Digital Electronic and Circuits EC403

Prepared by: Ms.Palasri Dhar, Mr. Shatadru Biswas

MODULE I

Introduction:

- Analog signals-The electrical signals that have any value over a rangeof time

Example: A sinusoidal signal.

- Analog circuits-Circuits used to process the analog signals.

Example: Amplifier, filter etc.

- Digital signal-Electrical signals those have only two discrete values- high(1) and low(0).

Example: Binary data stream

- Digital circuits-Electronic circuitsthose process Digital signals. Example: Calculator

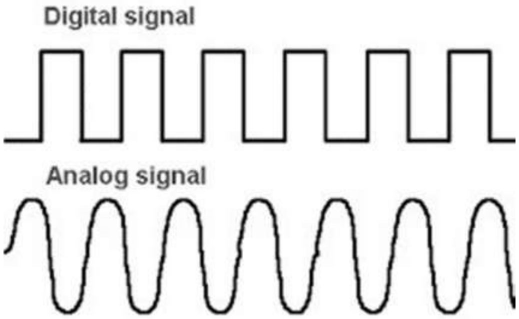


Figure 1 Picture of digital signal and analog signal

Introduction to Number System

Definition: Number system is the way to represent a number in different forms.

Types of Number system:

1. Binary Number System: It is the number system with base value 2 means it has only two digits to represent the data. The digits are (0, 1). E.g. 00,01,10,11,100 .
2. Decimal Number System: It is the number system with base value 10 means it has 10-digits to represent the data. The digits are(0-9). Eg. 0,1,2,3,4,5,6
3. Octal Number System: It is the number system with base value 8 means it has 8 digits to represent the data. The digits are (0-7).
4. Hexadecimal Number System : It is the number system with base value 16 means it has 16 digits to represent the data. The digits are (0-15). Eg. 0,1,2,3 .,9,A,B,C,D,E,F

1 The Binary Numbering System

We are habituated to use the decimal number system having the digits 0 to 9. This number system has a base of 10. But a computer or digital processor can recognize only binary 0 or 1.The Binary Numbering System is the most fundamental numbering system in all digital and computer based systems. It follow the same set of rules as the decimal numbering system. But unlike the decimal system which uses powers of ten, the binary numbering system works on powers of two giving a binary to decimal conversion from base-2 to base-10.

Digital logic and computer systems use just two values or states to represent a condition, a logic level 1 or a logic level 0 , and each 0 and 1 is considered to be a single digit in a Base-of-2 (bi) or binary numbering system . In the binary numbering system, a binary number such as 101100101 is expressed with a string of 1 s and 0 s with each digit along the string from right to left having a value twice that of the previous digit. But as it is a binary digit it can only have a value of either 1 or 0 therefore, q is equal to 2 (0 or 1) with its position indicating its weight within the string. As the decimal number is a weighted number, converting from decimal to binary (base 10 to base 2) will also produce a weighted binary number with the right-hand most bit being the Least Significant Bit or LSB, and the left-hand most bit being the Most Significant Bit or MSB.

2 Representation of a Binary Number

MSB	Binary Digit							LSB
2_8	2_7	2_6	2_5	2_4	2_3	2_2	2_1	2_0
256	128	64	32	16	8	4	2	1

3 Binary Number Names & Prefixes

The classification of individual bits into larger groups is generally referred to by the following more common names of:

Number of Binary Digits (bits)	Common Name
1	Bit
4	Nibble
8	Byte
16	Word
32	Double Word
64	Quad Word

4 Conversion from Binary to Decimal

Conversion of Integer Numbers

Expand the number given in binary form in the power of 2 and sum the values, the result which we will get will be in the decimal form. For example-

$$(1110)_2 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = (15)_{10}$$

Conversion of Decimal Point Number to Decimal

This can also be done in the same way, however after the decimal point the number should be multiplied with 2^{-1}, 2^{-2} etc. For example,

$$(1110.011)_2 = 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}$$
$$= (15.375)_{10}$$

5 Conversion from Decimal to Binary

Integer Numbers

Divide the number by 2 and take only the remainder, if division is completed than take only the remainder which gives the binary number.

Suppose we are converting the decimal number $(87)^{10}$. Now the conversion is shown below.

2	87	→ 1
2	43	→ 1
2	21	→ 1
2	10	→ 0
2	5	→ 1
2	2	→ 0
	1	

∴ $(87)_{10} = (1110101)_2$

For Fractional Numbers

In this case, the successive multiplication is done. The number which is to be converted is multiplied with base or radix of binary number which is 2. The integer part or the carry of the product is taken out and the same process is repeated until we get an integer. For example- The binary equivalent of $(.95)_{10}$ is evaluated as follows.

$$\begin{aligned}
.95 \times 2 &= 1.90 - - - - - - - - - - 1 \textit{ is taken out} \\
.90 \times 2 &= 1.80 - - - - - - - - - - 1 \\
.80 \times 2 &= 1.60 - - - - - - - - - - 1 \\
.60 \times 2 &= 1.20 - - - - - - - - - - -1 \\
.20 \times 2 &= .40 - - - - - - - - - - - - -1 \\
.40 \times 2 &= .80 - - - - - - - - - - - - -0
\end{aligned}$$

Since, we are not getting the integer value after successive multiplication, we can approximate the value to be $(.111110\ .)_2$.

6 Conversion from Octal to decimal

A regular decimal number is the sum of the digits multiplied with 10^n.

Example #1

137 in base 10 is equal to each digit multiplied with its corresponding 10^n:

$137_{10} = 1\times10^2+3\times10^1+7\times10^0 = 100+30+7$

Octal numbers are read the same way, but each digit counts 8^n instead of 10^n. Multiply each digit of the hex number with its corresponding 8^n.

7 Conversion from Octal to Binary

Converting from octal to binary is as easy as converting from binary to octal. Simply look up each octal digit to obtain the equivalent group of three binary digits.

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

Octal = 3 4 5

Binary = 011 100 101 = 011100101 binary

8 Conversion fromOctal to Hexadecimal

When converting from octal to hexadecimal, it is often easier to first convert the octal number into binary and then from binary into hexadecimal. For example, to convert 345 octal into hex:

(from the previous example)

Octal = 3 4 5

Binary = 011 100 101 = 011100101 binary

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

9 Conversion from Hexadecimal to decimal

A regular decimal number is the sum of the digits multiplied with 10^n.

Example #1

137 in base 10 is equal to each digit multiplied with its corresponding 10^n:

$137_{10} = 1\times10^2+3\times10^1+7\times10^0 = 100+30+7$

Hex numbers are read the same way, but each digit counts 16^n instead of 10^n. Multiply each digit of the hex number with its corresponding 16^n.

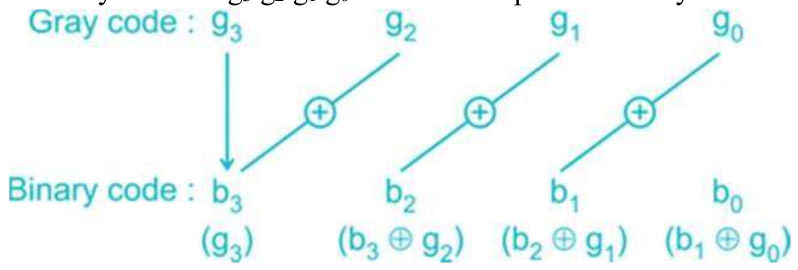
Example #2

3B in base 16 is equal to each digit multiplied with its corresponding 16^n:

$3B_{16} = 3\times16^1+11\times16^0 = 48+11 = 59$

10 Conversion from Gray Code to Binary Code

Let Gray Code be g_3 g_2 g_1 g_0. Then the respective Binary Code can be obtained as follows:


Gray code : g_3 g_2 g_1 g_0
Binary code : b_3 b_2 b_1 b_0
(g_3)
$(b_3 \oplus g_2)$
$(b_2 \oplus g_1)$
$(b_1 \oplus g_0)$

i.e.

$b_3 = g_3$ $b_2 =$

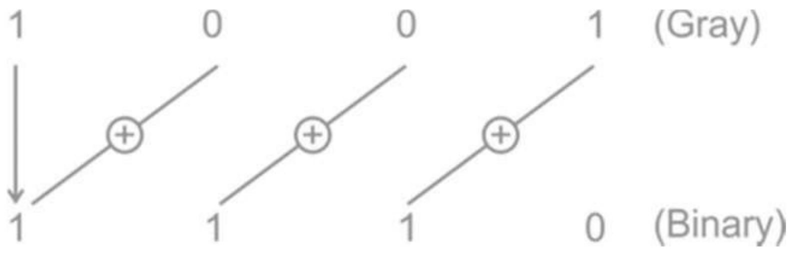
b_3□ g_2 $b_1 =$

b_2□ g_1

$b_0 = b_1$□ g_0 Example:

Gray Code: g_3 g_2 g_1 g_0 = 1 0 0 1 then Binary Code: b_3 b_2 b_1 b_0

$b_3 = g_3 = 1$



$b_2 = b_3$□ $g_2 = 1$ □ $0 = 1$

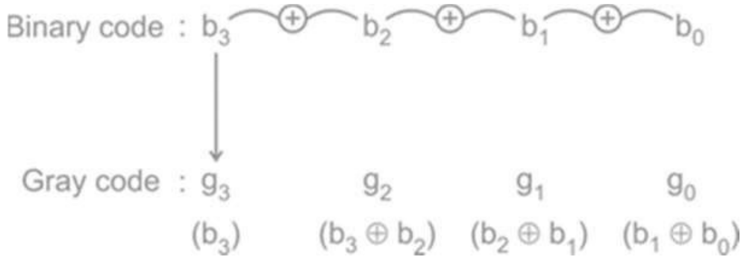
$b_1 = b_2$□ $g_1 = 1$ □ $0 = 1$

$b_0 = b_1$□ $g_0 =1$ □ $1 = 0$

∴ Final Binary Code: 1 1 1 0

11 Conversion from Binary code to Gray Code

Let Binary code be b_3 b_2 b_1 $b_{0.}$ Then the respective Gray Code can be obtained is as follows



Binary code : b_3 ⊕ b_2 ⊕ b_1 ⊕ b_0
Gray code : g_3 g_2 g_1 g_0
(b_3)
$(b_3 \oplus b_2)$
$(b_2 \oplus b_1)$
$(b_1 \oplus b_0)$

i.e.

$g_3 = b_3$ g_2 =

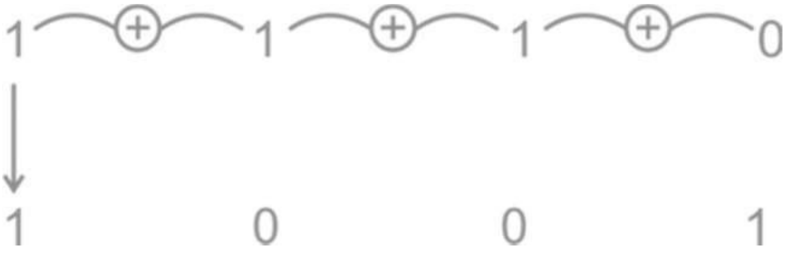
b_3□ b_2 g_1 =

b_2□ b_1 g_0 =

b_1□ b_0

Example:

Binary Code: b_3 b_2 b_1 b_0 = 1 1 1 0 Gray Code: g_3 g_2 g_1 g_0



$g_3 = b_3 = 1$

$g_2 = b_3$□ b_2 = 1 □ 1 = 0

$g_1 = b_2$□ b_1 = 1 □ 1 = 0

$g_0 = b_1$□ b_0 =1 □ 0 = 1

∴ Final Gray code: 1 0 0 1

12 Conversion Table from Binary to Gray Code:

Decimal	Binary	Gray
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

Binary Coded Decimal:

BCD or Binary Coded Decimal is that number system or code which has the binary numbers or digits to represent a decimal number.A decimal number contains 10 digits (0-9). Now the equivalent binary numbers can be found out of these 10 decimal numbers. In case of BCD the binary number formed by four binary digits, will be the equivalent code for the given decimal digits. In BCD we can use the binary number from 0000-1001 only, which are the decimal equivalent from 0-9 respectively. It is also called 8421 code.

Binary Coded Decimal Representation of a Decimal Number

Binary Power	2^3	2_2	2_1	2_0
Binary Weight:	8	4	2	1

The decimal weight of each decimal digit to the left increases by a factor of 10. In the BCD number system, the binary weight of each digit increases by a factor of 2 as shown. Then the first digit has a weight of 1 (2^0), the second digit has a weight of 2 (2^1), the third a weight of 4 (2^2), the fourth a weight of 8 (2^3).

Then the relationship between decimal (denary) numbers and weighted binary coded decimal digits is given below.

Truth Table for Binary Coded Decimal

Decimal Number	BCD 8421 Code
0	0000 0000
1	0000 0001
2	0000 0010
3	0000 0011
4	0000 0100
5	0000 0101
6	0000 0110
7	0000 0111
8	0000 1000
9	0000 1001
10 (1+0)	0001 0000
11 (1+1)	0001 0001
12 (1+2)	0001 0010
…	…
20 (2+0)	0010 0000
21 (2+1)	0010 0001

22 (2+2)	0010 0010
etc, continuing upwards in groups of four	

Then we can see that 8421 BCD code is nothing more than the weights of each binary digit, with each decimal (denary) number expressed as its four-bit pure binary equivalent.

Decimal-to-BCD Conversion

As we have seen above, the conversion of decimal to binary coded decimal is very similar to the conversion of hexadecimal to binary. Firstly, separate the decimal number into its weighted digits and then write down the equivalent 4-bit 8421 BCD code representing each decimal digit as shown.

Binary Coded Decimal Example No1

Using the above table, convert the following decimal (denary) numbers: 85_{10}, 572_{10} and 8579_{10} into their 8421 BCD equivalents.

85_{10} = 1000 0101 (BCD)

572_{10} = 0101 0111 0010 (BCD)

8579_{10} = 1000 0101 0111 1001 (BCD)

Note that the resulting binary number after the conversion will be a true binary translation of decimal digits. This is because the binary code translates as a true binary count.

BCD-to-Decimal Conversion

The conversion from binary coded decimal to decimal is the exact opposite of the above. Simply divide the binary number into groups of four digits, starting with the least significant digit and then write the decimal digit represented by each 4-bit group. Add additional zero s at the end if required to produce a complete 4-bit grouping. So for example, 110101_2 would become: $0011\ 0101_2$ or 35_{10} in decimal.

Convert the following binary numbers: 1001_2, 1010_2, 1000111_2 and 10100111000.101_2 into their decimal equivalents.

$1001_2 = 1001_{BCD} = 9_{10}$

1010_2 = this will produce an error as it is decimal 10_{10} and not a valid BCD number

$1000111_2 = 0100\ 0111_{BCD} = 47_{10}$

$10100111000.101_2 = 0101\ 0011\ 0001.1010_{BCD} = 538.625_{10}$

ASCII:

ASCII stands for American Standard Code for Information Interchange. Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as 'a' or '@' or an action of some sort. ASCII was developed a long time ago and now the non-printing characters are rarely used for their original purpose. Below is the ASCII character table and this includes descriptions of the first 32 non-printing characters. ASCII was actually designed for use with teletypes and so the descriptions are somewhat obscure. If someone says they want your CV however in ASCII format, all this means is they want 'plain' text with no formatting such as tabs, bold or underscoring - the raw format that any computer can understand. This is usually so they can easily import the file into their own applications without issues. Notepad.exe creates ASCII text, or in MS Word you can save a file as 'text only'

ASCII was developed by the American National Standards Institute (ANSI).
In an ASCII file, each alphabetic, numeric, or special character is represented with a 7-bit binary number (a string of seven 0s or 1s). 128 possible characters are defined.

UNIX and DOS-based operating systems use ASCII for text files. Windows NT and 2000 uses a newer code, Unicode. IBM's S/390 systems use a proprietary 8-bit code called EBCDIC. Conversion programs allow different operating systems to change a file from one code to another.

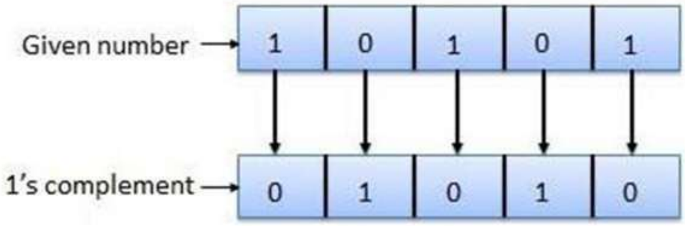
EBCDIC:(Extended Binary Coded Decimal Interchange Code)

EBCDIC is a binary code for alphabetic and numeric characters that IBM developed for its larger operating systems. It is the code for text files that is used in IBM's OS/390 operating system for its S/390 servers and that thousands of corporations use for their legacy applications and databases. In an EBCDIC file, each alphabetic or numeric character is represented with an 8-bit binary number (a string of eight 0's or 1's). 256 possible characters (letters of the alphabet, numerals, and special characters) are defined.
IBM's PC and workstation operating systems do not use IBM's proprietary EBCDIC. Instead, they use the industry standard code for text, ASCII. Conversion programs allow different operating systems to change a file from one code to another.

ONE S COMPLEMENT AND TWO S COMPLEMENT

13 One's complement

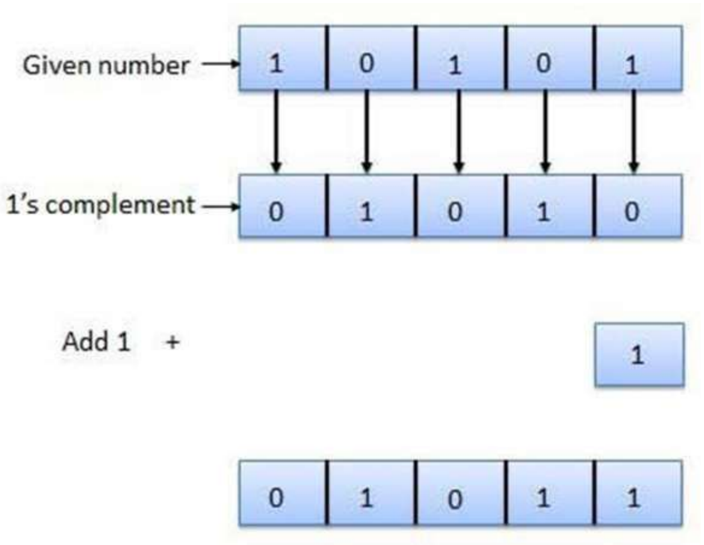
The 1's complement of a number is found by changing all 1's to 0's and all 0's to 1's. This is called as taking complement or 1's complement. Example of 1's Complement is as follows.



14 Two's complement

The 2's complement of binary number is obtained by adding 1 to the Least Significant Bit (LSB) of 1's complement of the number.

2's complement = 1's complement + 1
Example of 2's Complement is as follows.



9 s and 10 s complement methods:

The complements are used to make the arithmetic operations in digital system easier. In this article we will discuss about the following topics

1. 9s complement
2. 10s complement
3. 9s complement subtraction
4. 10s complement subtraction

Now first of all let us know what 9's complement is and how it is done.

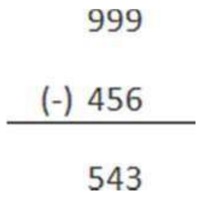
To obtain the 9's complement of any number we have to subtract the number with (10^n - 1) where n = number of digits in the number, or in a simpler manner we have to divide each digit of the given decimal number with 9. The table given below will explain the 9's complement more easily.

Decimal digit	9s complement
0	9
1	8
2	7
3	6
4	5
5	4
6	3
7	2
8	1
9	0

Now coming to 10's complement

it is relatively easy to find out the 10's complement after finding out the 9,s complement of that number. We have to add 1 with the 9's complement of any number to obtain the desired 10's complement of that number. Or if we want to find out the 10's complement directly, we can do it by following the following formula, (10^n - number), where n = number of digits in the number. An example is given below to illustrate the concept of obtaining 10 s complement

Let us take a decimal number 456, 9's complement of this number will be



```
     999
 (-) 456
 -------
     543
```

10's complement of this no.

9's complement subtraction:

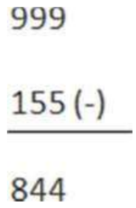
We will understand this method of subtraction via an example

A = 215

B = 155

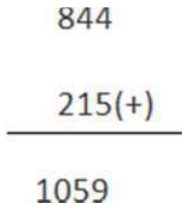
We want to find out A-B by 9's complement subtraction method

First we have to find out 9 s complement of B



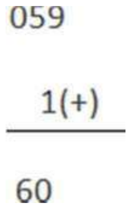
```
999
155 (-)
-------
844
```

Now we have to add 9 s complement of B to A



```
   844
   215(+)
 --------
  1059
```

The left most bit of the result is called carry and is added back to the part of the result without it



```
059
  1(+)
------
 60
```

Another different type of example is given

A = 4567

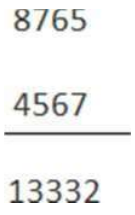
B = 1234

We need to find out A - B

9's complement of B

8765

Adding 9's complement of B with A



$$\begin{array}{r} 8765 \\ 4567 \\ \hline 13332 \end{array}$$

Adding the carry with the result we get

3333

Now the answer is - 3333

NB if there is no carry the answer will be (9 s complement of the answer)

Subtraction by 10's complement:

Again we will show the procedure by an example

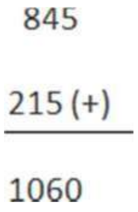
Taking the same data

A = 215

B = 155

10's complement of B = 845

Adding 10 s complement of B to A

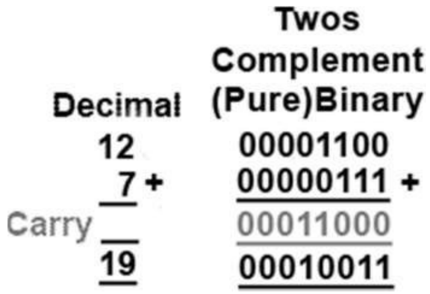


$$\begin{array}{r} 845 \\ 215\,(+) \\ \hline 1060 \end{array}$$

In this case the carry is omitted
The answer is 60
15 Representation of Negative Number

In case of a negative number we can go for 2 scomplement representation of a signed number. Example- 9 = 0000 1001 1 s complement = 1111 0110. Adding 1 we get = 11110111 which is the 2 s complement representation of (-9).

2.4 Two s Complement Addition: An example of addition using 8 bit twos complement notation. When adding two positive numbers, the sign bit (msb) will both be 0, so the numbers are written and added as a pure 8-bit binary addition



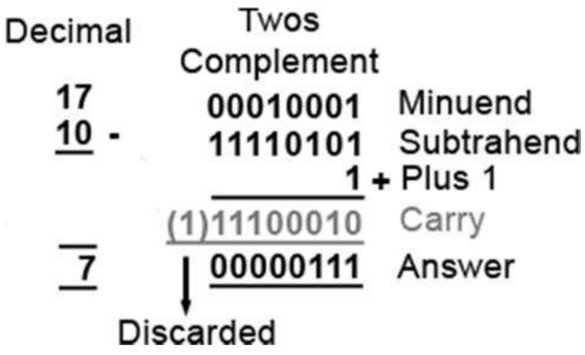
	Decimal	Twos Complement (Pure)Binary
	12	00001100
	7 +	00000111 +
Carry		00011000
	19	00010011

16 Two s Complement Subtraction

One positive number (the subtrahend) is subtracted from a larger positive number (the minuend). In this case the minuend is $17_{10}$and the subtrahend is 10_{10}.

When these three lines of digits, and any carry 1 bits are added, remembering that in twos complement, any carry from the most significant bit is discarded.

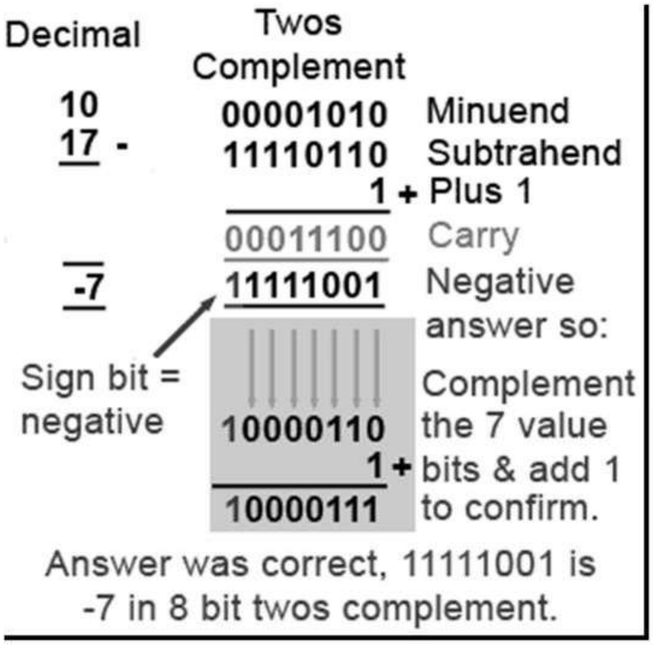
Case1:



Decimal	Twos Complement	
17	00010001	Minuend
10 -	11110101	Subtrahend
	1 +	Plus 1
	(1)11100010	Carry
7	00000111	Answer
	Discarded	

Some subtractions will of course produce an answer with a negative value. In Fig. below the result of subtracting 17 from 10 should -7_{10} but the twos complement answer of 11111001_2 certainly doesn’t look like −7. However the sign bit is indicating correctly that the answer is negative, so in this case the 7 bits indicating the value of the negative answer need to be 'twos complemented' once more to see the answer in a recognizable form.

Case2:



Answer was correct, 11111001 is -7 in 8 bit twos complement.

17 BOOLEANLAWS

There are different types of Boolean Laws as follows.

AND law

These laws use the AND operation. Therefore they are called as **AND** laws.

(i) $A.0 = 0$	(ii) $A.1 = A$
(iii) $A.A = A$	(iv) $A.\overline{A} = 0$

OR law

These laws use the OR operation. Therefore they are called as **OR** laws.

(i) $A + 0 = A$	(ii) $A + 1 = 1$
(iii) $A + A = A$	(iv) $A + \overline{A} = 1$

INVERSION law

This law uses the NOT operation. The inversion law states that double inversion of variable results in the original variable itself.

$\overline{\overline{A}} = A$

Commutative law

Any binary operation which satisfies the following expression is referred to as commutative operation.

(i) A.B = B. A (ii) A + B = B + A

Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

Associative law

This law states that the order in which the logic operations are performed is irrelevant as their effect is the same.

(i) (A.B).C = A.(B.C) (ii) (A + B) + C = A + (B + C)

Distributive law

This law is composed of two operators, AND and OR.

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

Let us show one use of this law to prove the expression

Proof: $A + B.C = (A+B).(A+C)$

$$A + B.C = A.1 + B.C \; [Since,\ A.1 = A]$$
$$= A.(1+B) + B.C \; [Since,\ B+1 = 1]$$
$$= A.1 + AB + BC$$
$$= A.(1+C) + AB + BC \; [Since,\ A.A = A.1 = A]$$
$$= A.(A+C) + B.(A+C)$$
$$= (A+B).(A+C)$$

(a) A (B + C) = A B + A C
(b) A + (B C) = (A + B) (A + C)

now proof for 1st no. is as simple as we can see

=AB+BC

=A(B+C) L.H.S=R.H.S.

now proof for 2nd. law
R.H.S. = (A+B)(A+C)

=A*A+A*C+B*A+B*C

=A+A*C+A*B+B*C   (.: B*A=A*B & A*A=A)

=A (1+C+B) + B*C =A+BC
(.:1+B+C=11)
L.H.S. =R.H.S.

Redundant Literal Rule

$AA+\overline{A}A+AB+\overline{A}B$

$$A+\overline{A}B=A+B$$
$$Similarly,$$

From truth table it is proved that, $A(\overline{A}+B)=AB$

Associative Laws for Boolean Algebra

This law is for several variables, where the OR operation of the variables result is same though the grouping of the variables. This law is quite same in case of AND operators.

