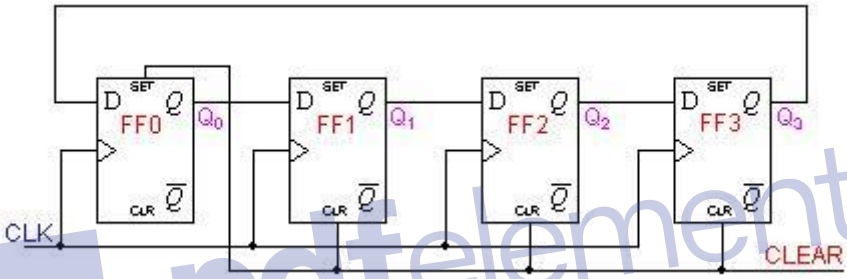


Shift Register Counters

Two of the most common types of shift register counters are introduced here: the Ring counter and the Johnson counter. They are basically shift registers with the serial outputs connected back to the serial inputs in order to produce particular sequences. These registers are classified as counters because they exhibit a specified sequence of states.

1- Ring Counters :

A ring counter is basically a circulating shift register in which the output of the most significant stage is fed back to the input of the least significant stage. The following is a 4-bit ring counter constructed from D flip-flops. The output of each stage is shifted into the next stage on the positive edge of a clock pulse. If the CLEAR signal is high, all the flip-flops except the first one FF0 are reset to 0. FF0 is preset to 1 instead.

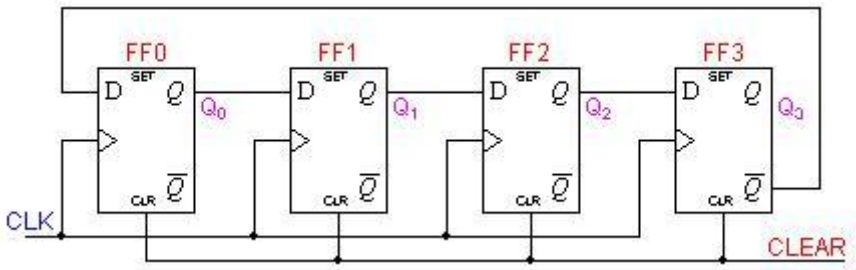


Since the count sequence has 4 distinct states, the counter can be considered as a mod-4 counter. Only 4 of the maximum 16 states are used, making ring counters very inefficient in terms of state usage. But the major advantage of a ring counter over a binary counter is that it is self-decoding. No extra decoding circuit is needed to determine what state the counter is in.

Clock pulse	Q_3	Q_2	Q_1	Q_0
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	1	0	0	0

2- *Johnson Counters*

Johnson counters are a variation of standard ring counters, with the inverted output of the last stage fed back to the input of the first stage. They are also known as twisted ring counters. An *n*-stage Johnson counter yields a count sequence of length *2n*, so it may be considered to be amod-*2n* counter. The circuit below shows a 4-bit Johnson counter.



The state sequence for the counter is given in the table below as well as the animation on the left.

Clock pulse	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	1	1	1
4	1	1	1	1
5	1	1	1	0
6	1	1	0	0
7	1	0	0	0

Again, the apparent disadvantage of this counter is that the maximum available states are not fully utilized. Only eight of the sixteen states are being used.