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

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

Absorption Laws for Boolean Algebra

$A+A.B=A$

$A+A.B=A(1+B)=A$

$Similarly,\ A(A+B)=A$

18 De-Morgan's Theorem

Theorem 1: The compliment of the product of two variables is equal to the sum of the compliment of each variable. Thus according to De-Morgan's laws or De-Morgan's theorem if A and B are the two variables or Boolean numbers. Then accordingly

$$\overline{A+B}=\overline{A}\;\overline{B}$$
$$and\ \ \overline{AB}=\overline{A}+\overline{B}$$

Theorem 2: The compliment of the sum of two variables is equal to the product of the compliment of each variable. Thus according to De Morgan s theorem if A and B are the two variables then.

$$(A+B)'=A'.B'$$

7.16 Examples of Boolean algebra

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

$$Here,\ \overline{(A+\overline{B})(C+\overline{D})}\ [\ As\ per\ De\ Morgan\ Therem\ \overline{x.y}=\overline{x}+\overline{y}\ ]$$

$$=\overline{(A+\overline{B})(C+\overline{D})}\ [\ As\ per\ De\ Morgan\ Therem\ \overline{x+y}=\overline{x}.\overline{y}\ ]$$

$$=\overline{A}.\overline{\overline{B}}+\overline{C}.\overline{\overline{D}}$$

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

1. Another example,

$$\overline{\overline{AB}+\overline{A}+AB}=\overline{\overline{AB}}.\overline{\overline{A}}.\overline{AB}\quad [Appling\ De\ Morgan\ Therem]$$

$$=AB.A.\overline{AB}$$

$$=AB.\overline{AB}.A=0$$

$$Example\ simplify,\ AB+A\overline{B}C+B\overline{C}=A(B+\overline{B}C)+B\overline{C}=A(B+\overline{B})(B+C)+$$

$$=AB+AC+B\overline{C}=AB(C+\overline{C})+AC+B\overline{C}=ABC+AB\overline{C}+AC+B\overline{C}$$

$$=AC(1+B)+B\overline{C}(A+1)=AC+B\overline{C}$$

Minimization of logic expressions by algebraic method:

Some examples:

1.

$$\begin{aligned}(A+B)(A+C) &= AA+AC+AB+BC\\ &= A+AC+AB+BC\\ &= A(1+C+B)+BC\\ &= A\cdot 1+BC\\ &= A+BC\end{aligned}$$

2.

$$\begin{aligned}(\overline{A}+B)(A+B) =\ & \mathrm{B}\,(\,\overline{A}+\mathrm{A})\\ & \mathrm{B}\,(1)\\ & \mathrm{B}\end{aligned}$$

3.

$$A\overline{C} + AB\overline{C} = \quad A\overline{C}\ (1 + \mathrm{B})$$

$$A\overline{C}\ (1)$$

$$A\overline{C}$$

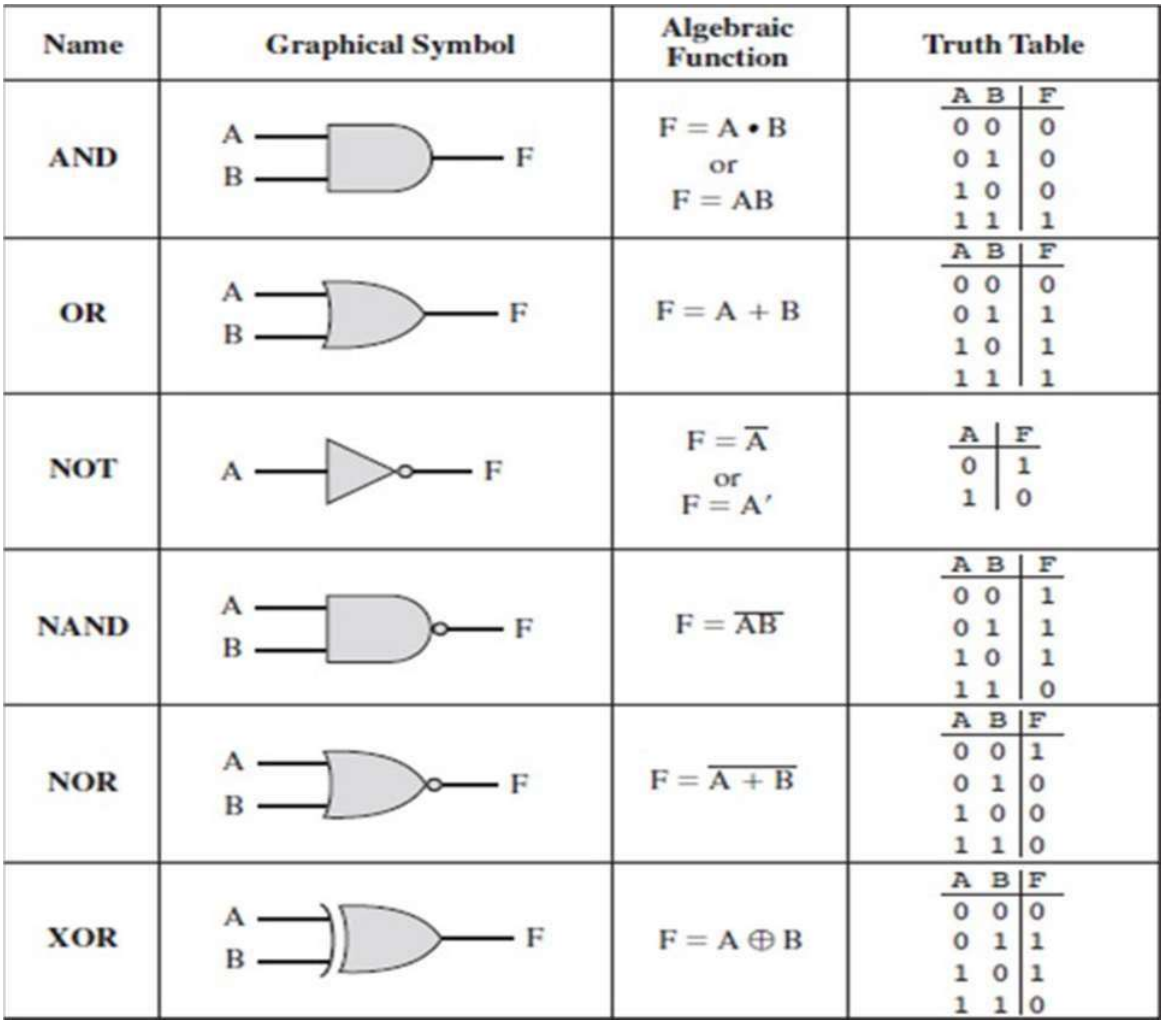
4.

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

$$A\,\overline{B}\,(1)$$

$$A\,\overline{B}$$

LOGIC GATES- Transistor or IC circuits produce digital output for given digital inputs following some logical operations.



Name	Graphical Symbol	Algebraic Function	Truth Table
AND	A B F	$F = A \bullet B$ or $F = AB$	A B F 0 0 0 0 1 0 1 0 0 1 1 1
OR	A B F	$F = A + B$	A B F 0 0 0 0 1 1 1 0 1 1 1 1
NOT	A F	$F = \overline{A}$ or $F = A'$	A F 0 1 1 0
NAND	A B F	$F = \overline{AB}$	A B F 0 0 1 0 1 1 1 0 1 1 1 0
NOR	A B F	$F = \overline{A + B}$	A B F 0 0 1 0 1 0 1 0 0 1 1 0
XOR	A B F	$F = A \oplus B$	A B F 0 0 0 0 1 1 1 0 1 1 1 0

BASIC GATES- AND, OR, NOT

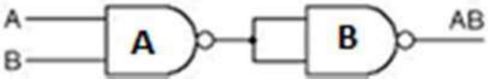
UNIVERSAL LOGIC GATES:NAND and NOR

Any basic gates can be implemented using these two universal gates using some logic operations.

(a) Implementation of AND gate using NAND gate:

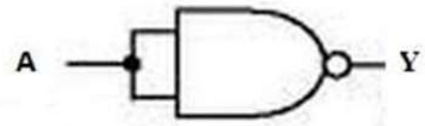
For a 2-input (A,B), AND gate has the Boolean function as Y= A.B

It can be written as Y= $\overline{\overline{A.B}}$



(b) Implementation of NOT gate using NAND gate:

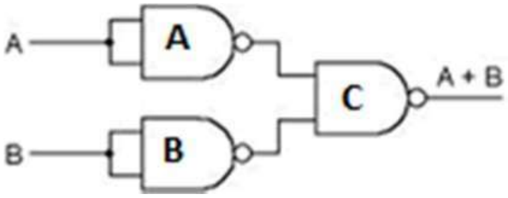
For a 2-input (A,B),NOT gate has the Boolean function as F= $\bar{A}$
It can be written as F= $\overline{A.A}$



(c) Implementation of OR gate using NAND gate:

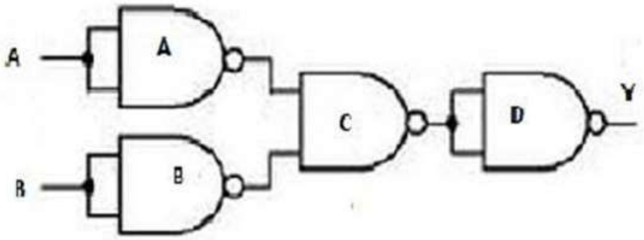
For a 2-input (A,B), OR gate has the Boolean function as Y= A+B.

It can be written as Y= $\overline{\overline{A+B}}$

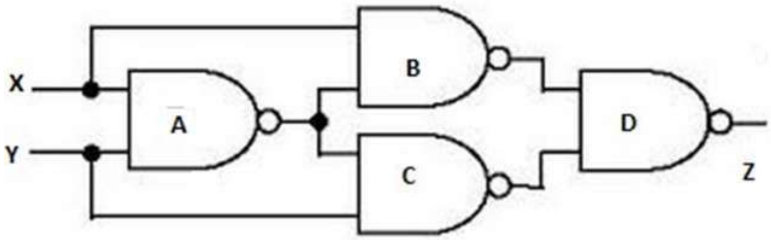


(d) Implementation of NOR gate using NAND gate:

For a 2-input (A, B), OR gate has the Boolean function as Y= $\overline{A+B}$. It can be written as
Y= $\overline{\overline{\bar{A}.\bar{B}}}$ = $\bar{A}.\bar{B}$ = $\overline{A+B}$

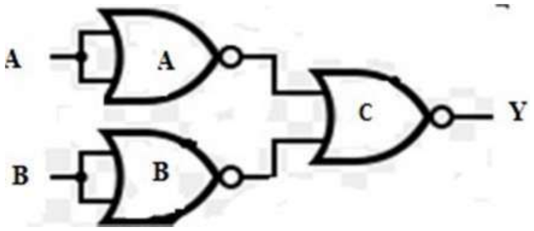


(e) Implementation of NOR gate using NAND gate:



(f) Implementation of AND gate using NOR gate:

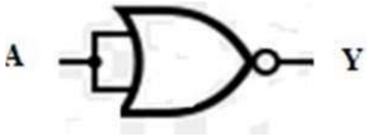
For a 2 input (A,B), AND gate has the Boolean function as Y= A.B. It can be written as Y= $\overline{\overline{A.B}}$



(g) Implementation of NOT gate using NOR gate:

For a 2-input (A,B),NOT gate has the Boolean function as F= $\bar{A}$

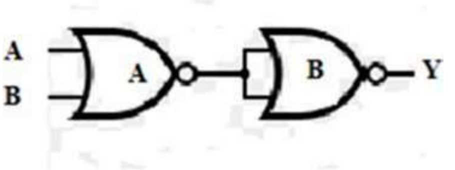
It can be written as F= $\overline{A.A}$



(h) Implementation of OR gate using NOR gate:

For a 2-input (A,B), OR gate has the Boolean function as Y= A+B

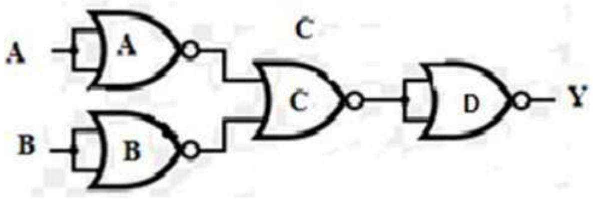
It can be written as Y= $\overline{\overline{A + B}}$



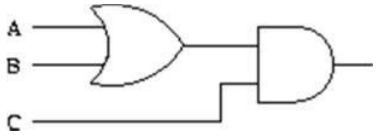
(i) Implementation of NAND gate using NOR gate:

For a 2-input (A,B), OR gate has the Boolean function as Y= $\overline{A.B}$ It can be written as

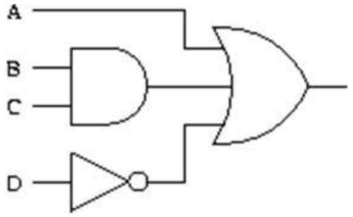
Y= $\overline{\overline{\bar{A} + \bar{B}}}$ =$\bar{A} + \bar{B}$ =$\overline{A.B}$



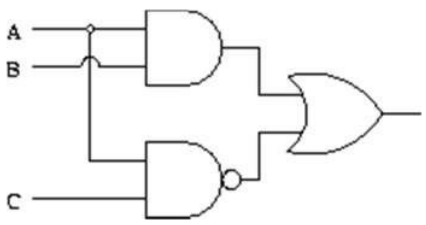
- Draw a logic circuit for (A + B)C.



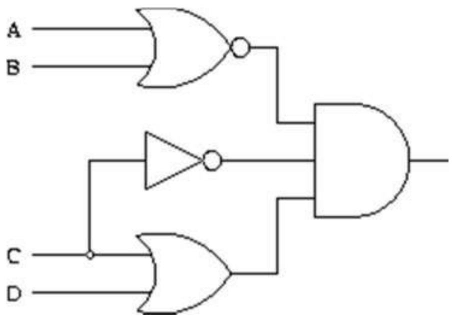
- Draw a logic circuit for A + BC + D.



- Draw a logic circuit for AB + AC.



- Draw a logic circuit for (A + B)(C + D)C.

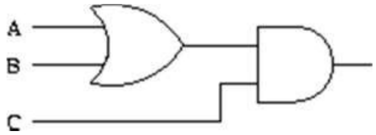


SOLVED PROBLEMS

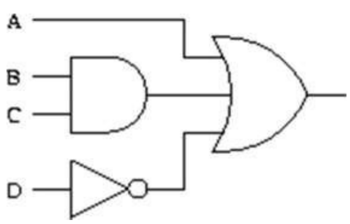
1. Simplify $AB + A\overline{B}C + B\overline{C}$

$$AB + A\overline{B}C + B\overline{C} = A(B + \overline{B}C) + B\overline{C} = A(B + \overline{B})(B + C) + B\overline{C}$$
$$= AB + AC + B\overline{C} = AB(C + \overline{C}) + AC + B\overline{C} = ABC + AB\overline{C} + AC + B\overline{C}$$
$$= AC(1 + B) + B\overline{C}(A + 1) = AC + B\overline{C}$$

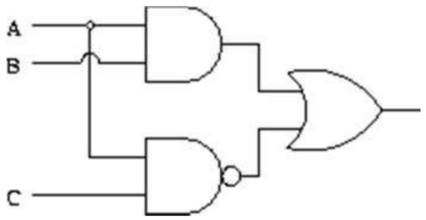
2. Draw a logic circuit for (A + B)C.



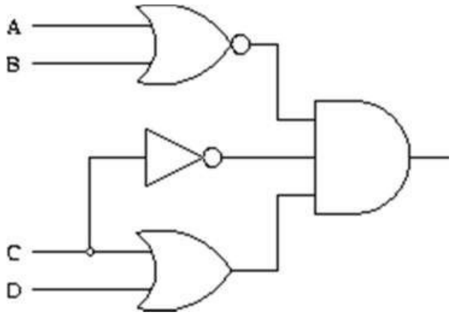
3. Draw a logic circuit for A + BC + D.



4. Draw a logic circuit for AB + AC.



5. Draw a logic circuit for (A + B) (C + D) C.



..

MCQ

1. Given that $(\sqrt{61})_b$ $=7_{10}$, the value of b is

A. 6 B. 8 C. 4 D. 5

Ans. B

Subtraction of two signed numbers is performed with

1's complement

A. 2's complement
B. 9's complement
C. 10's complement

Ans. B

2. Representation of 8620 in binary is

A. 1000_0111_1110_0000
B. 1000_0110_0010_0000
C. 1000_0110_1010_0000
D. 1011_0110_0010_0000

Ans. B

3. -9 with signed 2's complement representation is

A. 10001001
B. 11110110
C. 11110111
D. 11110011

Ans. C

4. NOT gate is also known as

A. Converter
B. Inverter
C. Complementer
D. Both b and c

Ans. D

5. $(72.45)_{10} =(?)_2$

A. 100100.01110
B. 110000.00111
C. 100100.1110
D. 111000.01110

Ans. A

6. $(381B)_{16}=(?)_8$

A. 33033
B. 34033
C. 30334
D. 33004

Ans. B

7. $(49)_{16=} (?)_2$

A. 10001111
B. 1110 0010
C. 01001011
D. 01011011

Ans. C

8. $(137)_8 = (?)_{16}$

A. 5F
B. 5B
C. F5
D. 59 Ans. A

9. $(632)_{16} = (?)_{10}$

A. 1587
B. 1588
C. 1557
D. 1586

Ans.D

....

Answer The Following Questions:

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

1. Simplify

$$\overline{\overline{AB}+\overline{A}+AB}=\overline{\overline{AB}}.\overline{\overline{A}}.\overline{AB}$$

2. Prove that,

3. Implement the Boolean function, (A + B) C using universal gate only.

4. State DeMorgan s theorem.

5. State and prove Absorption, Associative and Distributive law.

6. Using Boolean algebra, verify

(A+B)(B+C)(C+A)=AB+BC+CA

7. Explain the term Universal Logic gate

8. Explain how the basic gates are realised using NAND gate.

9. Explain how the basic gates are realised using NOR gate..

10. Write the truth table of XOR gate and construct it with NAND gate.

....

Solve the problems

1. Convert the following decimal number to binary equivalent:

   4097.18

2. Convert the following binary number to octal and then hexadecimal:

   1011101.1011

3. Simplify the following Boolean function:
   (A+B)(A'+C)

ANS:

1.1000000000001.001,

2. 135.54, 5D.B

3. AC+A'B+BC

Sum of Products & Product of Sums

All Boolean expressions can be converted into one of two standard forms; the sum-of-product form and the product-of sum forms.

The Sum-of-Product (SOP):

SOP form means that the inputs of each term are multiplied using AND function, then all terms are added together using OR function. The variables in each term are not necessarily all the variables of the function. For example, a SOP of F(A,B,C) may contain a term that contains only the variable A but not B nor C, in such case the term is not in its standard SOP form. Standard SOP term must contain all the function variables. From Boolean algebra thermos (X+X'=1), then if the term is multiplied by (X+X'), it becomes in the standard SOP form, but its value is not affected.

Example 1:

The following function is written in the SOP form: F(A,B,C)=A+BC'+A'BC

Solution 1:
The inputs to the function F are A, B and C. In each term the inputs are ANDed then all terms are ORed to form the function F. The last term A'BC contains all the inputs of the function (A, B and C), so, this term is written in standard form. But the second term BC' is not in standard form because the input A does not exist, then multiply by (A'+A). The same is done for the remaining term as follows:
F(A,B,C)=A(B+B')(C+C')+BC'(A+A')+A'BC

$F(A,B,C)=ABC+ABC'+AB'C+AB'C'+ABC'+A'BC'+A'BC$
$=ABC+ABC'+AB'C+AB'C'+A'BC'+A'BC$

The Product-of-Sum (POS):

POS form means that the inputs of each term are Added together using OR function then all terms are multiplied together using AND function. The variables in each term are not necessarily all the variables of the function. For example, a POS of F(A,B,C) may contain a term that contains only the variable A but not B nor C, in such case the term is not in its standard POS form. Standard POS term must contain all the function variables. From Boolean algebra thermos ($X.X'=0$), then if the term is added to ($X.X'$), it becomes in the standard POS form, but its value is not affected.

Example 2:
The following function is written in the POS form:
$F(A,B,C)=A.(B+C').(A'+B+C')$

Solution 2:

The inputs to the function F are A, B and C. In each term the inputs are ORed then all terms are ANDed to form the function F. The last term ($A'+B+C'$) contains all the inputs of the function (A, B and C), so, this term is written in standard form. But the second term ($B+C'$) is not in standard form because the input A does not exist, then add ($A'.A$). The same is done for the remaining term as follows:
$F(A,B,C)=[A+(B.B')+(C.C')].[(B+C')+(A.A')].(A'+B+C')$
$F(A,B,C)=[(A+B+C).(A+B+C').(A+B'+C).(A+B'+C')].[(A+B+C').(A'+B+C')].(A'+B+C')$
$=(A+B+C).(A+B+C').(A+B'+C).(A+B'+C').(A'+B+C')$ Minterms

Writing a function in its minterm format is equivalent to writing the function in its standard SOP format such that the value of the function at these terms is 1. So that if we have the truth table relating the input variables to the function F, then we can determine which cases result in F=1 and write the minterm form of the function.

Maxterms

Writing a function in its maxterm format is equivalent to writing the function in its standard POS format such that the value of the function at these terms is 0. So that if we have the truth table relating the input variables to the function F, we can determine which cases result in F=0 and write the maxterm form of the function.

Karnaugh Map

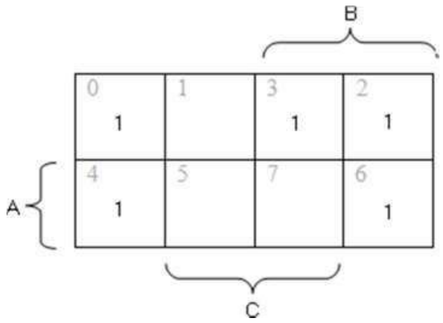
The Karnaugh map (K-map) provides a systematic way of simplifying Boolean algebra expressions. This can be done without thoroughly searching the basic theorems of Boolean algebra. Instead all the possible combinations of the variables written in the standard form for POS (product of sums) or SOP (sums of products) are plotted in cells arranged in a rectangle or square. Adjacent cells share a redundant Boolean variable. The simplification of the original Boolean expression comes form grouping the logical one s (minterms) or 0 s (maxterms). This eliminates the redundant variable and simplifies the original Boolean expression.

The circulation must be done according to the following rules:

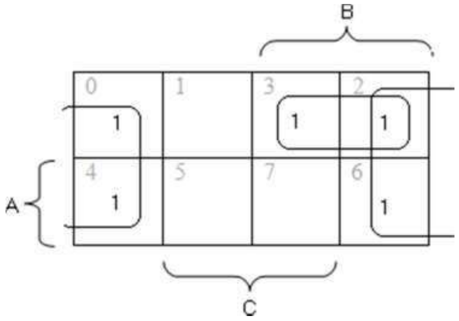
1. A group must contain either 1, 2, 4, 8, 16 ..cells.
2. Each cell in a group must be adjacent to one or more cells in the same group, but all cells do not have to be adjacent to each other.
3. ALWAYS include the most possible of 1 s in a group in accordance to rule (1).
4. Each 1 on the K-map must be included in at least one group. The 1 s in a group can be included in another group as long as the overlapping groups include non common 1 s.

Example 3:
Simply the following function using K-map
F(A,B,C)=A'B+ABC'+B'C'

Solution 3:
In order to use the K-map the function should be written in its min or max terms format.
F(A,B,C)=A'BC+A'BC'+ABC'+AB'C'+A'B'C'
F(A,B,C)=Σ(0,2,3,4,6)

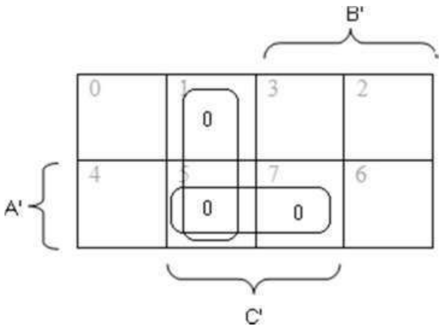


To write the function as simplified SOP then circle the 1's in K-map

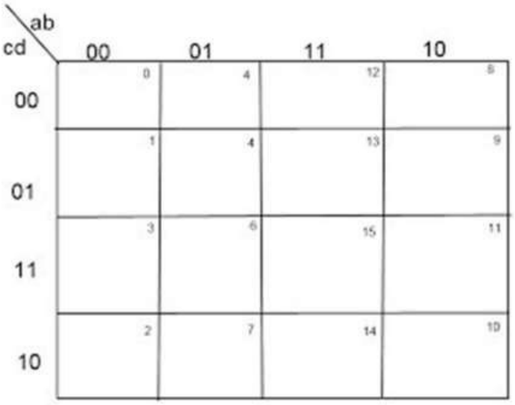


From K-map
F(A,B,C)=C'+BA'

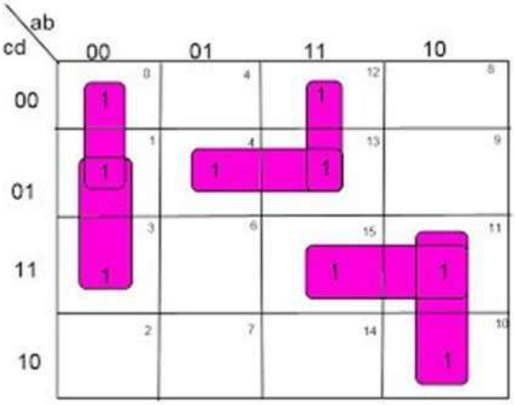
F is written in max term as follows
F(A,B,C)= Π(1,5,7)
To write the function as POS then circle the 0's in K-map



From K-map
F(A,B,C)=(A'+C').(B+C')
Syntax of 4 Variable K-Map:-

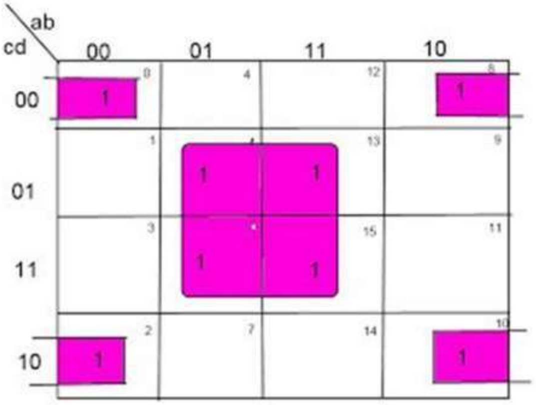


ex 1- Z=Σ(0,1,3,5,10,11,12,13,15)...SOP



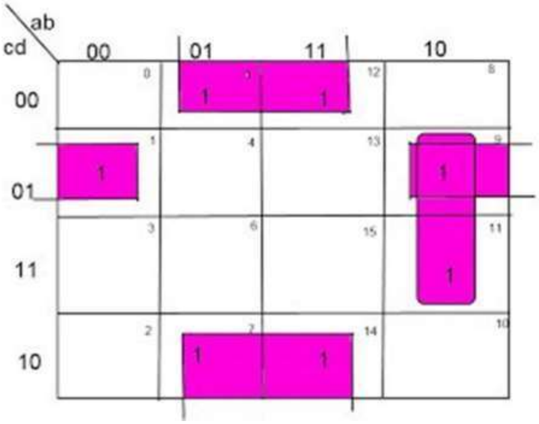
y=a'b'c'+a'b'd+bc'd+abc'+acd+ab'c

ex 2- Z=Σ(0,2,5,8,7,10,13,15)



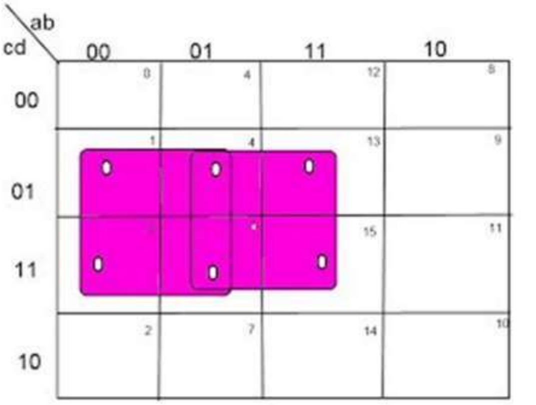
y=bd+b'd'(XOR)

Ex 3 - Z=Σ(4,1,9,6,12,14,11)



y=bd'+ab'd+a'b'd

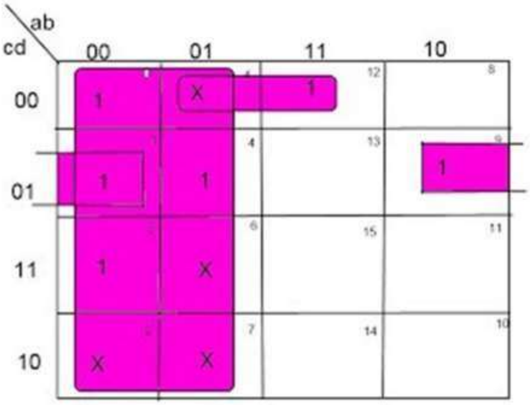
Ex 4 - S=Π(1,5,7,3,7,13,15)..POS



y=(a'+d').(b'+d')

Don't Care Condition(K-Map):-We use Don't care condition in K-Map because to make a group of 2 variable.

y=Σ(0,1,3,5,9,12)+Σd(2,4,6,7)



y=a'+bc'd'+b'c'd

Quine-McCluskeyAlgorithm

The method involves two steps:

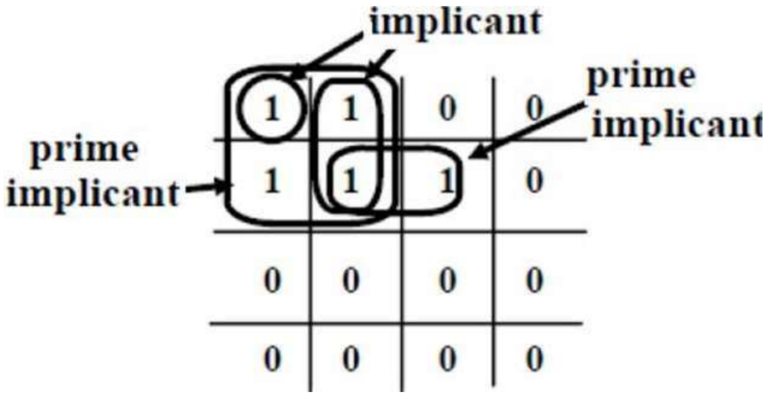
1. Finding all prime implicants of the function.
2. Use those prime implicants in a prime implicantchartto find the essential prime implicants of the function, as well as other prime implicantsthat are necessary to cover the function.

Implicant:A single 1, or group of 1's combined together on a map, is called an implicantof F.

Prime Implicant:An implicantwhich cannot further combine with other implicantsis called a Prime Implicant of F. A prime implicate is a group of 1,2,4,8 that cannot be contained in any larger group.

Prime implicant

1. Single 1 on a K-map represents a prime implicantif it is not adjacent to any other 1
2. Two adjacent 1 s on a K-map form a prime implicantif they are not contained in a group of four 1 s
3. Four adjacent 1 s on a K-map form a prime implicantif they are not contained in a group of eight 1 s



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

Essential Prime Implicant

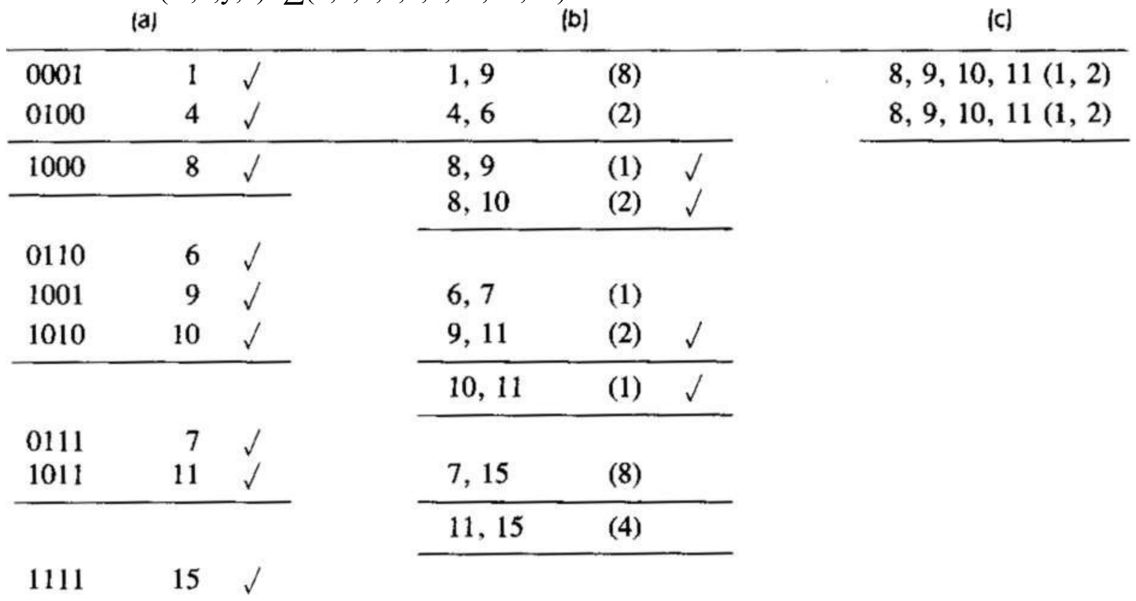
1.If minterm can be covered by only one prime implicant, then that prime implicant is called an Essential Prime Implicant
2.All essential prime implicants must be present in the minimum expression for F.
3.Therefore find all essential prime implicants first.
examine each 1 on map
if that 1 + all adjacent 1's covered by one term only (i.e. encircled by one loop only), that term is essential cover remaining 1's by minimum set of prime implicants

Groups of the minterms
1. Forms Groups of the mintermsis the following ways.
Group 1: terms with no 1s.
Group 2: terms which have only one 1.
Group 3 terms which have only two 1s and so on..
The terms should be written in the ascending order of their values.
Example: 0001, 1000, 0100, 0010 should be written as 0001, 0010, 0100, 1000 and forms group 2.
Each group has an equal number of ones in the input combination.
2.Each mintermcan only be adjacent to the mintermsin the next group.

Combining Minterms
2. Any two mintermswhich differ from each other by only one variable can be combined, and the unmatched variable removed (by marking - or x).
E.g., 0000 vs. 0100 yields 0-00 or 0x00 0000 vs. 1000 yields -000 or x000
3.When used in a combination, mark with a check. If cannot be combined, mark with a star. These are the prime implicants.
4.Continue again. Pair up rows from adjacent regions if they differ by exactly one bit.
5.Repeat until nothing left.

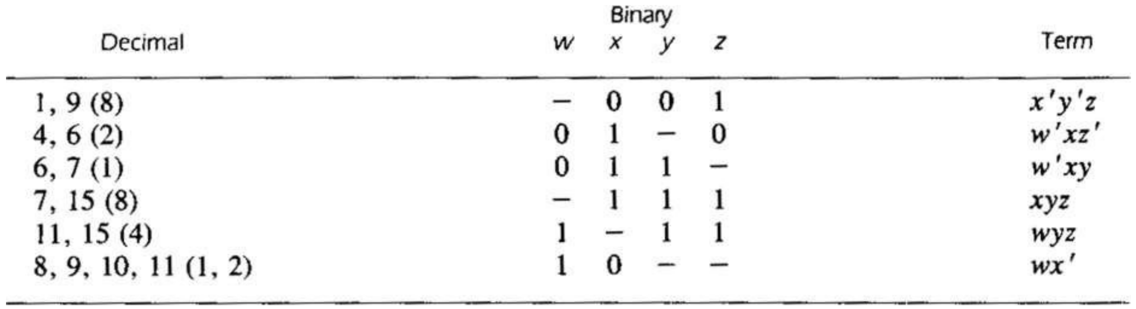
Prime implicantchart
1The table is required to find essential prime implicant
2 Prime-implicanttable rows = prime implicants columns = mintermsof the function place an "X" if the mintermis covered by the prime implicant.
If column has a single X, than the implicantassociated with the row is essential. It must appear in minimum cover as well as other prime implicantsthat are necessary to cover the function.
Example : Determine the prime implicantsof the function given below

F(w,x,y,z)=∑(1,4,6,7,8,9,10,11,15)



(a)			(b)			(c)
0001	1	✓	1, 9	(8)		8, 9, 10, 11 (1, 2)
0100	4	✓	4, 6	(2)		8, 9, 10, 11 (1, 2)
1000	8	✓	8, 9	(1)	✓	
			8, 10	(2)	✓	
0110	6	✓				
1001	9	✓	6, 7	(1)		
1010	10	✓	9, 11	(2)	✓	
			10, 11	(1)	✓	
0111	7	✓				
1011	11	✓	7, 15	(8)		
			11, 15	(4)		
1111	15	✓				

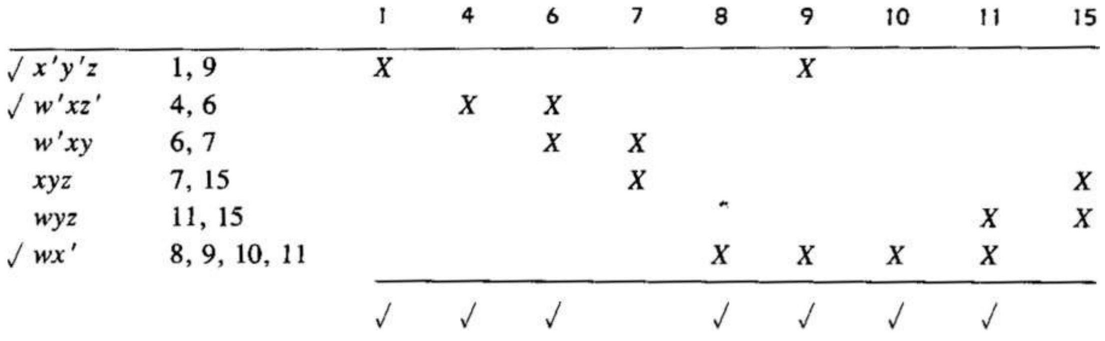
Prime implicants



	Binary				
Decimal	w	x	y	z	Term
1, 9 (8)	–	0	0	1	x'y'z
4, 6 (2)	0	1	–	0	w'xz'
6, 7 (1)	0	1	1	–	w'xy
7, 15 (8)	–	1	1	1	xyz
11, 15 (4)	1	–	1	1	wyz
8, 9, 10, 11 (1, 2)	1	0	–	–	wx'

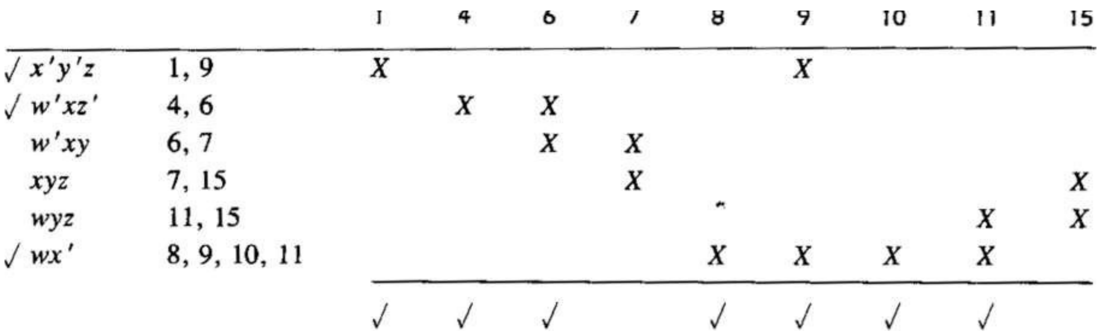
Prime implicant chart

- Eliminate all columns covered by essential primes
- Find minimum set of rows that cover the remaining columns



		1	4	6	7	8	9	10	11	15
✓ x'y'z	1, 9	X					X			
✓ w'xz'	4, 6		X	X						
w'xy	6, 7			X	X					
xyz	7, 15				X					X
wyz	11, 15								X	X
✓ wx'	8, 9, 10, 11					X	X	X	X	
		✓	✓	✓		✓	✓	✓	✓	

Prime implicant chart



		1	4	6	7	8	9	10	11	15
✓ $x'y'z$	1, 9	X					X			
✓ $w'xz'$	4, 6		X	X						
$w'xy$	6, 7			X	X					
xyz	7, 15				X					X
wyz	11, 15								X	X
✓ wx'	8, 9, 10, 11					X	X	X	X	
		✓	✓	✓		✓	✓	✓	✓	

the table shows that the selection of essential prime implicantscovers all the mintermsof the function except 7 and 15.

these two mintermsmust be included by the selection of one or more prime implicants.

Here xyz covers both mintermsand hence one to be selected.

We have thus found the following minimum set of prime implicantswhose sum gives the required minimized function.

$$F = x'y'z + w'xz' + wx' + xyz$$

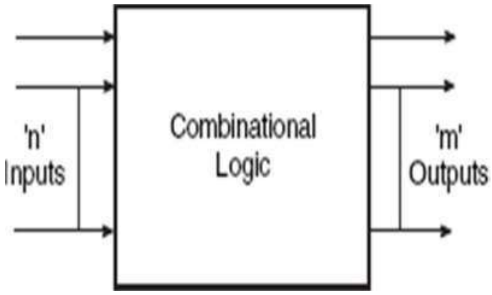
MODULE II

Combinational Circuits:

Combinational circuit is a circuit in which we combine the different gates in the circuit, The term combinational comes to us from mathematics. In mathematics a combination is an unordered set, which is a formal way to say that nobody cares which order the items came in. Most games work this way, if you rolled dice one at a time and get a 2 followed by a 3 it is the same as if you had rolled a 3 followed by a 2. With combinational logic, the circuit produces the same output regardless of the order the inputs are changed. There are circuits which depend on the when the inputs change, these circuits are called sequential logic. Even though you will not find the term sequential logic in the chapter titles, the next several chapters will discuss sequential logic. Practical circuits will have a mix of combinational and sequential logic, with sequential logic making sure everything happens in order and combinational logic performing functions like arithmetic, logic, or conversion.

Design Using Gates:

A combinational circuit is one where the output at any time depends only on the present combination of inputs at that point of time with total disregard to the past state of the inputs. The logic gate is the most basic building block of combinational logic. The logical function performed by a combinational circuit is fully defined by a set of Boolean expressions. The other category of logic circuits, called sequential logic circuits, comprises both logic gates and memory elements such as flip-flops. Owing to the presence of memory elements, the output in a sequential circuit depends upon not only the present but also the past state of inputs.



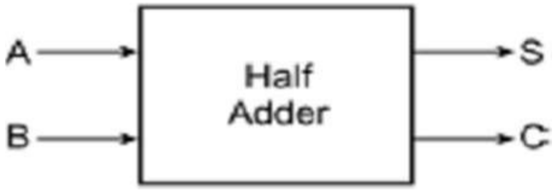
Generalized Combinational Circuit

This Figure shows the block schematic representation of a generalized combinational circuit having n input variables and m output variables or simply outputs. Since the number of input variables is n, there are 2n possible combinations of bits at the input. Each output can be expressed in terms of input .

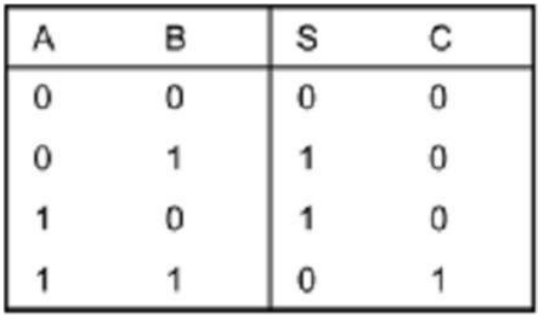
Adder

Half-Adder Circuit

A half-adder is an arithmetic circuit block that can be used to add two bits. Such a circuit thus has two inputs that represent the two bits to be added and two outputs, with one producing the SUM output and the other producing the CARRY. Figure shows the truth table of a half- adder, showing all possible input combinations and the corresponding outputs.



Block Diagram of Half Adder Circuit



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

Truth Table of Half Adder

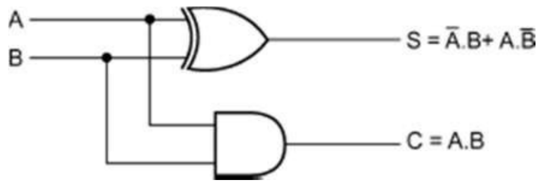
The Boolean expressions for the SUM and CARRY outputs are given by the equations below-

SUM S= A.B++ A.B

CARRY C = A.B

An examination of the two expressions tells that there is no scope for further simplification. While the first one representing the SUM output is that of an EX-OR gate, the second one representing the CARRY output is that of an AND gate. However, these two expressions can certainly be represented in different forms using various laws and theorems of Boolean algebra to illustrate the flexibility that the designer has in hardware- implementing as simple a combinational function as that of a half-adder.

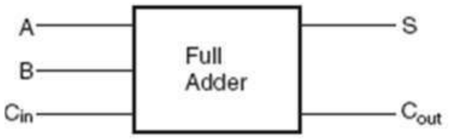
Logic Implementation of Half Adder

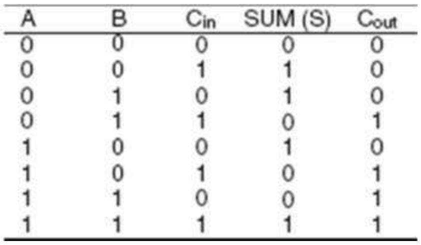


Although the simplest way to hardware-implement a half-adder would be to use a two-input EX-OR gate for the SUM output and a two-input AND gate for the CARRY output, as shown in Figure, it could also be implemented by using an appropriate arrangement of either NAND or NOR gates.

Full Adder Circuit

A full adder circuit is an arithmetic circuit block that can be used to add three bits to produce a SUM and a CARRY output. Such a building block becomes a necessity when it comes to adding binary numbers with a large number of bits. The full adder circuit overcomes the limitation of the half-adder, which can be used to add two bits only. Let us recall the procedure for adding larger binary numbers. We begin with the addition of LSBs of the two numbers. We record the sum under the LSB column and take the carry, if any, forward to the next higher column bits. As a result, when we add the next adjacent higher column bits, we would be required to add three bits if there were a carry from the previous addition. We have a similar situation for the other higher column bits. Also until we reach the MSB. A full adder is therefore essential for the hardware implementation of an adder circuit capable of adding larger binary numbers. A half-adder can be used for addition of LSBs only. Figure shows the truth table of a full adder circuit showing all possible input combinations and corresponding outputs. In order to arrive at the logic circuit for hardware implementation of a full adder, we will firstly write the Boolean expressions for the two output variables, that is, the SUM and
CARRY outputs, in terms of input variables. These expressions are then simplified





A	B	C_{in}	SUM (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

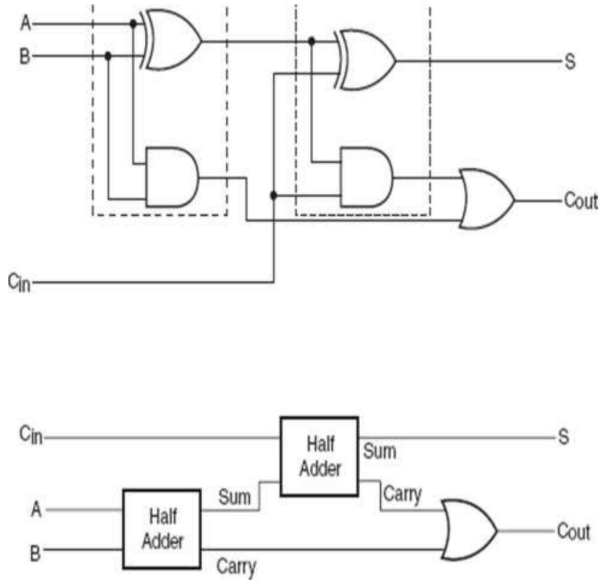
Block Diagram of Full Adder Circuit and Truth Table of Full Adder:

by using any of the simplification techniques described in the previous chapter. The Boolean expressions for the two output variables are given in Equation below for the SUM output (S) and in above Equation for the CARRY output (Cout):

$$S = \overline{A}.\overline{B}.C_{in} + \overline{A}.B.\overline{C}_{in} + A.\overline{B}.\overline{C}_{in} + A.B.C_{in}$$

$$C_{out} = \overline{A}.B.C_{in} + A.\overline{B}.C_{in} + A.B.\overline{C}_{in} + A.B.C_{in}$$

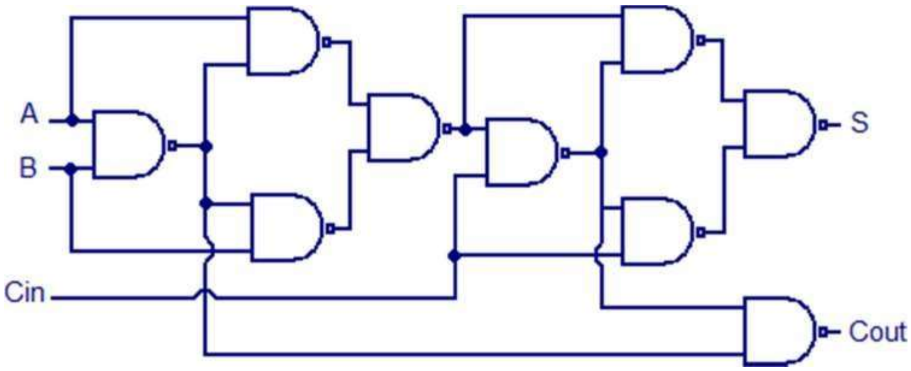
The next step is to simplify the two expressions. Logic circuit diagram of full adder Boolean expression above can be implemented with a two-input EX-OR gate provided that one of the inputs is Cin and the other input is the output of another two-input EX-OR gate with A and B as its inputs. Similarly, Boolean expression above can be implemented by ORing two minterms. One of them is the AND output of A and B. The other is also the output of an AND gate whose inputs are Cin and the output of an EX-OR operation on A and B. The whole idea of writing the Boolean expressions in this modified form was to demonstrate the use of a half-adder circuit in building a full adder. In the above it shows logic implementation of Equations. However, a single full adder circuit can be used to add one-bit binary numbers only. A cascade arrangement of these adders can be used to construct adders capable of adding binary numbers with a larger number of bits. For example, a four-bit binary adder would require four full adders of the type shown in Figure to be connected in cascade. After minimization we get the following logic circuit-



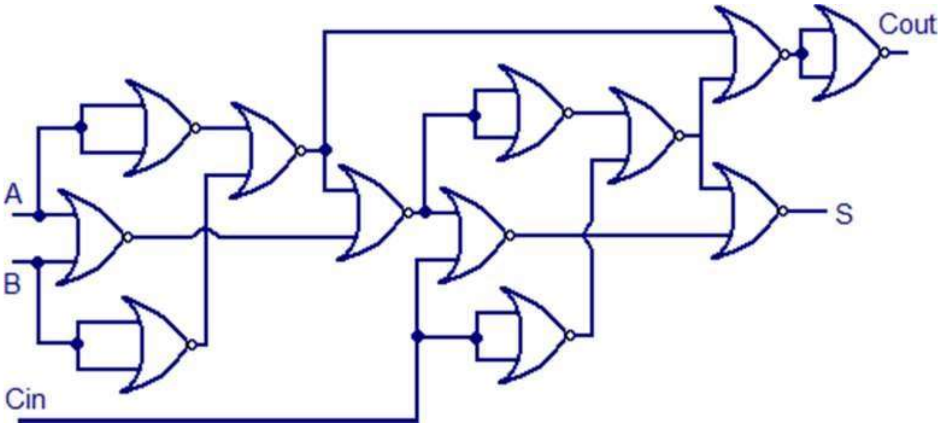
Logic circuit of Full Adder

Full adder using NAND or NOR logic

Alternatively the full adder can be made using NAND or NOR logic. These schemes are universally accepted and their circuit diagrams are shown below.



Full adder using NAND Gates

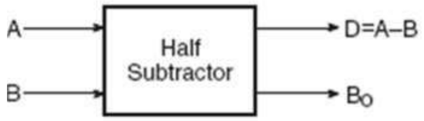


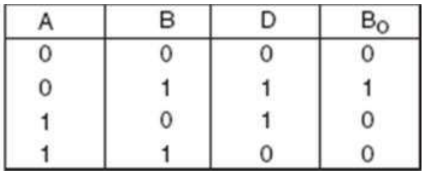
Full adder using NOR Gates

Half-Subtractor Circuit

We will study the use of adder circuits for subtraction operations in the following pages. Before we do that, we will briefly look at the counterparts of half-adder and full adder circuits in the half-subtractor and full subtractor for direct implementation of subtraction operations using logic gates. A halfsubtractor is a combinational circuit that can be used to subtract one binary digit from another to produce a

DIFFERENCE





A	B	D	B_O
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

output and a BORROW output.

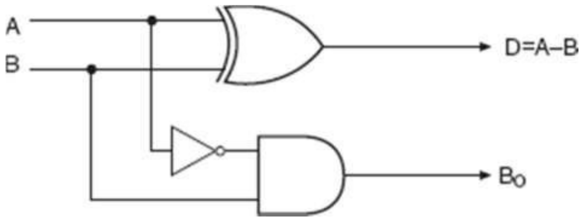
Block Diagram of Half Subtractor Circuit and Truth Table of Half Subtractor:

The BORROW output here specifies whether a '1' has been borrowed to perform the subtraction. The truth table of a half-subtractor, as shown in Figure, explains this further. The Boolean expressions for the two outputs are given by the equations

$$D = \overline{A}.B + A.\overline{B}$$

$$B_0 = \overline{A}.B$$

It is obvious that there is no further scope for any simplification of the Boolean expressions given by above equations. While the expression for the DIFFERENCE (D) output is that of an EX-OR gate, the expression for the BORROW output (Bo) is that of an AND gate with input A complemented before it is fed to the gate. Figure shows the logic implementation of a half-subtractor. Comparing a half-subtractor with a half-adder, we find that the expressions for the SUM and DIFFERENCE outputs are just the same. The expression for BORROW in the case of the half-subtractor is also similar to what we have for CARRY in the case of the half-adder. If the input A, that is, the minuend, is complemented, an AND gate can be used to implement the BORROW output.

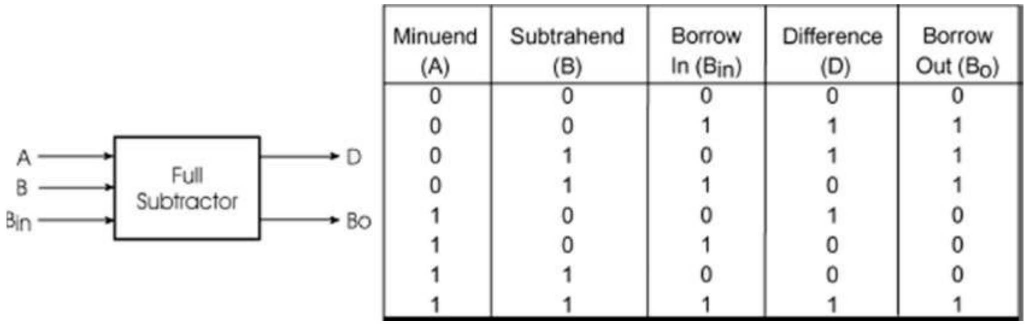


Logic Diagram of a Half Subtractor

It is obvious that there is no further scope for any simplification of the Boolean expressions given by above equations. While the expression for the DIFFERENCE (D) output is that of an EXOR gate, the expression for the BORROW output (Bo) is that of an AND gate with input A complemented before it is fed to the gate. This Figure shows the logic implementation of a halfsubtractor. Comparing a half- subtractor with a half-adder, we find that the expressions for the SUM and DIFFERENCE outputs are just the same. The expression for BORROW in the case of the halfsubtractor is also similar to what we have for CARRY in the case of the half-adder. If the input A, that is, the minuend, is complemented, an AND gate can be used to implement the BORROW output.

Full Subtractor Circuit

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 Bin . There are two outputs, namely the DIFFERENCE output D and the BORROW output Bo.



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

Block Diagram of Full Subtractor Circuit and Truth Table of Full Subtractor:

The BORROW output bit tells whether the minuend bit needs to borrow a '1' from the next possible higher minuend bit. Figure shows the truth table of a full subtractor. The Boolean expressions for the two output variables are given by the equations-

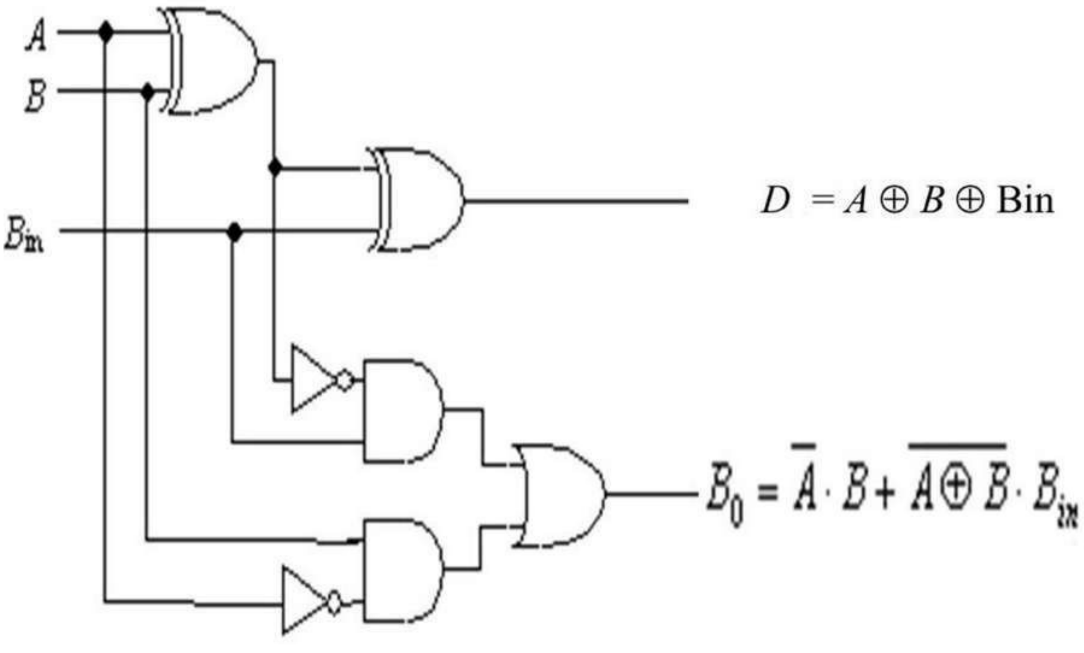
$$D = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + A.\overline{B}.\overline{B}_{in} + A.B.B_{in}$$

$$B_o = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + \overline{A}.B.B_{in} + A.B.B_{in}$$

After minimizing the above expressions we get-

$$D = A \oplus B \oplus \mathrm{Bin}$$

$$B_0 = \overline{A} \cdot B + \overline{A \oplus B} \cdot B_{in}$$


A
B
Bin
$D = A \oplus B \oplus \text{Bin}$
$B_0 = \bar{A} \cdot B + \overline{A \oplus B} \cdot B_{in}$

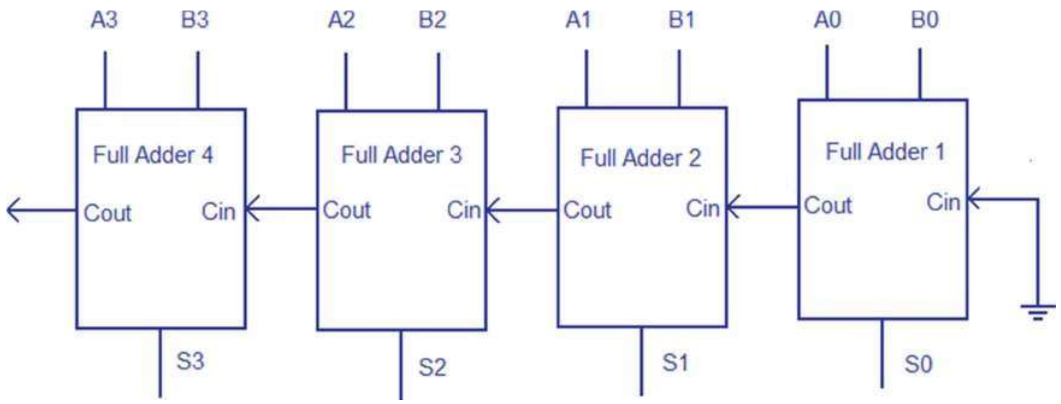
Logic circuit of Full Subtractor

N-Bit Parallel Adder

The Full Adder is capable of adding only two single digit binary number along with a carry input. But in practical we need to add binary numbers which are much longer than just one bit. To add two nbit binary numbers we need to use the n-bit parallel adder. It uses a number of full adders in cascade. The carry output of the previous full adder is connected to carry input of the next full adder.

4 Bit Parallel Adder

In the block diagram, A 0 and B 0 represent the LSB of the four bit words A and B. Hence Full Adder- 0 is the lowest stage. Hence its Cin has been permanently made 0. The rest of the connections are exactly same as those of n-bit parallel adder is shown in fig. The four bit parallel adder is a very common logic circuit.



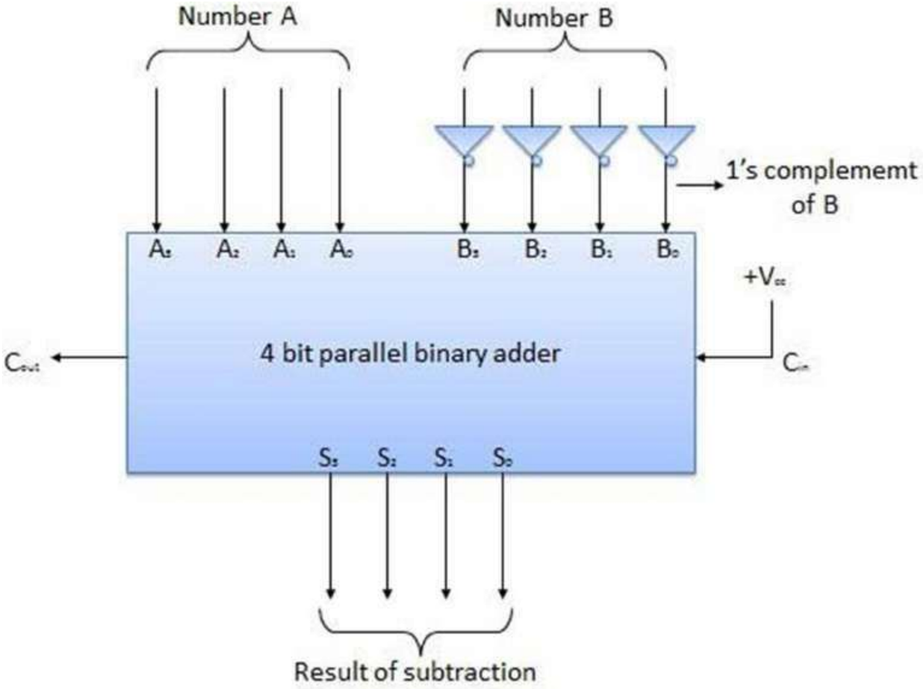
Block diagram of 4 Bit Parallel Adder

N-Bit Parallel Subtractor

The subtraction can be carried out by taking the 1's or 2's complement of the number to be subtracted. For example we can perform the subtraction A − B by adding either 1's or 2's complement of B to A. That means we can use a binary adder to perform the binary subtraction.

4 Bit Parallel Subtractor

The number to be subtracted B is first passed through inverters to obtain its 1's complement. The 4-bit adder then adds A and 2's complement of B to produce the subtraction. S3 S2 S1 S0 represents the result of binary subtraction A − B and carry output Cout represents the polarity of the result. If A > B then Cout = 0 and the result of binary form A − B then Cout = 1 and the result is in the 2's complement form.



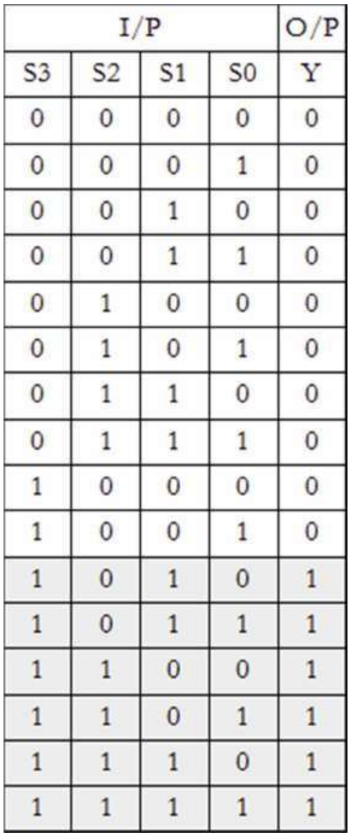
Block diagram of 4 Bit Parallel Subtractor

BCD Adder:

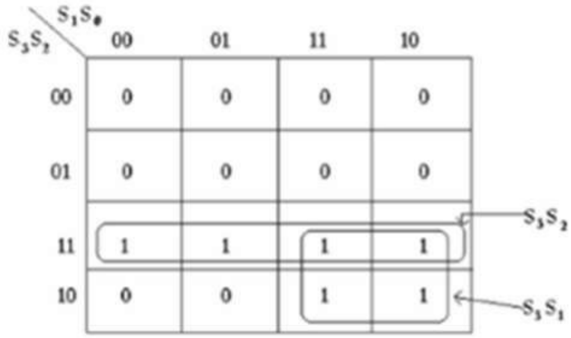
BCD binary numbers represent Decimal digits 0 to 9. A 4-bit BCD code is used to represent the ten numbers 0 to 9. Since the 4-bit Code allows 16 possibilities, therefore the first 10 4-bit combinations are considered to be valid BCD combinations. The latter six combinations are invalid and do not occur.BCD Code has applications in Decimal Number display Systems such as Counters and Digital Clocks. BCD Numbers can be added together using BCD Addition. BCD Addition is similar to normal Binary Addition except for the case when sum of two BCD digits exceeds 9 or a Carry is generated. When the Sum of two BCD numbers exceeds 9 or a Carry is generated a 6 is added to convert the invalid number into a valid number. The carry generated by adding a 6 to the invalid BDC digit is passed on to the next BCD digit. Addition of two BCD digits requires two 4-bit Parallel Adder Circuits. One 4-bit Parallel Adder adds the two BCD digits. A BCD Adder uses a circuit which checks the result at the output of the first adder circuit to determine if the result has exceeded 9 or a Carry has been generated. A BCD adder adds two BCD digits and produces output as a BCD digit. A BCD or Binary Coded Decimal digit cannot be greater than 9.
So briefly we can say-

- The two BCD digits are to be added using the rules of binary addition. If sum is less than or equal to 9 and carry is 0, then no correction is needed. The sum is correct and in true BCD form.
- But if sum is greater than 9 or carry =1, the result is wrong and correction must be done. The wrong result can be corrected adding six (0110) to it.
- For implementing a BCD adder using a binary adder circuit IC 7483, additional combinational circuit will be required, where the Sum output S_3-S_0 is checked for invalid values from 10 to 15. The truth table and K-map for the same is as shown:

Truth table for BCD Numbers

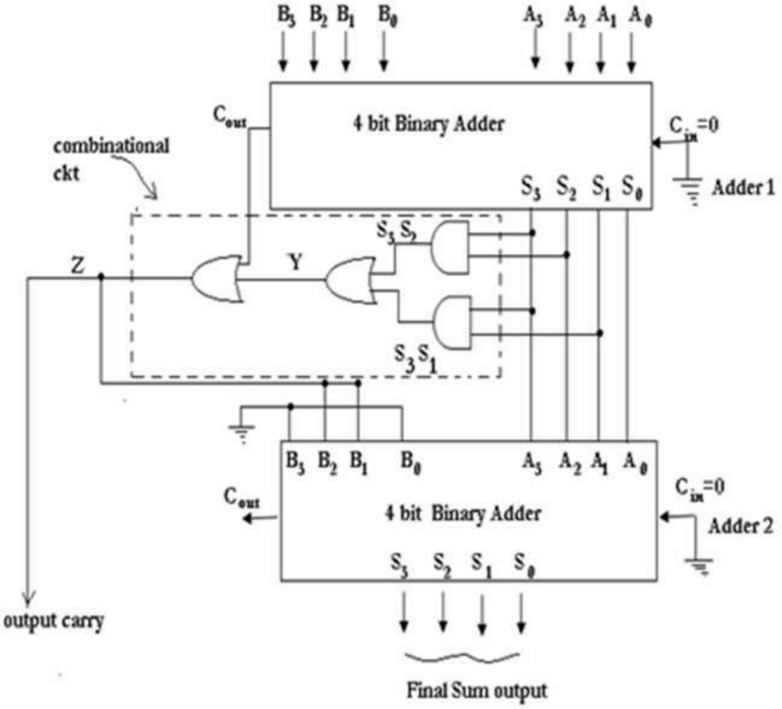


I/P				O/P
S3	S2	S1	S0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



S_3S_2 \ S_1S_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

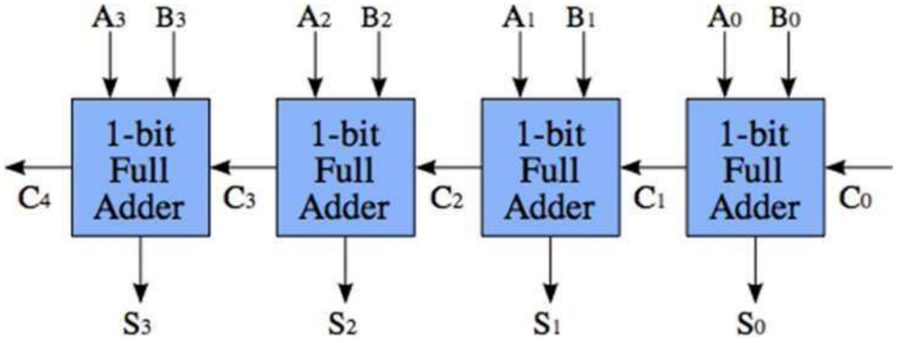
K- Map for the above Truth table



Logic circuit of BCD Adder

Carry-look ahead Adder:

A system of ripple-carry adders is a sequence of standard full adders that makes it possible to add numbers that contain more bits than that of a single full adder. Each full adder has a carry in (Cin) and a carryout (Cout) bit, and the adders are connected by connecting Cout on step k to Cin on step k+1 (Cin on step 0 is C0 in the Figure, Cout on step 3 is C4 in the Figure)

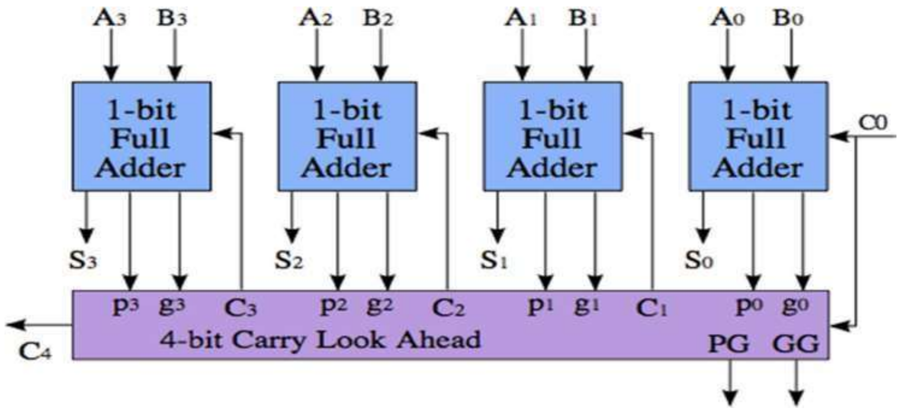


The challenge with ripple-carry adders, is the propagation delay of the carry bits. Assume that, in an instant the values of A and B change, such that

A1 = 0
B1 = 1 A0
= 1
B0 = 1

Since A0 and B0 are high, the first full adder will produce a carry, i. e. C1 = 1. However, it takes some time for the logic to settle down, so C1 doesn't change until a little after A1 and B1 changed. Thus, before C1 shows up, the second full adder does not produce a carry, but as C1 shows up, the second adder will recompute and produce a carry, i. e. C2 = 1. In the worst case, C4 is not correctly computed until 4*propagation delay, and Cn is not computed until n*propagation delay.

A carry-lookahead adder system solves this problem, by computing whether a carry will be generated before it actually computes the sum. There are multiple schemes of doing this, so there is no "one" circuit that constitutes a look-ahead adder. The idea is something like this:



Logic circuit of 4 bit Carry-lookahead Adder

The calculation of C4 is no faster than in the the ripple-carry above, nor is PG and GG - the magic only happens when you put several of these blocks together to add even larger numbers.

The important to note part of the picture, is that the purple block is producing three values: C4, PG (Propagate) and GG (Generate). PG goes high if this block will propagate Cin to Cout, and GG goes high if the block will generate an overflow regardless of Cin. (Also, the block may neither propegate nor generate a carry, in which case both PG and GG are low, and Cout is 0.) PG and GG can be calculated in the purple block regardless of the value of C0 - thus, when C0 finally arrives, the purple block can simply consult its previously calculated result, and if the result is a "propagate," then C0 is propagated directly to C4; this is four times faster than propagating through all the four full adders.

Assignment:

1. Draw the Half Subtractor circuit using two input NOR gate.
2. Draw the Half Subtractor circuit using two input NOR gate.
3. Draw a logic circuit that has three inputs A, B and C and whose output will be high only when majority inputs are high.
4. (A) What is the number of carry-lookahead levels L in a 128-bit carry- lookahead adder using the same component types are in Figure 2? (B) How many carry-lookahead circuits are required?

   (c) What is the estimated maximum delay through the 128-bit adder assuming that each gate delay is 0.20 ns?

Model Questions:

1. A binary parallel adder produces arithmetic sum in
   a. serial
   b. parallel
   c. sequence
   d. both a and b
2. Sum of two n-bit binary numbers can be done
   a. serially
   b. parallely
   c. sequentially
   d. both a and b
3. Full adder forms sum of
   a. 2bits
   b. 3bits
   c. 4bits
   d. 5bits
4. Subtraction of binary numbers can be done conveniently with
   a. high cost circuit
   b. low cost circuits
   c. complements
   d. borrows
5. The sum of a Full adder can be represented by
   a. 2 input X-OR gate
   b. 3 input NAND gate
   c. 2 input NAND gate
   d. 3 input X-OR gate

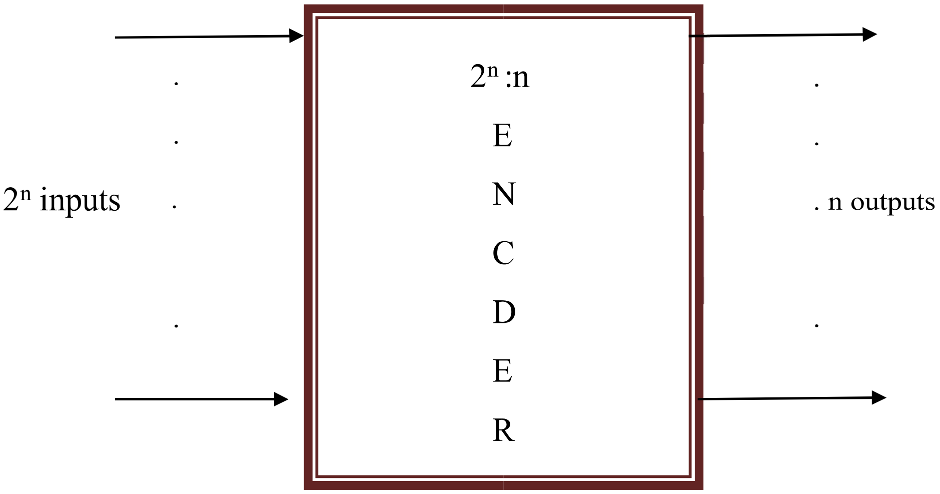
6. Draw the circuit diagram of Half Adder using NAND gate only. Describe the operation performed by Half adder.
7. Draw a block diagram of a full adder labeling all the inputs and outputs. Write the truth table for full adder.
8. Draw the circuit diagram of Half Subtractor using NAND gate only. Describe its operation.
9. Draw a block diagram of a full Subtractor labeling all the inputs and outputs. Write the truth table for full Subtractor.

19. Draw the Half Adder circuit using two input NOR gate.

11. Draw the Full Adder circuit using two input NOR gate.

Encoder:

It is a circuit which accepts an active level(high level or low level) from one of its 2^n or less (in some cases) inputs which representing a digit and converts it to a n bit coded output. For an example 2 to 1,4 to 2,8 to 3 also called octal to binary,10 to 4 also called decimal to BCD,16 to 4 encoder etc.It can encode symbols or characters also.



Block diagram of encoder

8 to 3(Octal to Binary) Encoder Design:

It accepts 8 inputs & produces 3 bit coded output according to the activated input. Here active high logic is used for activated input.

Truth Table

Input								Output		
D0	D1	D2	D3	D4	D5	D6	D7	A	B	C
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

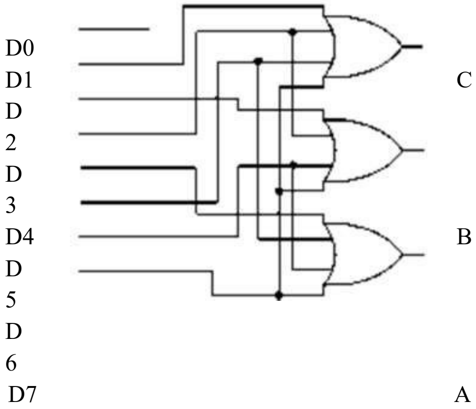
Output Expression

A=D4+D5+D6+D7

B=D2+D3+D6+D7

C=D1+D3+D5+D7

Circuit Diagram



Circuit of 8 to 3(octal to binary) encoder

Decimal to BCD(10 to 4) Encoder:
It accepts 10 inputs & produces 4 bit coded output according to the activated input. Here active high logic is used for activated input.

Truth Table

Input										Output			
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	A	B	C	D
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

Output

Expression

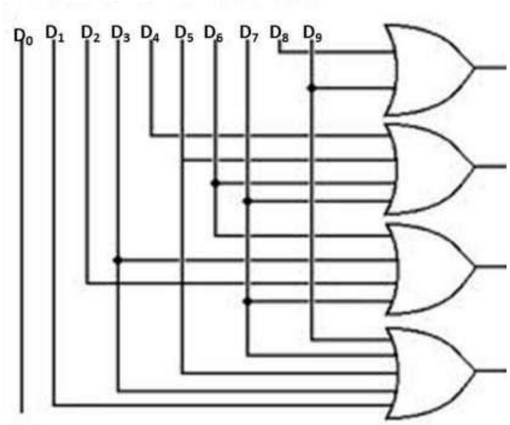
$A=D_8+D_9$

$B=D_4+D_5+D_6+D_7$

$C=D_2+D_3+D_6+D_7$

$D= D_1+D_3+D_5+D_7+D_9$

Circuit Diagram



A

B

C

D

Circuit of decimal to BCD(10 to 4) encoder

Application

- Analog to digital converter
- Used in transmitter of communication system

Decoder:

It is a logic circuit that converts n bit binary input data(code) into a maximum of 2^n output lines.Here each output line will be activated only for one possible combinations of inputs. For an example 1 to 2(1:2),2 to 4(2:4),3 to 8(3:8) also called binary to octal,4 to 10(4:10) also called BCD to decimal,4 to 16 (4:16) decoder etc.

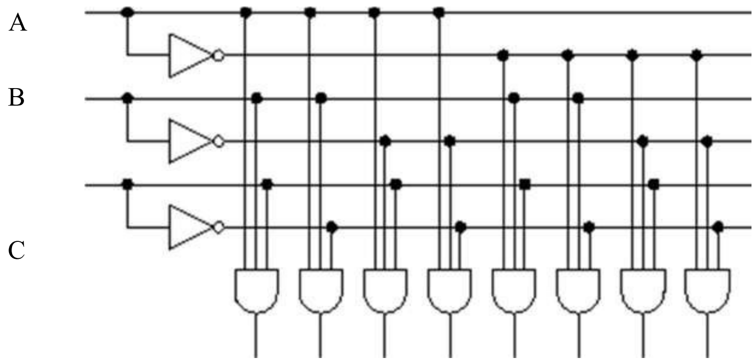
3 to 8 decoder design:
In this circuit only one of eight output lines is high for a particular input combination. As the inputs represent 3 bit binary numbers and the outputs represent eight digits in the octal no.

system, so it is also called binary to octal decoder.

Truth Table

Input			Output							
A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_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

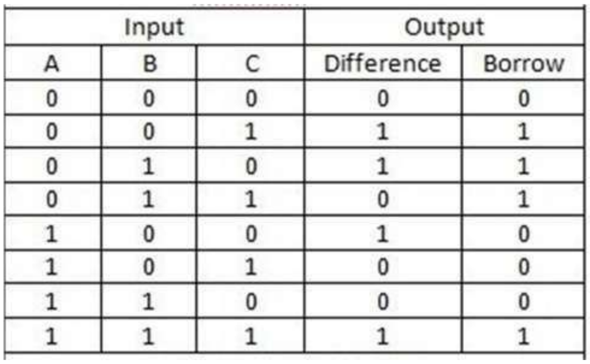
Circuit Diagram



Boolean Function Representation using

Decoder: Full Subtractor using decoder:

Truth table of full subtractor

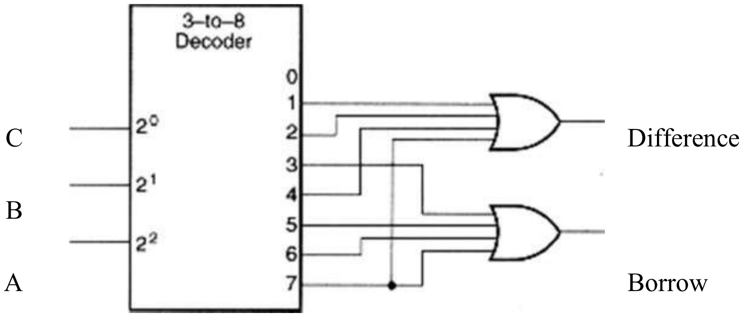


Input			Output	
A	B	C	Difference	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Expression

Difference=∑m(1,2,4,7)
Borrow=∑m(1,2,3,7)

Circuit Design



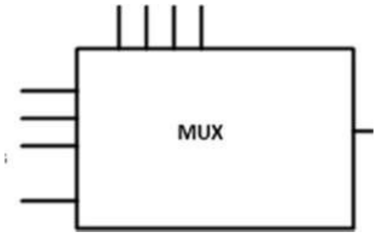
Multiplexer(MUX):

It is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. Selection of a particular input line(called data input) is controlled by a set of control lines(called select input). A multiplexer is denoted as 2^n :1 MUX where n is no. of select input and 2^n is no. of data input. For an example 2 to 1(2:1),4 to 1(4:1),8 to 1(8:1),16 to 1(16:1) MUX etc.

Block Diagram

n select input

n data lines single output line



2:1 Multiplexers Design:

- 2 data input I_0, I_1□
- 1 select lines- S_0□
- Single output line-Y□

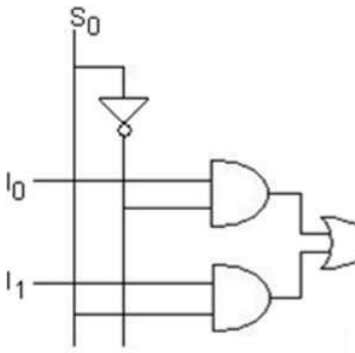
Truth Table

Data Input	Select Input S_0	Output Y
I_0	0	I_0
I_1	1	I_1

Expression

$Y=S_0'I_0+ S_0I_1$

Circuit Diagram



4:1 Multiplexers Design:

- 4 data input I_0, I_1, I_2, I_3□

- 2 select lines- S_1,S_0□
- Single output line-

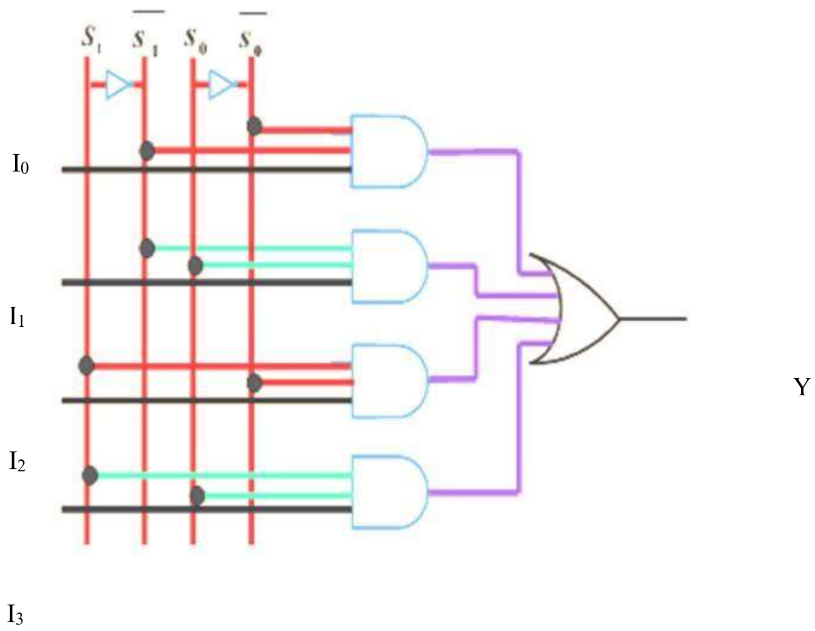
Y Truth Table□

Data Input	Select Input		Output Y
	S_1	S_0	
I_0	0	0	I_0
I_1	0	1	I_1
I_2	1	0	I_2
I_3	1	1	I_3

Expression

$Y=S_0' S_1' I_0+ S_0 S_1' I_1+ S_0' S_1 I_2+ S_0 S_1 I_3$

Circuit Diagram



Boolean Function Implementation Using MUX:

- One variable is chosen for input lines and rest of the term for select lines.□
- The min terms with the variable selected for input are listed in two rows as implementation table□
- All the min terms which are present in the function are circled□ o If no circled variable in any column, then 0 is applied to the corresponding input line o If both circled variables in any column,

then 1 is applied to the corresponding input line o If bottom variable is circled and top is not circled, original variable is applied to the corresponding input line

- If bottom variable is not circled and top is circled, then complemented variable is applied to the corresponding input line

Examples:

Implement the function F(A, B, C, D)= Σ (1, 2, 5, 7, 9, 14) using MUX

Let, take the variable A for input lines and B, C & D for selection lines.

No. of variable N=4 ,

$2^{N-1} = 2^3$ => 8:1 MUX is required

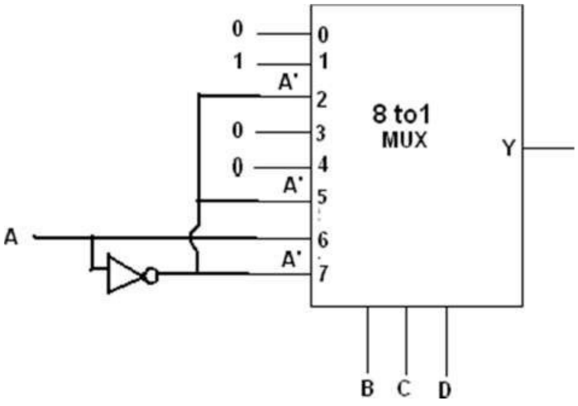
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
0	1	A'	0	0	A	A'	A

Implementation Table

A'

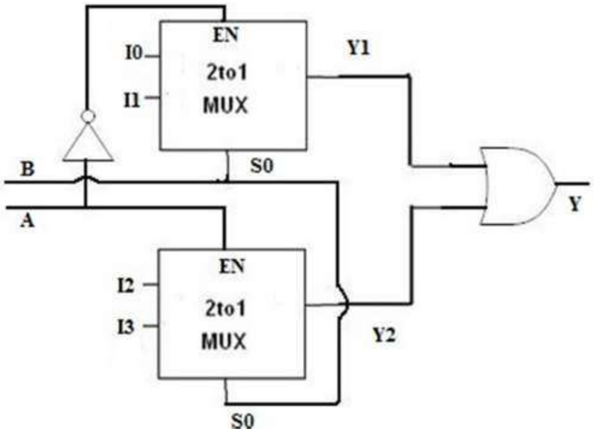
A

Designed Circuit



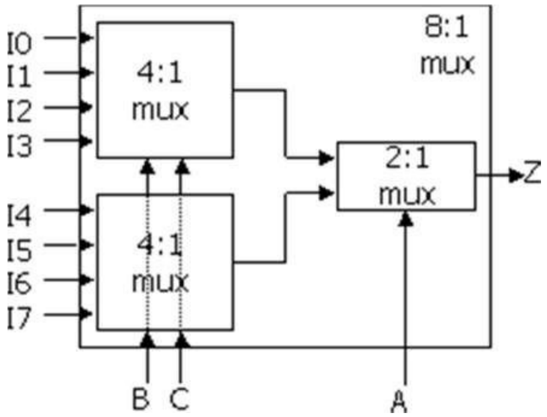
Cascading of MUX:

4:1 MUX using 2:1 MUX and OR gate:

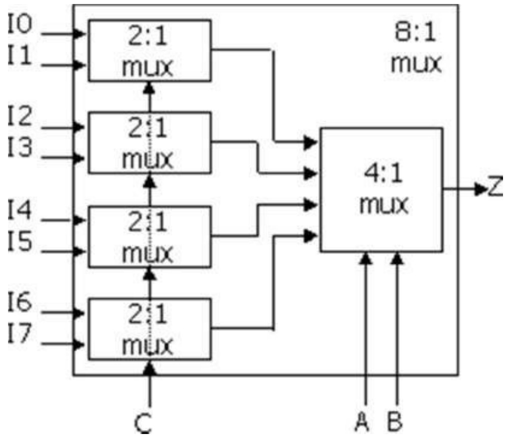


8:1 MUX using 4:1 MUX & 2:1 MUX:

- Using two 4:1 MUX and one 2:1 MUX



- Using one 4:1 MUX and four 2:1 MUX(Alternate Circuit)



Demultiplexer:

A demultiplexer (or demux) takes the input from single data input line and directs it to one of several digital output lines. A demultiplexer with n select lines has 2^n outputs. It is also called a data distributor.

1 to 4 demultiplexer:

- One data input D
- 2 select lines (S0, S1)
- 4 outputs (Y0 - Y3)

Truth table

Data Input	Select Input		Y3	Y2	Y1	Y0
	S1	S0				
D	0	0	0	0	0	D
D	0	1	0	0	D	0
D	1	0	0	D	0	0
D	1	1	D	0	0	0

Expression

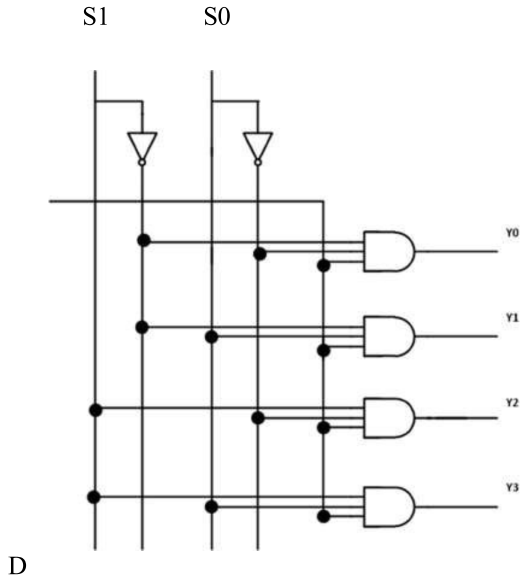
Y0=S0'S1'D

Y1= S0S1'D

Y2= S0'S1D

Y3= S0S1D

Circuit Diagram



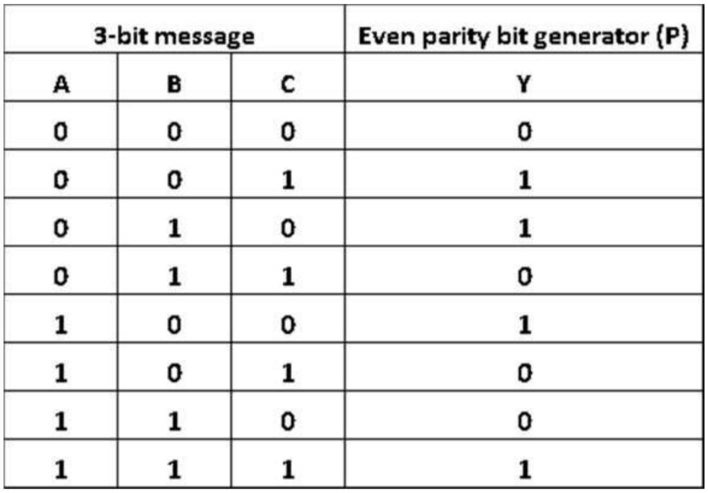
Parity Generator & Checker Circuit:

In digital systems, for the data transmission due to noise the data can alter: 0 to 1 and 1 to 0. The parity generation and detection technique is a widely used single bit error detection techniques. Here, parity bit is added with the original data sequence to make number of one s either even or odd. The message containing the data bits along with parity bit is transmitted. So,parity generator is a combinational circuit that generates the parity bit in the transmitter, whereas parity checker a combinational circuit that checks the parity bit in the receiver . In even parity, the total number of ones in transmitted data stream is even including the added parity bit and in odd parity the total number of ones in transmitted data stream is odd including the added parity bit.

3 bit Even Parity Generator:

Here a 3-bit message is to be transmitted with even parity bit. Let the three inputs are A,B and C and the parity bit is P.

Truth Table



3-bit message			Even parity bit generator (P)
A	B	C	Y
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

From the above truth table, the simplified expression of the parity bit can be written as

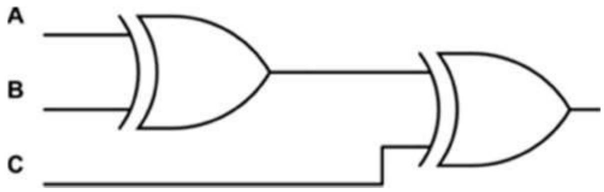
$$P = \bar{A}\,\bar{B}\,C + \bar{A}\,B\,\bar{C} + A\,\bar{B}\,\bar{C} + A\,B\,C$$

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

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

$$P = A \oplus B \oplus C$$

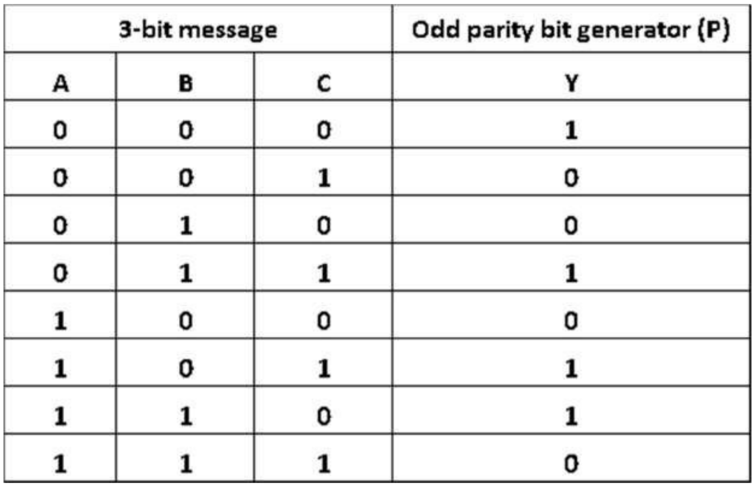
Circuit Diagram



P(Y)

Odd Parity Generator:

Here a 3-bit message is to be transmitted with odd parity bit. Let the three inputs are A,B and C and the parity bit is P.

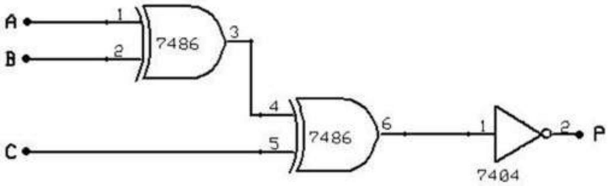


3-bit message			Odd parity bit generator (P)
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

The output parity bit expression for this generator circuit is obtained as

P = A x-nor B x-nor C

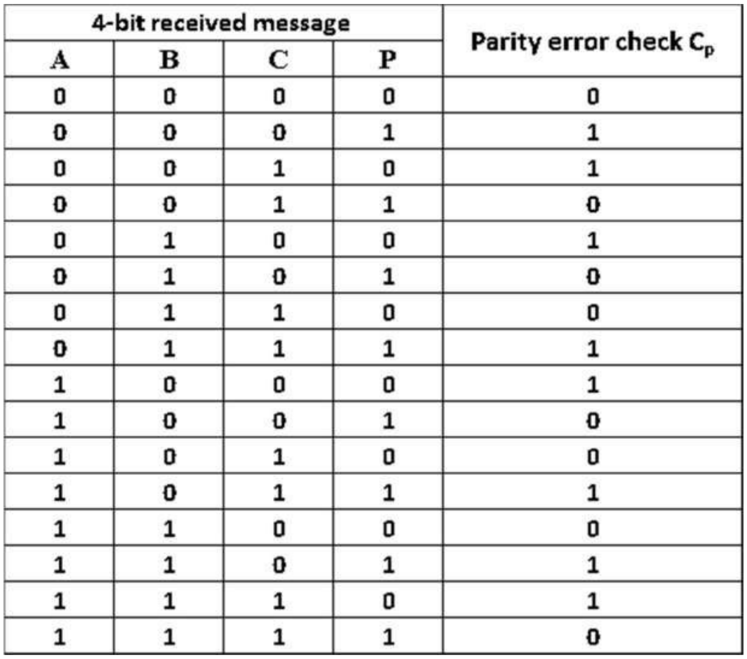
Circuit Diagram



Even Parity Checker:

Let three input bit along with even parity bit is generated at the transmitting end. These 4 bits are applied as input to the parity checker circuit which checks the possibility of error on the data. Since the data is transmitted with even parity, four bits received at circuit must have an even number of 1s.If any error occurs, the received message consists of odd number of 1s. The output is denoted by PEC (parity error check), PEC = 1 i.e. the four bits received have odd number of 1s and PEC = 0 i.e.the 4-bit message has even number of 1s.

Truth Table



4-bit received message				Parity error check C_p
A	B	C	P	
0	0	0	0	0
0	0	0	1	1
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	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Expression

Let P=D,then

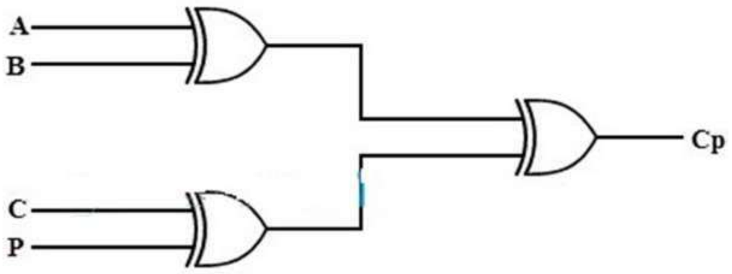
$$PEC = \bar{A}\ \bar{B}\,(\bar{C}\,D + C\,\bar{D}) + \bar{A}\,B\,(\bar{C}\,\bar{D} + C\,D) + A\,B\,(\bar{C}\,D + C\,\bar{D}) + A\,\bar{B}\,(\bar{C}\,\bar{D} + C\,D)$$

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

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

$$= (A \oplus B) \oplus (C \oplus D)$$

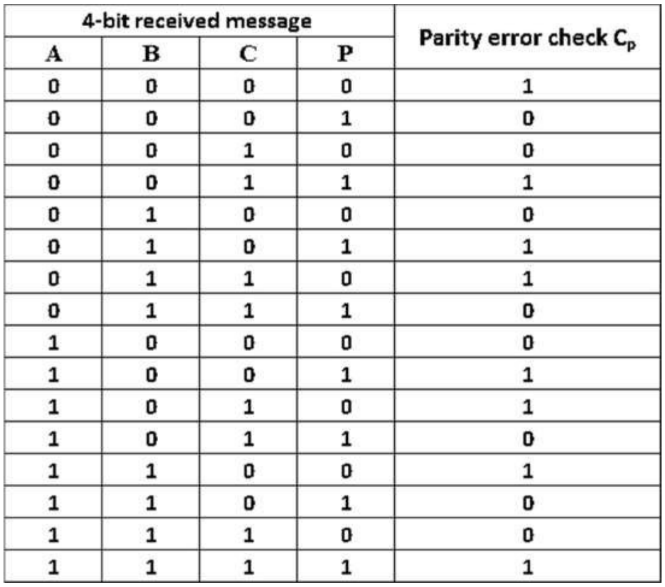
Circuit Diagram



Odd Parity Checker:

Let three bit message along with odd parity bit is transmitted at the transmitting end. Odd parity checker circuit receives these 4 bits and checks whether any error are present in the data. If the total number of 1s in the data is odd, then it indicates no error, whereas if the total number of 1s is even then it indicates the error since the data is transmitted with odd parity at transmitting end.PEC =1 if the 4-bit message received consists of even number of 1s and PEC= 0 if the message contains odd number of 1s .

Truth Table



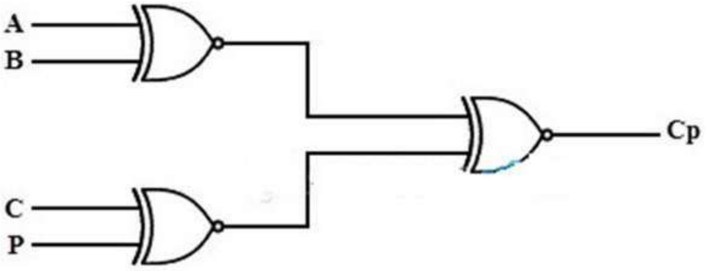
4-bit received message				Parity error check C_p
A	B	C	P	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Expression

Let P=D,then

PEC = (A Ex-NOR B) Ex-NOR (C Ex-NOR D)

Circuit Diagram



Magnitude Comparator:

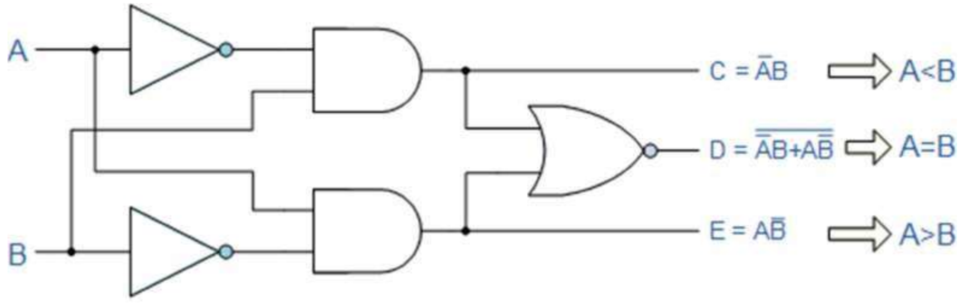
A Magnitude Comparator is a digital comparator which has three output terminals, one each for equality, greater than and less than. For example, a single bit magnitude comparator two (A and B) inputs would produce the following three output conditions when compared to each other:A=B,A>B,A<B.

Single Bit Magnitude Comparator:

Truth Table

Input		Output		
B	A	A > B	A = B	A < B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Circuit Diagram

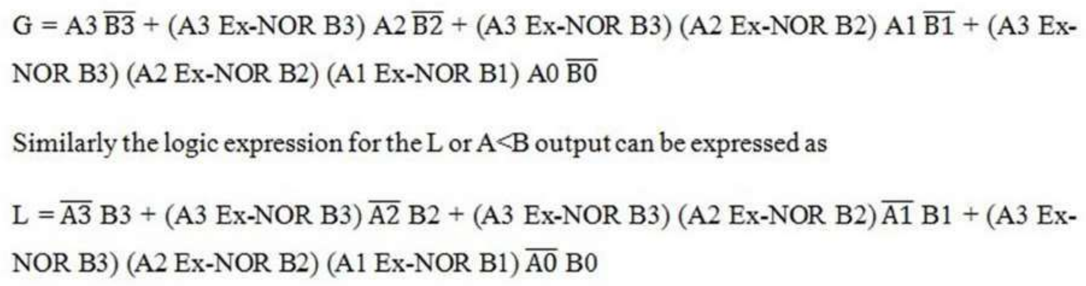


4- Bit Magnitude Comparator:

It can be used to compare two four-bit words. The two 4-bit numbers are A = A3 A2 A1 A0 and B3 B2 B1 B0 where A3 and B3 are the most significant bits.

The output logic statements of this converter are
If A3 = 1 and B3 = 0, then A>B or if A3 and B3 are equal, and if A2 = 1 and B2 = 0, then A > B or if A3 and B3 are equal & A2 and B2 are equal, and A1 = 1, and B1 = 0, then A>B or if A3 and B3 are equal, A2 and B2 are equal and A1 and B1 are equal, and A0 = 1 and B0 = 0, then A > B.

From the above statements, the output A > B logic expression can be written as



$G = A3\,\overline{B3}$ + (A3 Ex-NOR B3) $A2\,\overline{B2}$ + (A3 Ex-NOR B3) (A2 Ex-NOR B2) $A1\,\overline{B1}$ + (A3 Ex-NOR B3) (A2 Ex-NOR B2) (A1 Ex-NOR B1) $A0\,\overline{B0}$

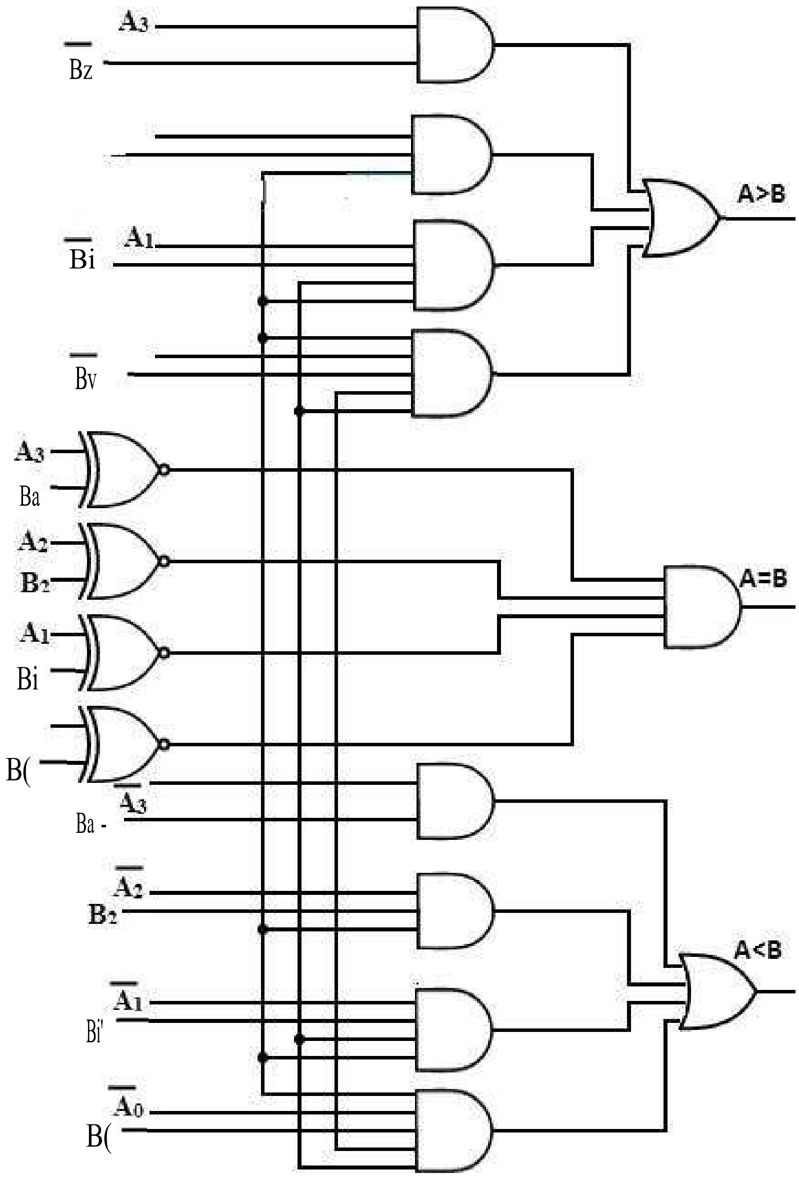
Similarly the logic expression for the L or A<B output can be expressed as

$L = \overline{A3}\,B3$ + (A3 Ex-NOR B3) $\overline{A2}\,B2$ + (A3 Ex-NOR B3) (A2 Ex-NOR B2) $\overline{A1}\,B1$ + (A3 Ex-NOR B3) (A2 Ex-NOR B2) (A1 Ex-NOR B1) $\overline{A0}\,B0$

The equal output is produced when all the individual bits of one number are exactly coincides with corresponding bits of another number. Then the logical expression for A=B output can be written as

E = (A3 Ex-NOR B3) (A2 Ex-NOR B2) (A1 Ex-NOR B1) (A0 Ex-NOR B0)

From the above output Boolean expressions, the logic circuit for this comparator can be implemented as given below:



Multiple Choice Question:

1. In 16:1 MUX the no. of select lines are:

   a) 6
   b) 3
   c) 4
   d) 5

2. .Are used for converting one type of number system in to other form

   a) encoder

   b) logic gate

c) half adder

d) full adder

3. Select the statement that best describes the parity method of error detection:

a) Parity checking is best suited for detecting double-bit errors that occur during the transmission of codes from one location to another

b) Parity checking is not suitable for detecting single-bit errors in transmitted codes.

c) Parity checking is best suited for detecting single-bit errors in transmitted codes.

d) Parity checking is capable of detecting and correcting errors in transmitted codes.

Sample Question:

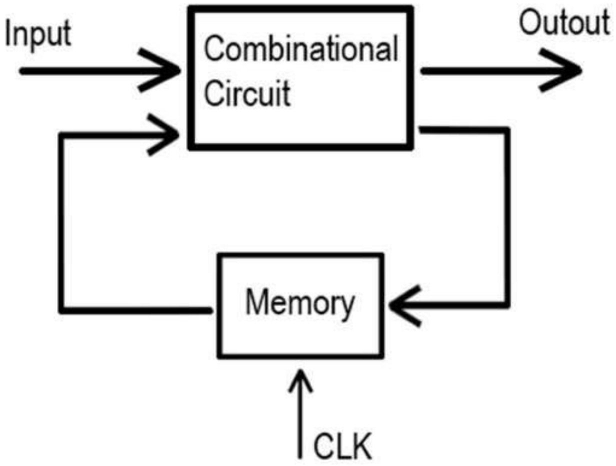
1. Implement full subtractor using MUX.
   Hint:[Boolean function design method using MUX]
2. Design a 4 bit even parity generator circuit with truth table and neat circuit diagram. Hint:[parity circuit design method]
3. Design the logic circuit of BCD to decimal encoder. Hint:[decoder design -4 to 10]
4. Implement full adder using decoder
   Hint:[full subtractor design method using decoder]

Assignment:

1. Design 16:1 MUX using 4:1 MUX. Write down the applications of MUX.
2. Design 2 bit magnitude comparator.
3. What are the differences between decoder and demultiplexer.

Module III

Sequential Circuit:

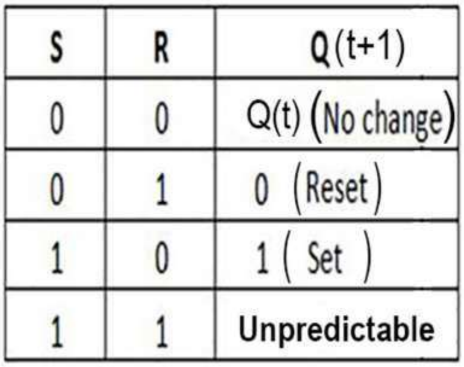


Unlike Combinational circuits that change state depending upon the actual signals being applied to their inputs at that time, Sequential Logic circuits have some form of inherent Memory built in. This means that sequential logic circuits are able to take into account their previous input state as well as those actually present, a sort of before and after effect is involved with sequential circuits.

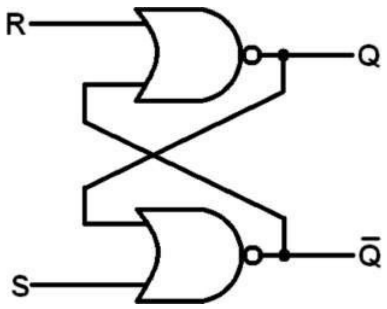
SR Latch

The simplest bistable device, therefore, is known as a set-reset, or S-R, latch.

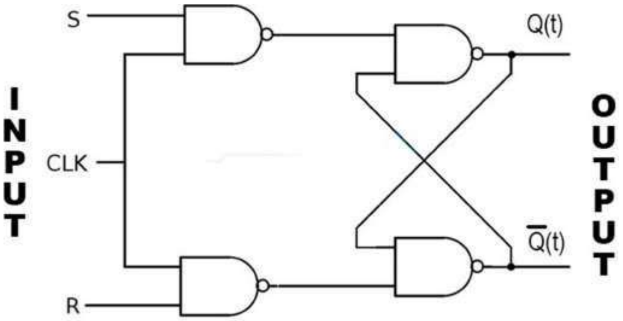
Truth Table



S	R	Q(t+1)
0	0	Q(t) (No change)
0	1	0 (Reset)
1	0	1 (Set)
1	1	Unpredictable



Circuit using NOR



Circuit using NAND

SR Flip Flop
The SR flip-flop, also known as a SR Latch, can be considered as one of the most basic sequential logic circuit possible. This simple flip-flop is basically a one-bit memory bistable device that has two inputs, one which will SET the device (meaning the output = 1), and is labelled S and another which will

RESET the device (meaning the output = 0) , labelled R.

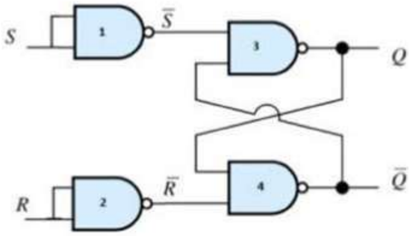
Then the SR description stands for Set-Reset . The reset input resets the flip-flop back to its original state with an output Q that will be either at a logic level 1 or logic 0 depending upon this set/reset condition.

The Set State
Consider the circuit shown above. If the input R is at logic level 0 (R = 0) and input S is at logic level 1 (S = 1), the NAND gate Y has at least one of its inputs at logic 0 therefore, its output Q must be at a logic level 1 (NAND Gate principles). Output Q is also fed back to input A and so both inputs to NAND gate X are at logic level 1 , and therefore its output Q must be at logic level 0 . Again NAND gate principals. If the reset input R changes state, and goes HIGH to logic
1 with S remaining HIGH also at logic level 1 , NAND gate Y inputs are now R = 1 and B = 0 . Since one of its inputs is still at logic level 0 the output at Q still remains HIGH at logic level 1 and there is no change of state. Therefore, the flip-flop circuit is said to be Latched or Set with Q = 1 and Q = 0 .

Reset State
In this second stable state, Q is at logic level 0 , (not Q = 0) its inverse output at Q is at logic level 1 , (Q = 1), and is given by R = 1 and S = 0 . As gate X has one of its inputs at logic 0 its output Q must equal logic level 1 (again NAND gate principles). Output Q is fed back to input B , so both inputs to NAND gate Y are at logic 1 , therefore, Q = 0 .
If the set input, S now changes state to logic 1 with input R remaining at logic 1 , output Q still remains LOW at logic level 0 and there is no change of state. Therefore, the flip-flop circuits Reset state has also been latched and we can define this set/reset action in the following truth table.

Unpredictable State

It can be seen that when both inputs S = 1 and R = 1 the outputs Q and Q can be at either logic level 1 or 0 , depending upon the state of the inputs S or R BEFORE this input condition existed. Therefore the condition of S = R = 1 does not change the state of the outputs Q and Q.
However, the input state of S = 0 and R = 0 is an undesirable or invalid condition and must be avoided.

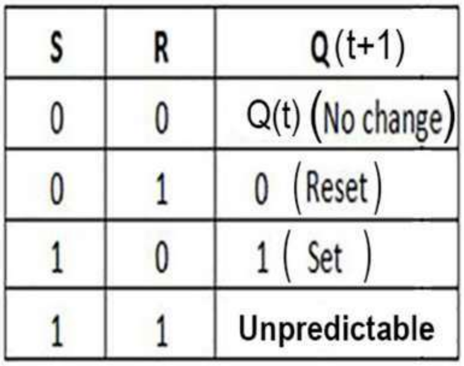


The Clock is a pulse rate

If CLK = 0 Q(t+1)

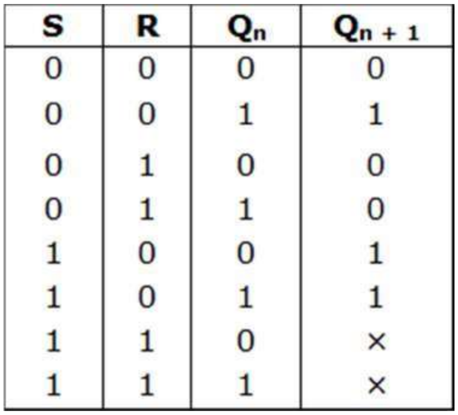
follows Q(t)

If CLK = 1



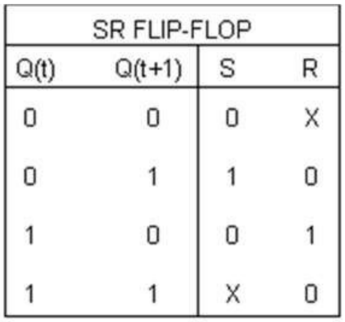
S	R	Q(t+1)
0	0	Q(t) (No change)
0	1	0 (Reset)
1	0	1 (Set)
1	1	Unpredictable

Characteristic Table



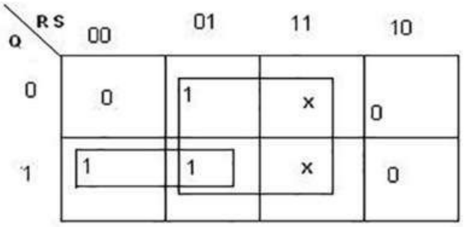
S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	×
1	1	1	×

Excitation Table



SR FLIP-FLOP			
Q(t)	Q(t+1)	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

EXCITATION TABLE OF SR FLIP-FLOP



Q \ RS	00	01	11	10
0	0	1	x	0
1	1	1	x	0

We place don't cares for R=1, S=1

The Equation we get is

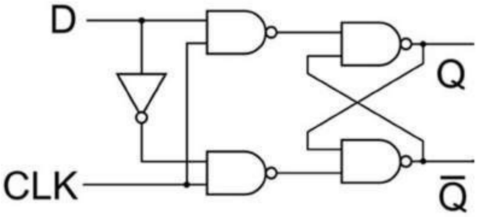
$$Q(t+1)=S+Q(t).\overline{R}$$

D Flip Flop

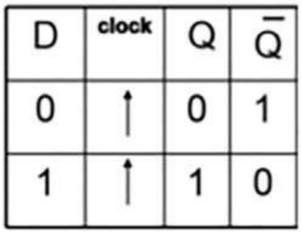
One of the main disadvantages of the basic SR NAND Gate bistable circuit is that the indeterminate input condition of SET = 0 and RESET = 0 is forbidden.

This state will force both outputs to be at logic 1 , over-riding the feedback latching action and whichever input goes to logic level 1 first will lose control, while the other input still at logic 0 controls the resulting state of the latch.

But in order to prevent this from happening an inverter can be connected between the SET and the RESET inputs to produce another type of flip flop circuit known as a Data Latch, Delay flip flop, Dtype Bistable, D-type Flip Flop or just simply a D Flip Flop as it is more generally called.



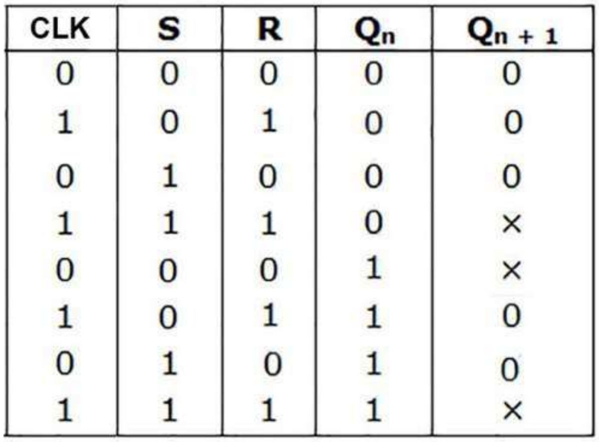
Truth Table



D	clock	Q	$\overline{Q}$
0	↑	0	1
1	↑	1	0

Use of Clock Pulse

Clock signal is generally the digital signal (consisting alternate HIGH and LOW signal). Primary objective to use them in digital circuits is to synchronize the action of one component in the circuit with other parts. The signal is used in different ways in different circuits.



CLK	S	R	Q_n	Q_{n+1}
0	0	0	0	0
1	0	1	0	0
0	1	0	0	0
1	1	1	0	×
0	0	0	1	×
1	0	1	1	0
0	1	0	1	0
1	1	1	1	×

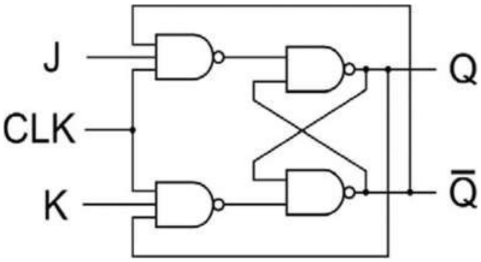
JK Flip Flop

The basic S-R NAND flip-flop circuit has many advantages and uses in sequential logic circuits but it suffers from two basic switching problems.

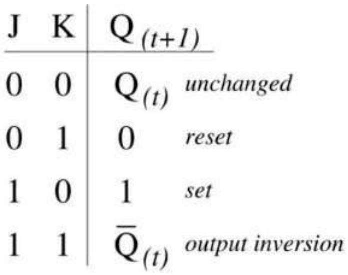
Number one, the S = 0 and R = 0 condition (S = R = 0) must always be avoided, and number two, if S or R change state while the enable input is high the correct latching action may not occur. Then to overcome these two fundamental design problems with the SR flip-flop design, the JK flip Flop was developed.

Both the S and the R inputs of the previous SR bistable have now been replaced by two inputs called the J and K inputs, respectively after its inventor Jack Kilby. Then this equates to: J = S and K = R. The two 2-input AND gates of the gated SR bistable have now been replaced by two 3-input NAND gates with the third input of each gate connected to the outputs at Q and Q. This cross coupling of the SR flip- flop allows the previously invalid condition of S = 1 and R = 1 state to be used to produce a toggle action as the two inputs are now interlocked.

If the circuit is now SET the J input is inhibited by the 0 status of Q through the lower NAND gate. If the circuit is RESET the K input is inhibited by the 0 status of Q through the upper NAND gate. As Q and Q are always different we can use them to control the input. When both inputs J and K are equal to logic 1 , the JK flip flop toggles as shown in the following truth table.

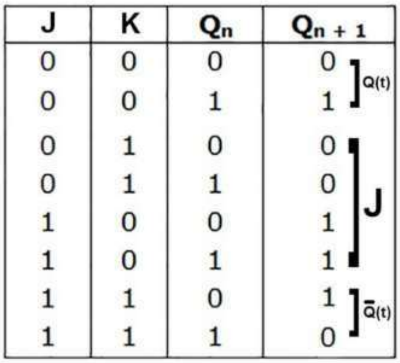


Truth Table:



J	K	$Q_{(t+1)}$	
0	0	$Q_{(t)}$	*unchanged*
0	1	0	*reset*
1	0	1	*set*
1	1	$\overline{Q}_{(t)}$	*output inversion*

Characteristic Table:



J	K	Q_n	Q_{n+1}	
0	0	0	0	$Q(t)$
0	0	1	1	
0	1	0	0	J
0	1	1	0	
1	0	0	1	
1	0	1	1	
1	1	0	1	$\overline{Q}(t)$
1	1	1	0	

Excitation Table:

J K FLIP-FLOP			
Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

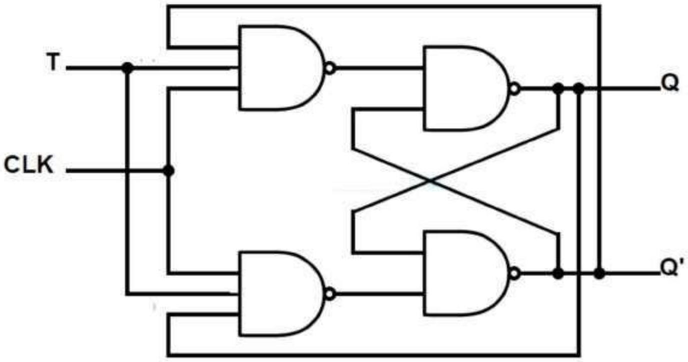
Q_p \ J K	00	01	11	10
0	0	0	1	1
1	1		0	1

The Equation we get is

$$Q(t) = J.\overline{Q}(t)+\overline{K}.Q(t)$$

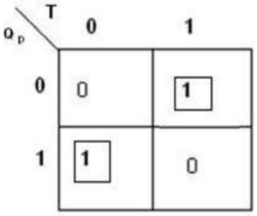
T Flip Flop

The T or "toggle" flip-flop changes its output on each clock edge, giving an output which is half the frequency of the signal to the T input.



Truth Table:

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



The Equation we get is

Q(t+1)=T.Q(t)+T.Q(t)

=T XOR Q(t)

<u>Race Around Condition</u>

When the input to the JK flip-flop is j=1 and k=1, the race around condition occurs, i.e it occurs when the time period of the clock pulse is greater than the propagation delay of the flip flop. so the output changes or toggles in a single clock period. If it toggles even number of times the output is same but if it toggles odd number of times then the output is complimented. To avoid race around condition we cant make the clock pulse smaller than the propagation delay so we use

Clock Pulse are 2 types

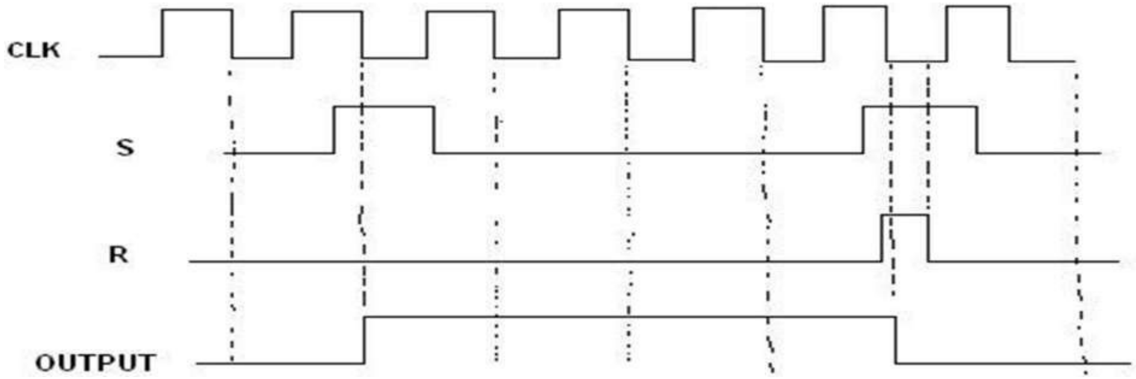
1. Level Trigger
2. Edge Trigger

To overcome Race Around Condition we use Master Slave Flipflop

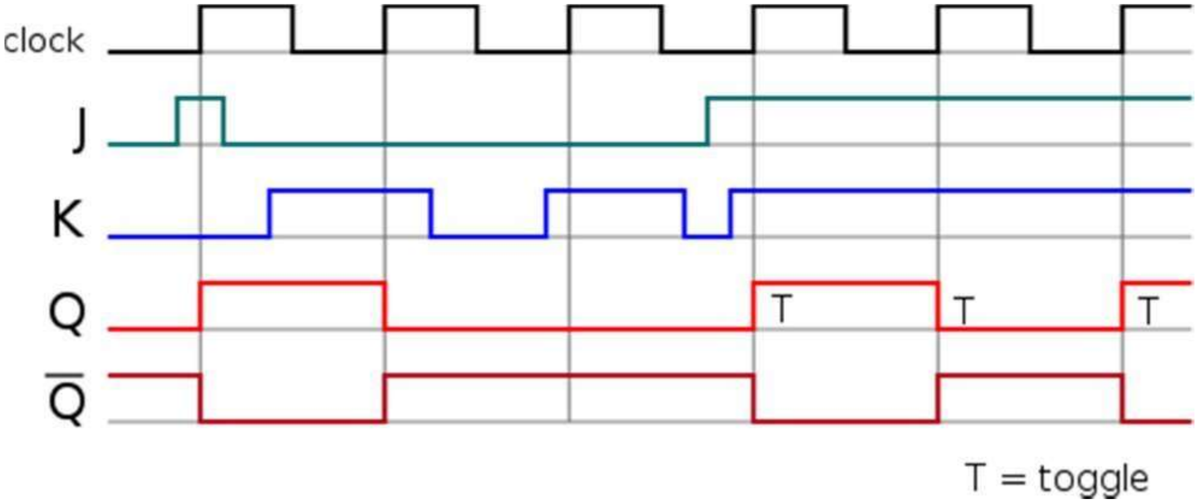
Master JK FF

Slave SR FF

Determine the output waveform Q(t+1) if the input shown in fig A are applied to a gate SR FF, the clock pulse shown in fig B and Q(t) is initially in Set condition.



Determine the output waveform Q(t+1) if the input shown in fig A are applied to a gate JK FF, the edge trigger clock pulse shown in fig B and Q(t) is initially in Set condition.

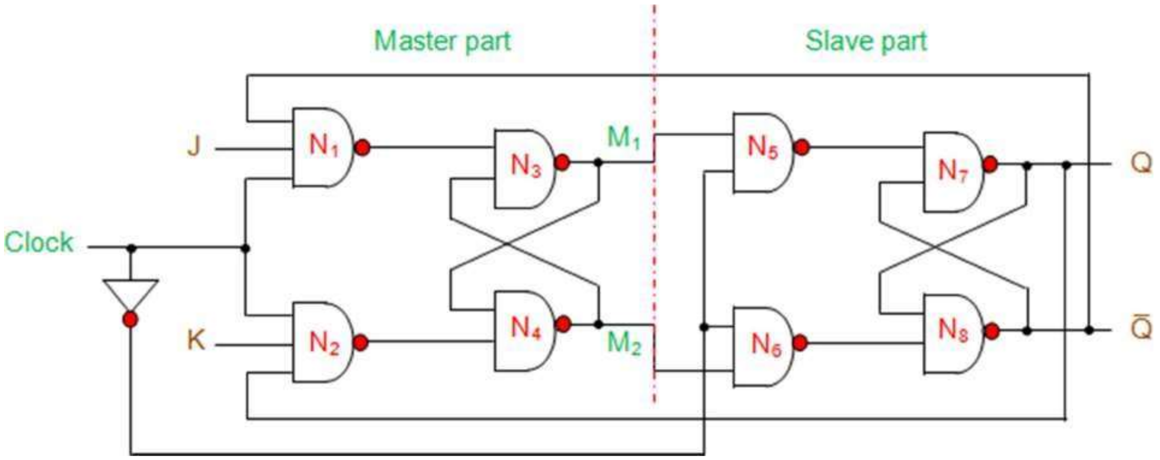


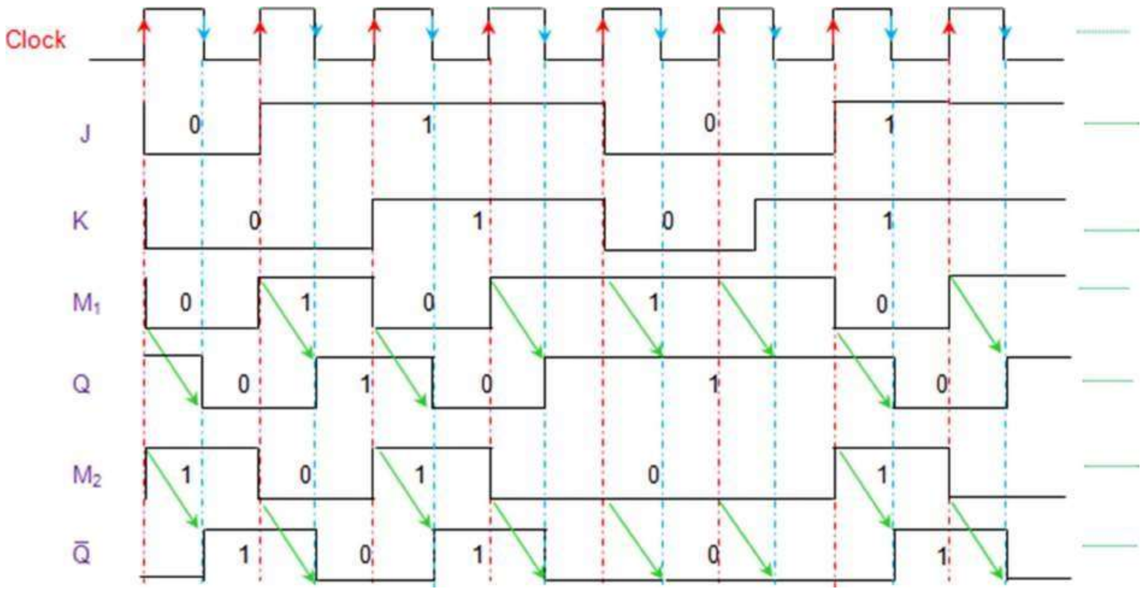
1. Master slave JK flip flop
2. Positive or negative edge triggering

Master Slave Flip Flop

Master Slave flip flop are the cascaded combination of two flip-flops among which the first is designated as master flip-flop while the next is called slave flip-flop . Here the master flip-flop is triggered by the external clock pulse train while the slave is activated at its inversion i.e. if the master is positive edge-triggered, then the slave is negative-edge triggered and vice-versa. This means that the data enters into the flip-flop at leading/trailing edge of the clock pulse while it is obtained at the output pins during trailing/leading edge of the clock pulse. Hence a master-slave flip-flop completes its operation only after the appearance of one full clock pulse for which they are also known as pulse- triggered flip-flops. The internal structure of a master-slave JK flip-flop in terms of NAND gates and an

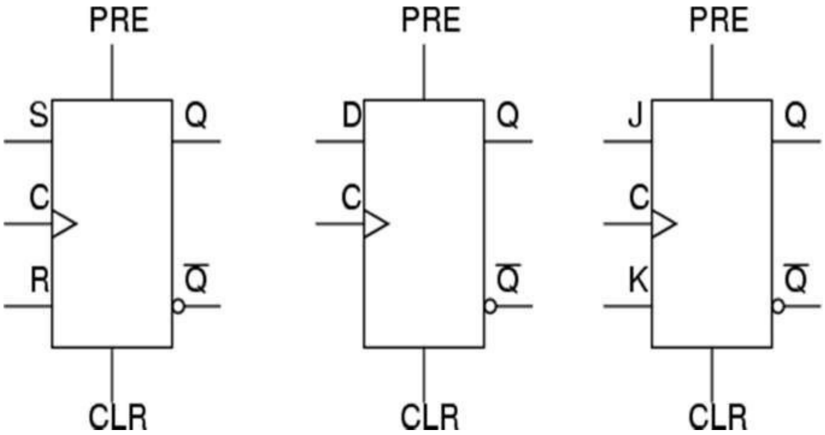
inverter (to complement the clock signal) is shown in Figure 2. Here it is seen that the NAND gate 1 (N_1) has three inputs viz ., external clock pulse (Clock), input J and output $\overline{Q}$; while the NAND gate 2 (N_2) has external clock pulse (Clock), input K and output Q as its inputs.

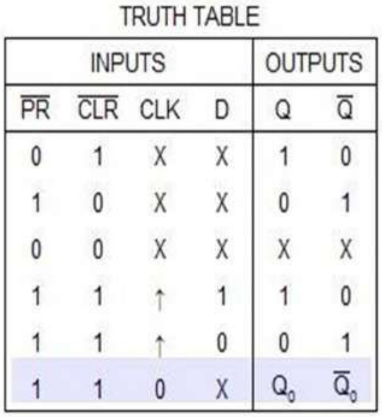




Asynchronous Flip-Flop Inputs

The normal data inputs to a flip flop (D, S and R, or J and K) are referred to as synchronous inputs because they have effect on the outputs (Q and not-Q) only in step, or in sync, with the clock signal transitions. These extra inputs that I now bring to your attention are called asynchronous because they can set or reset the flip-flop regardless of the status of the clock signal. Typically, they re called preset and clear:





TRUTH TABLE

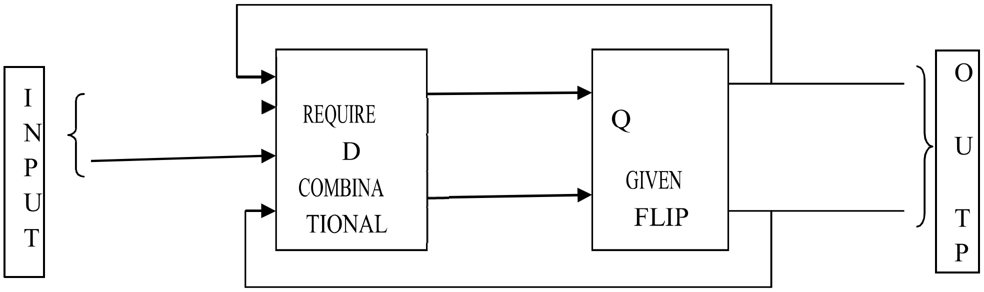
INPUTS				OUTPUTS	
$\overline{PR}$	$\overline{CLR}$	CLK	D	Q	$\overline{Q}$
0	1	X	X	1	0
1	0	X	X	0	1
0	0	X	X	X	X
1	1	↑	1	1	0
1	1	↑	0	0	1
1	1	0	X	Q_0	$\overline{Q}_0$

When the preset input is activated, the flip-flop will be set (Q=1, not-Q=0) regardless of any of the synchronous inputs or the clock. When the clear input is activated, the flip-flop will be reset (Q=0, not-Q=1), regardless of any of the synchronous inputs or the clock. So, what happens if both preset and clear inputs are activated? Surprise, surprise: we get an invalid state on the output, where Q and not- Q go to the same state, the same as our old friend, the S-R latch! Preset and clear inputs find use when multiple flip-flops are ganged together to perform a function on a multi-bit binary word, and a single line is needed to set or reset them all at once.

Asynchronous inputs, just like synchronous inputs, can be engineered to be active-high or activelow. If they re active-low, there will be an inverting bubble at that input lead on the block symbol, just like the negative edge-trigger clock inputs.Sometimes the designations PRE and CLR will be shown with inversion bars above them, to further denote the negative logic of these inputs:

Realization Of Flip-Flops Conversions:

For the conversion of one flip flop to another, a combinational circuit has to be designed first. If a Flip Flop is required, the inputs are given to the combinational circuit and the output of the combinational circuit is connected to the inputs of the actual flip flop. Thus, the output of the actual flip flop is the output of the required flip flop. In this post, the following flip flop conversions will be explained. The conversion from one type of flip-flop to another one needs a synchronous approach using the Excitation table and K-map simplification.



BASIC BLOCK DAIGRAM OF FLIP-FLOP CONVERSION

1) SR flip-flop to JK flip-flop:

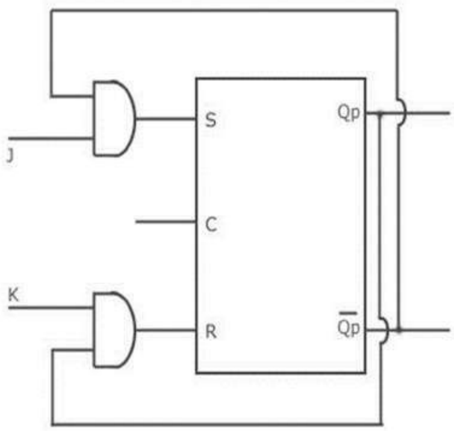
As discuss, J and K will be given as external inputs to S and R. As shown in the logic diagram below, S and R will be the outputs of the combinational circuit. The truth tables for the flip flop conversion are given below. The present state is represented by Qp and Qp+1 is the next state to be obtained when the J and K inputs are applied. For two inputs J and K, there will be eight possible combinations. For each combination of J, K and Qp, the corresponding Qp+1 states are found. Qp+1 simply suggests the future values to be obtained by the JK flip flop after the value of Qp. The table is then completed by writing the values of S and R required to get each Qp+1 from the corresponding Qp. That is, the values of S and R that are required to change the state of the flip flop from Qp to Qp+1 are written.

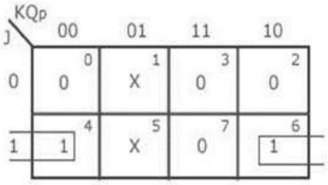
S-R Flip Flop to J-K Flip Flop

Conversion Table

J-K Inputs		Outputs		S-R Inputs	
J	K	Qp	Qp+1	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

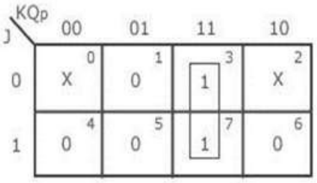
Logic Diagram





J \ KQp	00	01	11	10
0	0	X	0	0
1	1	X	0	1

$S = \overline{J}Q_p$



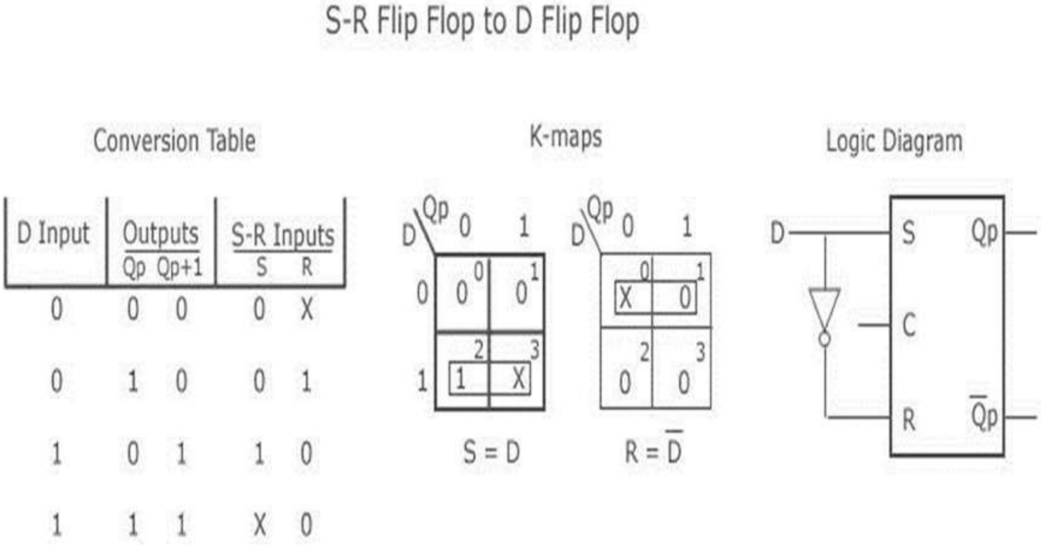
J \ KQp	00	01	11	10
0	X	0	1	X
1	0	0	1	0

$R = KQ_p$

K-Map

2) <u>SR Flip-flop to D Flip-flop:</u>

S and R are the actual inputs of the flip flop and D is the external input of the flip flop. The four combinations, the logic diagram, conversion table, and the K-map for S and R in terms of D and Qp are shown below.

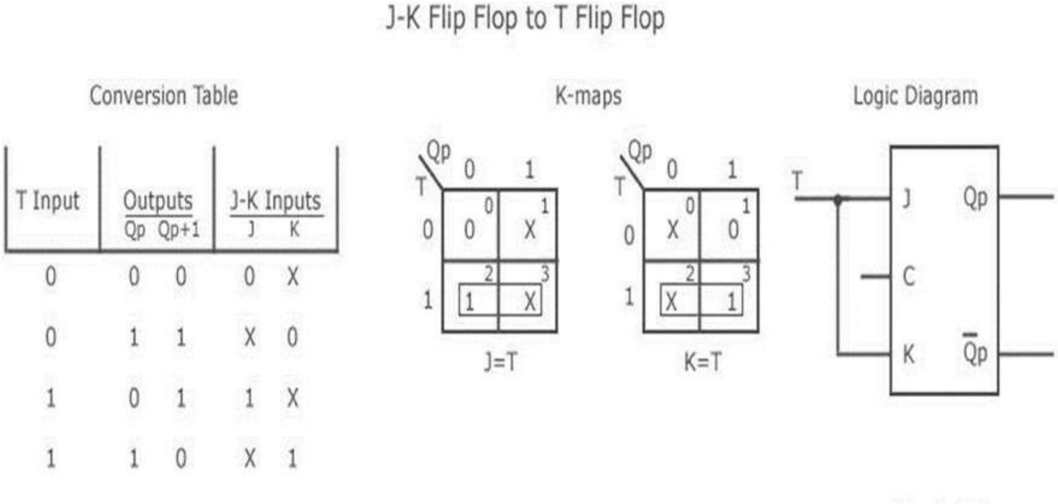

S-R Flip Flop to D Flip Flop

Conversion Table

D Input	Outputs		S-R Inputs	
	Qp	Qp+1	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

3) JK Flip-flop to T Flip-flop:

J and K are the actual inputs of the flip flop and T is taken as the external input for conversion. Four combinations are produced with T and Qp. J and K are expressed in terms of T and Qp. The conversion table, K-maps, and the logic diagram are given below.


J-K Flip Flop to T Flip Flop

Conversion Table

T Input	Outputs		J-K Inputs	
	Qp	Qp+1	J	K
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1

Multiple choice Type Questions:

1. A simple flip-flop
A. is 2 bit memory
B. is 1 bit memory
C. is a four state device
D. has nothing to do with memory

2. An SR flip flop cannot accept the following input entry
A. Both input zero
B. zero at R and one at S C. zero at S and one at R
D. Both inputs one

3. The main difference between JK and RS flip-flop is that?
A. JK flip-flop does not need a clock pulse
B. there is feedback in JK flip-flop
C. JK flip-flop accepts both inputs as 1
D. JK flip-flop is acronym of junction cathode multivibrator

4. How is a J-K flip-flop made to toggle?
A. J = 0, K = 0
B. J = 1, K = 0 C. J = 0, K = 1
D. J = 1, K = 1

5. Which of the following is correct for a gated D flip-flop? A. The output toggles if one of the inputs is held HIGH.
B. Only one of the inputs can be HIGH at a time.
C. The output complement follows the input when enabled. D. Q output follows the input D when the enable is HIGH.

6. With regard to a D latch, .
A. the Q output follows the D input when EN is LOW
B. the Q output is opposite the D input when EN is LOW
C. the Q output follows the D input when EN is HIGH
D. the Q output is HIGH regardless of EN's input state

7. A correct output is achieved from a master-slave J-K flip-flop only if its inputs are stable while the:
A. clock is LOW
B. slave is transferring
C. flip-flop is reset

8. On a master-slave flip-flop, when is the master enabled?
A. when the gate is LOW
B. when the gate is HIGH

C. both of the above
D. neither of the above

9. Popular application flip-flop are ?
A. Counters
B. Shift registers
C. Transfer registers
D. All of above

10. SR Flip flop can be converted to T-type flip-flop if ?
A. S is connected to Q
B. R is connected to Q
C. Both S and R are shortend
D. S and R are connected to Q and Q' respectively

Assignments:

1. Implement a clocked JK flip-flop using NAND gates.
2. What do you mean by Race-around condition of a flip-flop? How can it be overcome?
3. Explain the Master-Slave flip-flop.
4. How truth table of J/K flip-flop satisfied with a D flip-flop. Design the circuit for it.
5. What is Edge trigger and Level trigger clock pluse?

Model Questions:

1. (a) Perform the conversion from D flip-flop to S-R flip-flop.
(b) What is the difference between combinational and sequential circuits? What is flip-flop?
(c) Draw the logic diagram of the master- slave JK flip-flop. Why is it called so? Explain it.

2. (a)What do you mean by Asynchronous inputs of a flip-flop? What is edge trigger flip-flop and why is it required? Convert S-R flip-flop to J-K flip-flop.
(b) Draw the state table of a JK flip-flop and write down its characteristic equation.
(c)What are the main differences between a latch and a flip-flop? (d) Perform the conversion from D flip-flop to JK flip-flop.

Register:

Shift registers, like counters, are a form of sequential logic. Sequential logic, unlike combinational logic is not only affected by the present inputs, but also, by the prior history. In other words, sequential logic remembers past events.

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

Formerly, very long (several hundred stages) shift registers served as digital memory. This obsolete application is reminiscent of the acoustic mercury delay lines used as early computer memory.

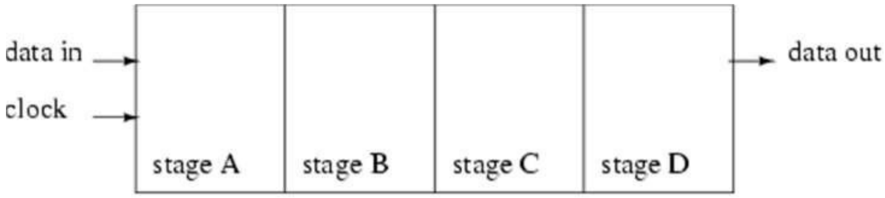
Serial data transmission, over a distance of meters to kilometers, uses shift registers to convert parallel data to serial form. Serial data communications replaces many slow parallel data wires with a single serial high speed circuit.

Serial data over shorter distances of tens of centimeters, uses shift registers to get data into and out of microprocessors. Numerous peripherals, including analog to digital converters, digital to analog converters, display drivers, and memory, use shift registers to reduce the amount of wiring in circuit boards.

Some specialized counter circuits actually use shift registers to generate repeating waveforms. Longer shift registers, with the help of feedback generate patterns so long that they look like random noise, pseudo-noise.

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

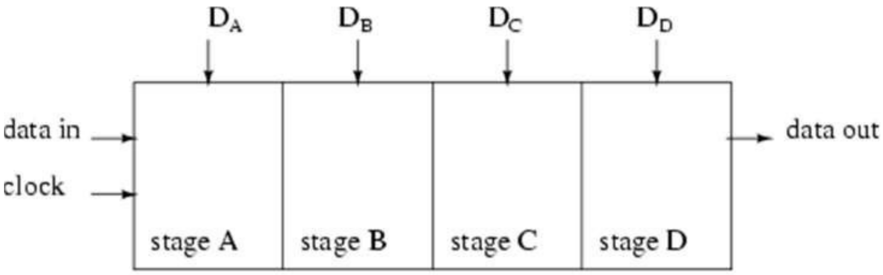
- Serial-in/serial-out
- Parallel-in/serial-out
- Serial-in/parallel-out
- Universal parallel-in/parallel-out
- Ring counter



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

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

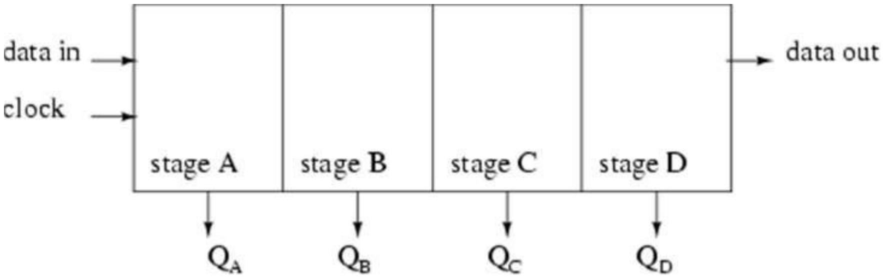
Data at data in , above, will be present at the Stage A output after the first clock pulse. After the second pulse stage A data is transfered to stage B output, and data in is transfered to stage A output. After the third clock, stage C is replaced by stage B; stage B is replaced by stage A; and stage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



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

Data is loaded into all stages at once of a parallel-in/serial-out shift register. The data is then shifted out via data out by clock pulses. Since a 4- stage shift register is shown above, four clock pulses are required to shift out all of the data. In the diagram above, stage D data will be present at the data out up until the first clock pulse; stage C data will be present at data out between the first clock and the second clock pulse; stage Bdata will be present between the second clock and the third clock; and stage A data will be present between the third and the fourth clock. After the fourth clock pulse and thereafter, successive bits of data in should appear at data out of the shift register after a delay ofour clock pulses.

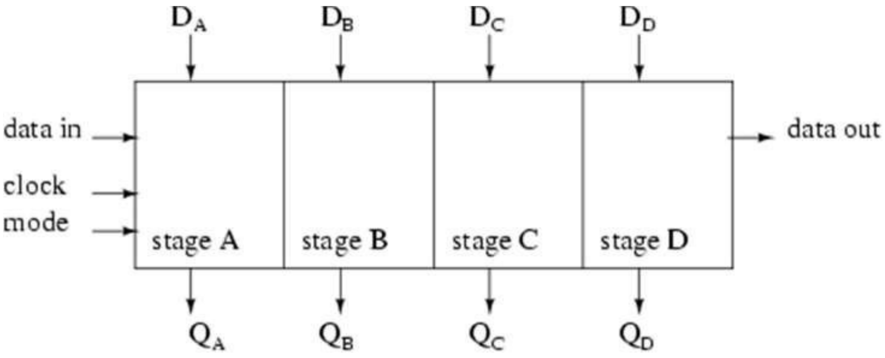
If four switches were connected to D_A through D_D, the status could be read into a microprocessor using only one data pin and a clock pin. Since adding more switches would require no additional pins, this approach looks attractive for many inputs.



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

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

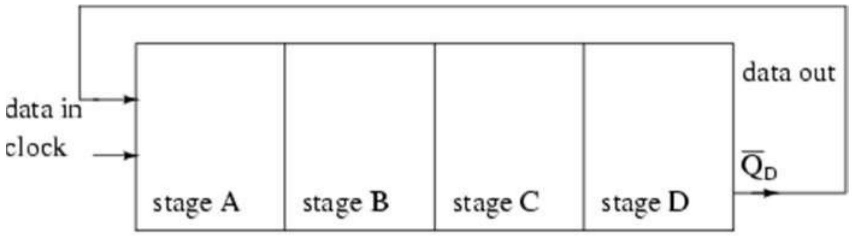
After the first clock, the data at data in appears at Q_A. After the second clock, The old Q_A data appears at Q_B; Q_A receives next data from data in . After the third clock, Q_B data is at Q_C. After the fourth clock, Q_C data is at Q_D. This stage contains the data first present at data in . The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



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

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

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



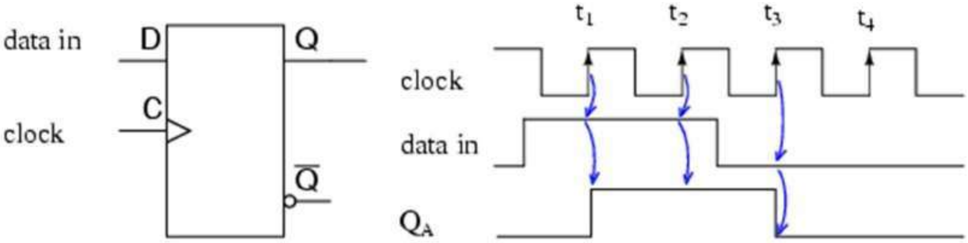
Ring Counter, shift register output fed back to input

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

Circuit diagram of serial in serial out shift register:

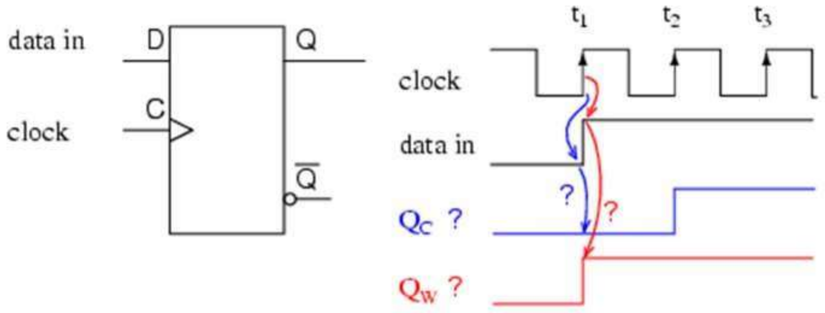
Serial-in, serial-out shift registers delay data by one clock time for each stage. They will store a bit of data for each register. A serial-in, serial-out shift register may be one to 64 bits in length, longer if registers or packages are cascaded.

Below is a single stage shift register receiving data which is not synchronized to the register clock. The data in at the D pin of the type D FF (Flip-Flop) does not change levels when the clock changes for low to high. We may want to synchronize the data to a system wide clock in a circuit board to improve the reliability of a digital logic circuit.



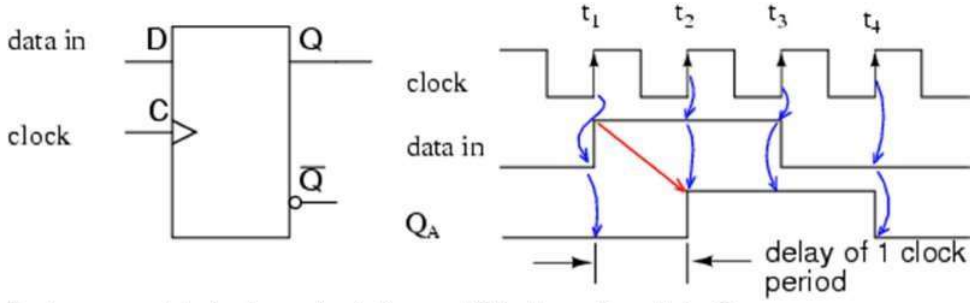
Data present at clock time is transfered from D to Q.

The obvious point (as compared to the figure below) illustrated above is that whatever data in is present at theD pin of a type D FF is transfered from D to output Q at clock time. Since our example shift register uses positive edge sensitive storage elements, the output Q follows the D input when the clock transitions from low to high as shown by the up arrows on the diagram above. There is no doubt what logic level is present at clock time because the data is stable well before and after the clock edge. This is seldom the case in multi-stage shift registers. But, this was an easy example to start with. We are only concerned with the positive, low to high, clock edge. The falling edge can be ignored. It is very easy to see Q follow D at clock time above. Compare this to the diagram below where the data in appears to change with the positive clock edge.



Does the clock t_1 see a 0 or a 1 at data in at D?   Which output is correct, Q_C or Q_W?

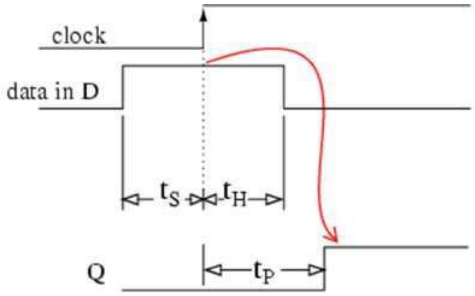
Since data in appears to changes at clock time t_1 above, what does the type D FF see at clock time? The short over simplified answer is that it sees the data that was present at D prior to the clock. That is what is transfered to Q at clock time t_1. The correct waveform is Q_C. At t_1 Q goes to a zero if it is not already zero. TheD register does not see a one until time t_2, at which time Q goes high.



Data present t_H before clock time at D is transfered to Q.

Since data, above, present at D is clocked to Q at clock time, and Q cannot change until the next clock time, theD FF delays data by one clock period, provided that the data is already synchronized to the clock. The $Q_A$waveform is the same as data in with a one clock period delay.

A more detailed look at what the input of the type D Flip-Flop sees at clock time follows. Refer to the figure below. Since data in appears to changes at clock time (above), we need further information to determine what the D FF sees. If the data in is from another shift register stage, another same type D FF, we can draw some conclusions based on data sheet information. Manufacturers of digital logic make available information about their parts in data sheets, formerly only available in a collection called a data book. Data books are still available; though, the manufacturer s web site is the modern source.



Data must be present (t_S) before the clock and after(t_H) the clock. Data is delayed from D to Q by propagation delay (t_P)

The following data was extracted from the CD4006b data sheet for operation at $5V_{DC}$, which serves as an example to illustrate timing.

[*]

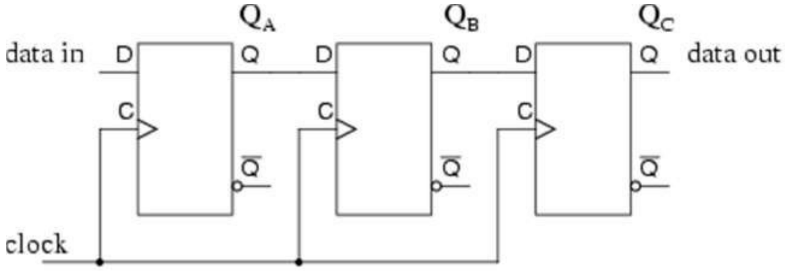
- t_S=100ns
- t_H=60ns
- t_P=200-400ns typ/max

t_S is the setup time, the time data must be present before clock time. In this case data must be present at D100ns prior to the clock. Furthermore, the data must be held for hold time t_H=60ns after clock time. These two conditions must be met to reliably clock data from D to Q of the Flip-Flop.

There is no problem meeting the setup time of 60ns as the data at D has been there for the whole previous clock period if it comes from another shift register stage. For example, at a clock frequency of 1 Mhz, the clock period is 1000 µs, plenty of time. Data will actually be present for 1000µs prior to the clock, which is much greater than the minimum required t_S of 60ns.

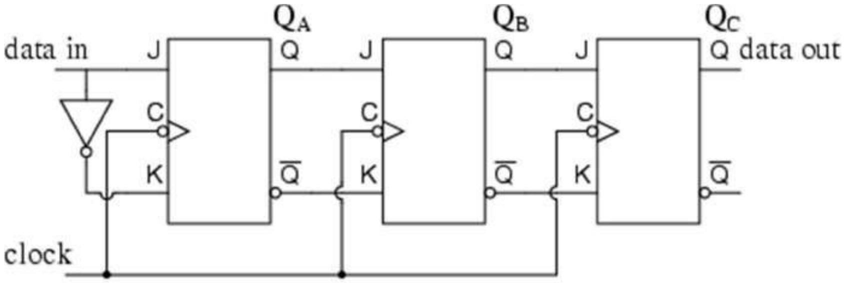
The hold time t_H=60ns is met because D connected to Q of another stage cannot change any faster than the propagation delay of the previous stage t_P=200ns. Hold time is met as long as the propagation delay of the previous D FF is greater than the hold time. Data at D driven by another stage Q will not change any faster than 200ns for the CD4006b.

To summarize, output Q follows input D at nearly clock time if Flip-Flops are cascaded into a multi- stage shift register.



Serial-in, serial-out shift register using type "D" storage elements

Three type D Flip-Flops are cascaded Q to D and the clocks paralleled to form a three stage shift register above.

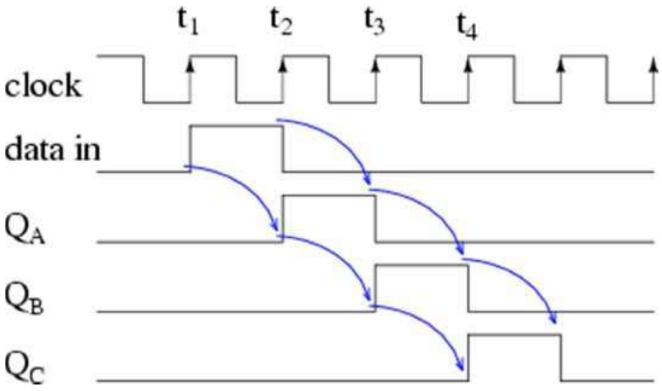


Serial-in, serial-out shift register using type "JK" storage elements

Type JK FFs cascaded Q to J, Q to K with clocks in parallel to yield an alternate form of the shift register above.

A serial-in/serial-out shift register has a clock input, a data input, and a data output from the last stage. In general, the other stage outputs are not available Otherwise, it would be a serial-in, parallel- out shift register..

The waveforms below are applicable to either one of the preceding two versions of the serialin, serial-out shift register. The three pairs of arrows show that a three stage shift register temporarily stores 3-bits of data and delays it by three clock periods from input to output.



At clock time t_1 a data in of 0 is clocked from D to Q of all three stages. In particular, D of stage A sees a logic0, which is clocked to Q_A where it remains until time t_2.

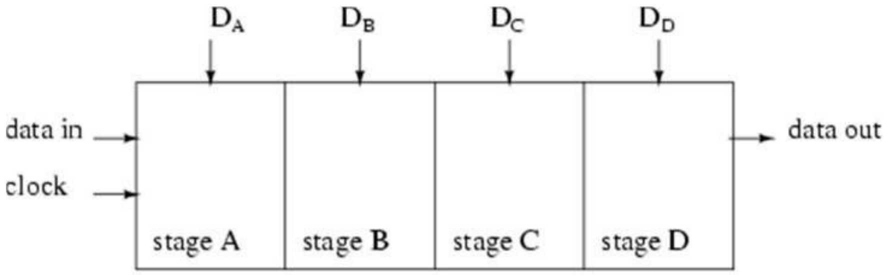
At clock time t_2 a data in of 1 is clocked from D to Q_A. At stages B and C, a 0, fed from preceding stages is clocked to Q_B and Q_C.

At clock time t_3 a data in of 0 is clocked from D to Q_A. Q_A goes low and stays low for the remaining clocks due to data in being 0. Q_B goes high at t_3 due to a 1 from the previous stage. Q_C is still low after t_3 due to a low from the previous stage.

Q_C finally goes high at clock t_4 due to the high fed to D from the previous stage Q_B. All earlier stages have 0s shifted into them. And, after the next clock pulse at t_5, all logic 1s will have been shifted out, replaced by 0s

Circuit diagram of parallel in serial out shift register:

Parallel-in/ serial-out shift registers do everything that the previous serial-in/ serial-out shift registers do plus input data to all stages simultaneously. The parallel-in/ serial-out shift register stores data, shifts it on a clock by clock basis, and delays it by the number of stages times the clock period. In addition, parallel-in/ serial-out really means that we can load data in parallel into all stages before any shifting ever begins. This is a way to convert data from a parallel format to a serial format. By parallel format we mean that the data bits are present simultaneously on individual wires, one for each data bit as shown below. By serial format we mean that the data bits are presented sequentially in time on a single wire or circuit as in the

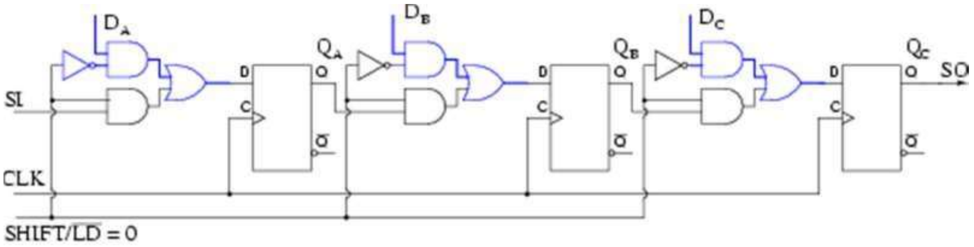


Parallel-in, serial-out shift register with 4-stages

case of the data out on the block diagram below.

Below we take a close look at the internal details of a 3-stage parallel-in/ serial-out shift register. A stage consists of a type D Flip-Flop for storage, and an AND-OR selector to determine whether data will load in parallel, or shift stored data to the right. In general, these elements will be replicated for the number of stages required. We show three stages due to space limitations.

Four, eight or sixteen bits is normal for real parts.

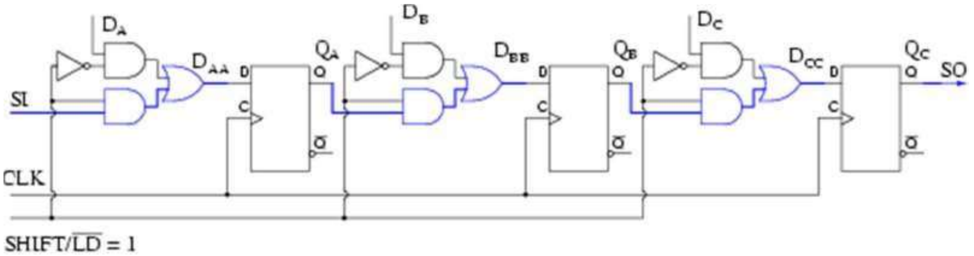


Parallel-in/ serial-out shift register showing parallel load path

Above we show the parallel load path when SHIFT/LD is logic low. The upper NAND gates serving D_A D_B $D_C$are enabled, passing data to the D inputs of type D Flip-Flops Q_A Q_B D_C respectively. At the next positive going clock edge, the data will be clocked from D to Q of the three FFs.

Three bits of data will load into Q_A Q_B D_C at the same time.

The type of parallel load just described, where the data loads on a clock pulse is known as synchronous load because the loading of data is synchronized to the clock. This needs to be differentiated from asynchronous load where loading is controlled by the preset and clear pins of the Flip-Flops which does not require the clock. Only one of these load methods is used within an individual device, the synchronous load being more common

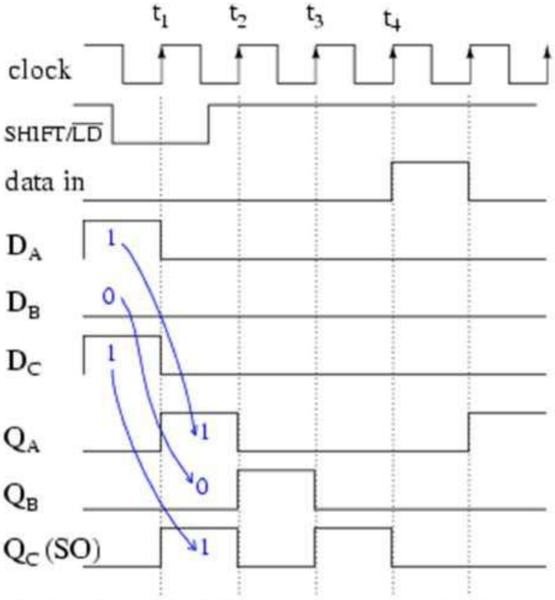


Parallel-in/ serial-out shift register showing shift path

in newer devices.

The shift path is shown above when SHIFT/LD is logic high. The lower AND gates of the pairs feeding the OR gate are enabled giving us a shift register connection of SI to D_A , Q_A to D_B , Q_B to D_C , Q_C to SO. Clock pulses will cause data to be right shifted out to SO on successive pulses.

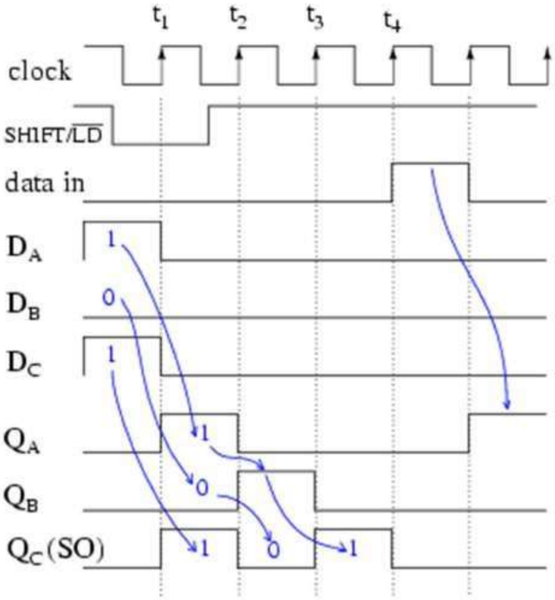
The waveforms below show both parallel loading of three bits of data and serial shifting of this data. Parallel data at D_A D_B D_C is converted to serial data at SO.



Parallel-in/ serial-out shift register load/shift waveforms

What we previously described with words for parallel loading and shifting is now set down as waveforms above. As an example we present 101 to the parallel inputs D_{AA} D_{BB} D_{CC}. Next, the SHIFT/LD goes low enabling loading of data as opposed to shifting of data. It needs to be low a short time before and after the clock pulse due to setup and hold requirements. It is considerably wider than it has to be. Though, with synchronous logic it is convenient to make it wide. We could have made the active low SHIFT/LD almost two clocks wide, low almost a clock before t_1 and back high just before t_3. The important factor is that it needs to be low around clock time t_1 to enable parallel loading of the data by the clock.

Note that at t_1 the data 101 at D_A D_B D_C is clocked from D to Q of the Flip-Flops as shown at Q_A Q_B Q_C at time t_1. This is the parallel loading of the data synchronous with the clock.



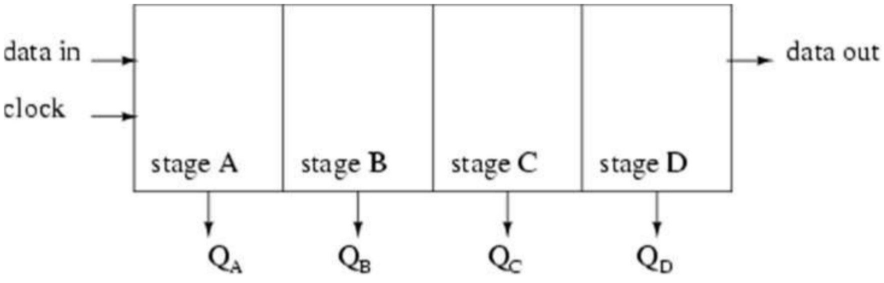
Parallel-in/ serial-out shift register load/shift waveforms

Now that the data is loaded, we may shift it provided that SHIFT/LD is high to enable shifting, which it is prior to t_2. At t_2 the data 0 at Q_C is shifted out of SO which is the same as the Q_C waveform. It is either shifted into another integrated circuit, or lost if there is nothing connected to SO. The data at Q_B, a 0 is shifted to Q_C. The 1at Q_A is shifted into Q_B. With data in a 0, Q_A becomes 0. After t_2, Q_A Q_B Q_C = 010.

After t_3, Q_A Q_B Q_C = 001. This 1, which was originally present at Q_A after t_1, is now present at SO and Q_C. The last data bit is shifted out to an external integrated circuit if it exists. After t_4 all data from the parallel load is gone. At clock t_5 we show the shifting in of a data 1 present on the SI, serial input.

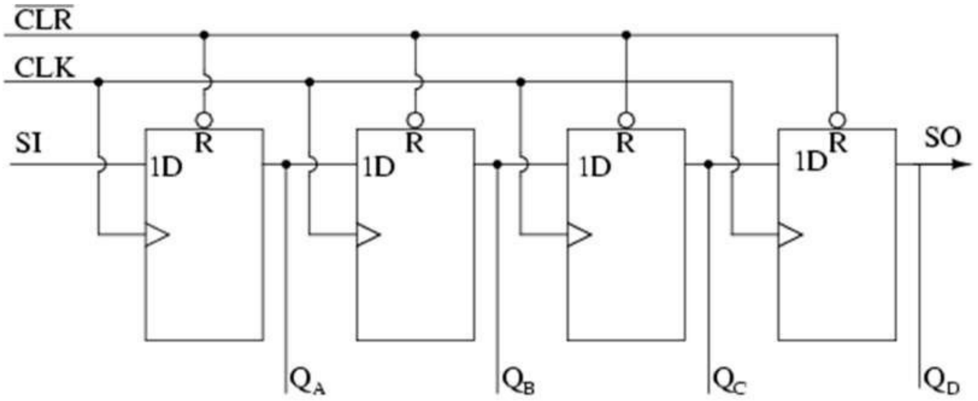
Circuit diagram of serial in parallel out shift register:

A serial-in/parallel-out shift register is similar to the serial-in/ serial-out shift register in that it shifts data into internal storage elements and shifts data out at the serial-out, data-out, pin. It is different in that it makes all the internal stages available as outputs. Therefore, a serialin/parallel-out shift register converts data from serial format to parallel format. If four data bits are shifted in by four clock pulses via a single wire at data-in, below, the data becomes available simultaneously on the four Outputs Q_A to Q_D after the fourth clock pulse.



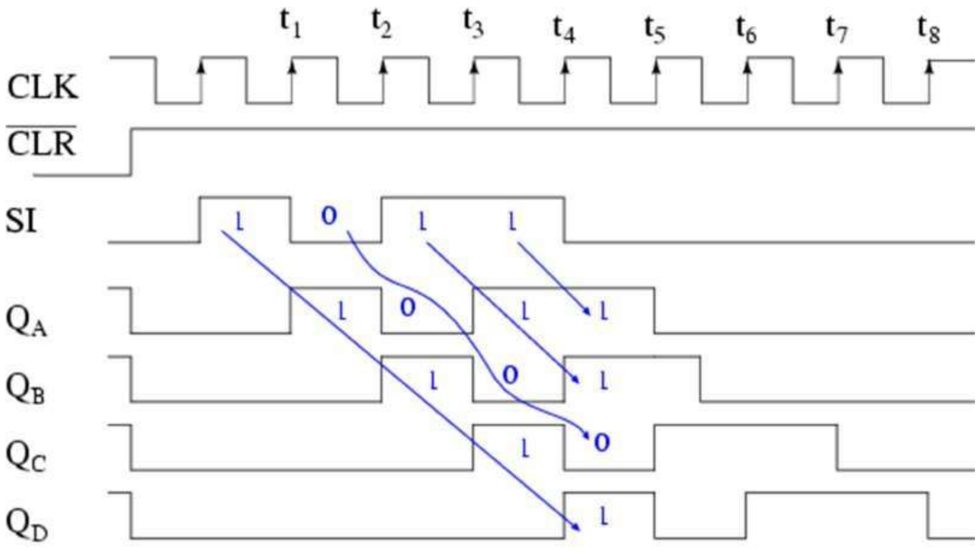
Serial-in, parallel-out shift register with 4-stages

The practical application of the serial-in/parallel-out shift register is to convert data from serial format on a single wire to parallel format on multiple wires. Perhaps, we will illuminate four LEDs (Light Emitting Diodes) with the four outputs (Q_A Q_B Q_C Q_D).



Serial-in/ Parallel out shift register details

The above details of the serial-in/parallel-out shift register are fairly simple. It looks like a serialin/ serial-out shift register with taps added to each stage output. Serial data shifts in at SI (Serial Input). After a number of clocks equal to the number of stages, the first data bit in appears at SO (Q_D) in the above figure. In general, there is no SO pin. The last stage (Q_D above) serves as SO and is cascaded to the next package if it exists. See waveforms below for above shift register.



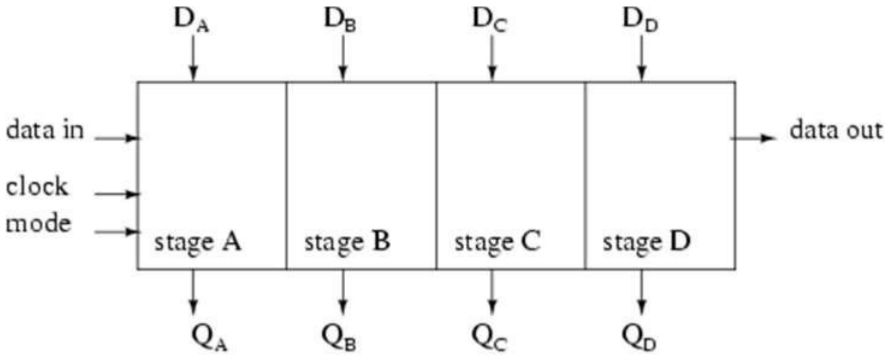
Serial-in/ parallel-out shift register waveforms

The shift register has been cleared prior to any data by CLR , an active low signal, which clears all type D Flip-Flops within the shift register. Note the serial data 1011 pattern presented at the SI input. This data is synchronized with the clock CLK. This would be the case if it is being shifted in from something like another shift register, for example, a parallel-in/ serial-out shift register (not shown here). On the first clock at t1, the data1 at SI is shifted from D to Q of the first shift register stage. After t2 this first data bit is at Q_B. After t3 it is at Q_C. After t4 it is at Q_D. Four clock pulses have shifted the first data bit all the way to the last stage Q_D. The second data bit a 0 is at Q_C after the 4th clock. The third data bit a 1 is at Q_B. The fourth data bit another 1 is at Q_A. Thus, the serial data input pattern 1011 is contained in (Q_D Q_C Q_B Q_A). It is now available on the four outputs.

It will available on the four outputs from just after clock t_4 to just before t_5. This parallel data must be used or stored between these two times, or it will be lost due to shifting out the Q_D stage on following clocks t_5 to t_8 as shown above.

Circuit diagram of parallel in parallel out shift register:

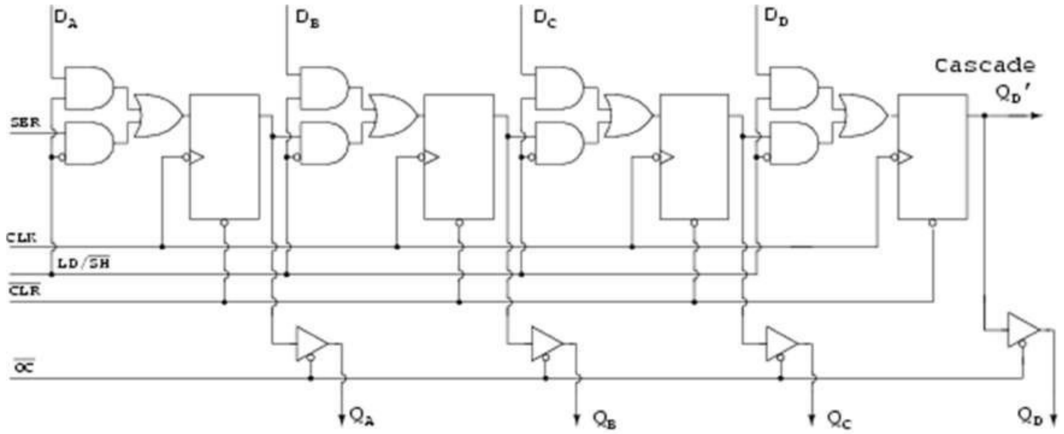
The purpose of the parallel-in/ parallel-out shift register is to take in parallel data, shift it, then output it as shown below. A universal shift register is a do-everything device in addition to the parallel-in/ parallel-out function.



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

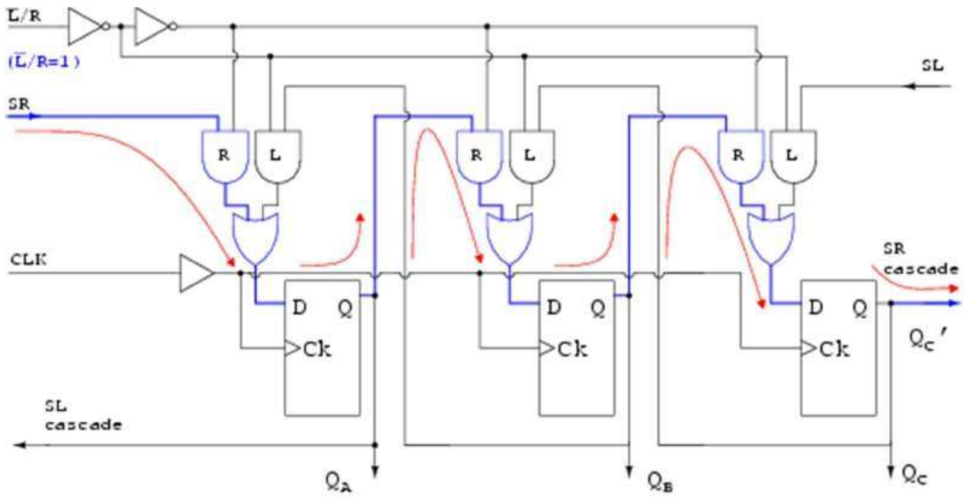
Above we apply four bit of data to a parallel-in/ parallel-out shift register at D_A D_B D_C D_D. The mode control, which may be multiple inputs, controls parallel loading vs shifting. The mode control may also control the direction of shifting in some real devices. The data will be shifted one bit position for each clock pulse. The shifted data is available at the outputs Q_A Q_B Q_C Q_D . The data in and data out are provided for cascading of multiple stages. Though, above, we can only cascade data for right shifting. We could accommodate cascading of left-shift data by adding a pair of left pointing signals, data in and data out , above.

The internal details of a right shifting parallel-in/ parallel-out shift register are shown below. The tri- state buffers are not strictly necessary to the parallel-in/ parallel-out shift register, but are part of the real-world device shown below.



74LS395 parallel-in/ parallel-out shift register with tri-state output

Universal shift register:

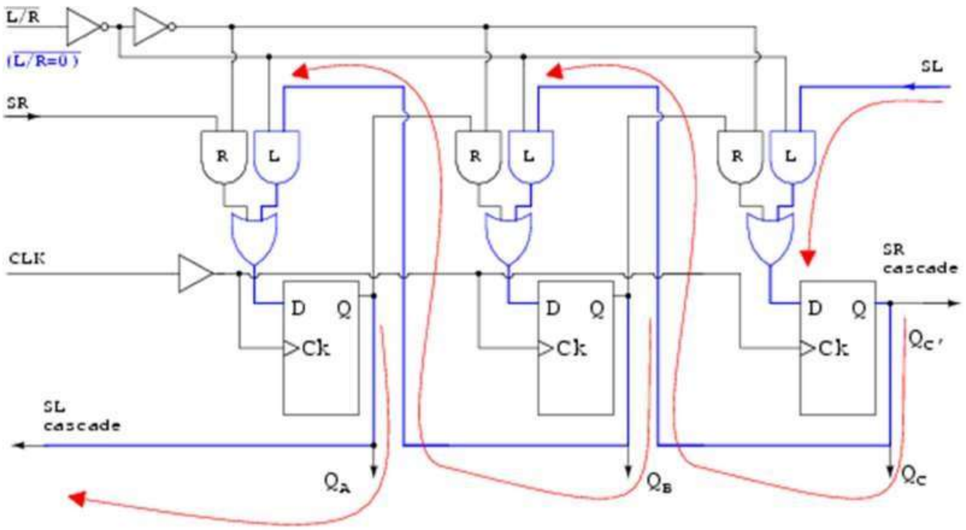


Shift left/ right, right action

What we have above is a hypothetical shift register capable of shifting either direction under the control of L /R. It is setup with L /R=1 to shift the normal direction, right. L /R=1 enables the multiplexer AND gates labeled R. This allows data to follow the path illustrated by the arrows, when a clock is applied. The connection path is the same as the"too simple shift right figure above.

Data shifts in at SR, to Q_A, to Q_B, to Q_C, where it leaves at SR cascade. This pin could drive SR of another device to the right.

What if we change L /R to L /R=0?

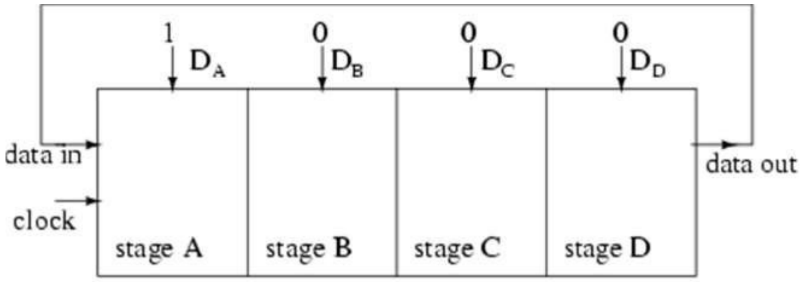


Shift left/ right register, left action

With L /R=0, the multiplexer AND gates labeled L are enabled, yielding a path, shown by the arrows, the same as the above shift left figure. Data shifts in at SL, to Q_C, to Q_B, to Q_A, where it leaves at SL cascade. This pin could drive SL of another device to the left.

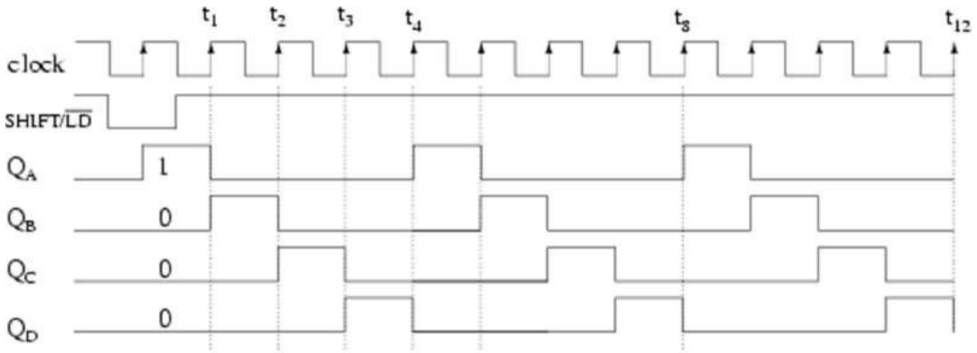
Ring Counter:

If the output of a shift register is fed back to the input. a ring counter results. The data pattern contained within the shift register will recirculate as long as clock pulses are applied. We make provisions for loading data into the parallel-in/ serial-out shift register configured as a ring counter below. Any random pattern may be loaded. The most generally useful pattern is a single 1.



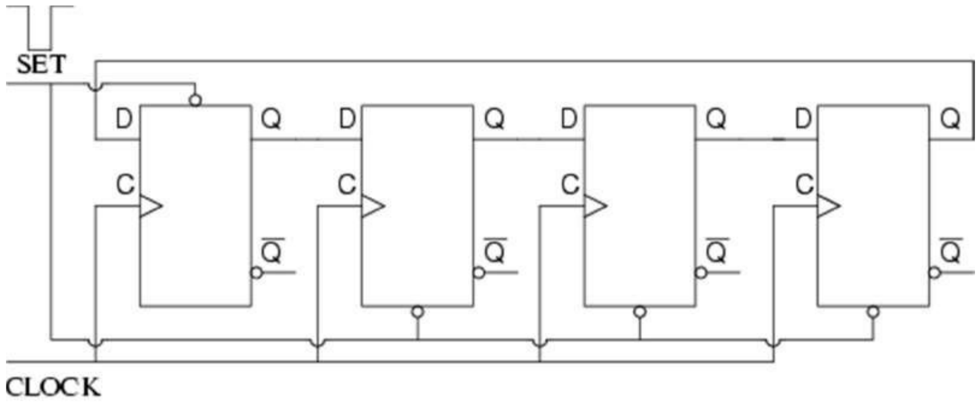
Parallel-in, serial-out shift register configured as a ring counter

Loading binary 1000 into the ring counter, above, prior to shifting yields a viewable pattern. The data pattern for a single stage repeats every four clock pulses in our 4-stage example. The waveforms for all four stages look the same, except for the one clock time delay from one stage to the next. See figure below.



Load 1000 into 4-stage ring counter and shift

The circuit above is a divide by 4 counter. Comparing the clock input to any one of the outputs, shows a frequency ratio of 4:1. How may stages would we need for a divide by 10 ring counter? Ten stages would recirculate the 1 every 10 clock pulses.

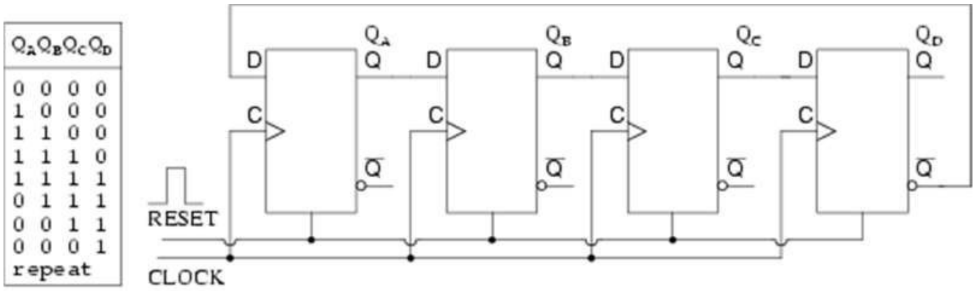


Set one stage, clear three stages

An alternate method of initializing the ring counter to 1000 is shown above. The shift waveforms are identical to those above, repeating every fourth clock pulse. The requirement for initialization is a disadvantage of the ring counter over a conventional counter. At a minimum, it must be initialized at power-up since there is no way to predict what state flip-flops will power up in.

Johnson Counter:

The switch-tail ring counter, also know as the Johnson counter, overcomes some of the limitations of the ring counter. Like a ring counter a Johnson counter is a shift register fed back on its self. It requires half the stages of a comparable ring counter for a given division ratio. If the complement output of a ring counter is fed back to the input instead of the true output, a Johnson counter results. The difference between a ring counter and a Johnson counter is which output of the last stage is fed back (Q or Q). Carefully compare the feedback connection below to the previous ring counter.

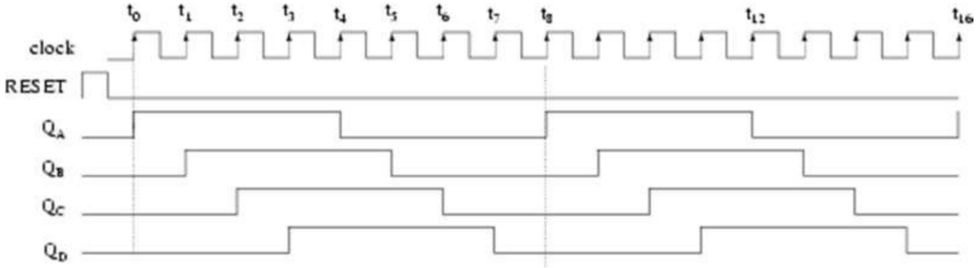


Q_A	Q_B	Q_C	Q_D
0	0	0	0
1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1
0	1	1	1
0	0	1	1
0	0	0	1
repeat			

Johnson counter (note the $\overline{Q}_D$ to D_A feedback connection)

This reversed feedback connection has a profound effect upon the behavior of the otherwise similar circuits. Recirculating a single 1 around a ring counter divides the input clock by a factor equal to the number of stages. Whereas, a Johnson counter divides by a factor equal to twice the number of stages. For example, a 4-stage ring counter divides by 4. A 4-stage Johnson counter divides by 8.

Start a Johnson counter by clearing all stages to 0s before the first clock. This is often done at power- up time. Referring to the figure below, the first clock shifts three 0s from (Q_A Q_B Q_C) to the right into (Q_B Q_C Q_D). The 1at Q_D' (the complement of Q) is shifted back into Q_A. Thus, we start shifting 1s to the right, replacing the 0s. Where a ring counter recirculated a single 1, the 4-stage Johnson counter recirculates four 0s then four 1s for an 8-bit pattern, then repeats.

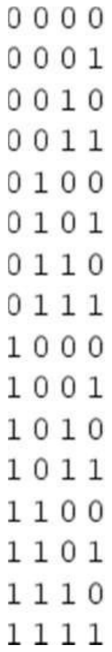


Four stage Johnson counter waveforms

The above waveforms illustrates that multi-phase square waves are generated by a Johnson counter. The 4-stage unit above generates four overlapping phases of 50% duty cycle. How many stages would be required to generate a set of three phase waveforms? For example, a three stage Johnson counter, driven by a 360 Hertz clock would generate three 120^o phased square waves at 60 Hertz.

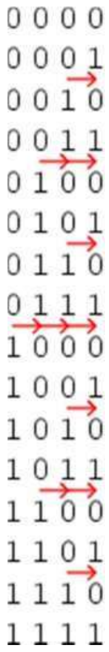
Asynchronous Counters:

Since we know that binary count sequences follow a pattern of octave (factor of 2) frequency division, and that J-K flip-flop multivibrators set up for the toggle mode are capable of performing this type of frequency division, we can envision a circuit made up of several J-K flip-flops, cascaded to produce four bits of output. The main problem facing us is to determine how to connect these flip-flops together so that they toggle at the right times to produce the proper binary sequence. Examine the following binary count sequence, paying attention to patterns preceding the toggling of a bit between 0 and 1:



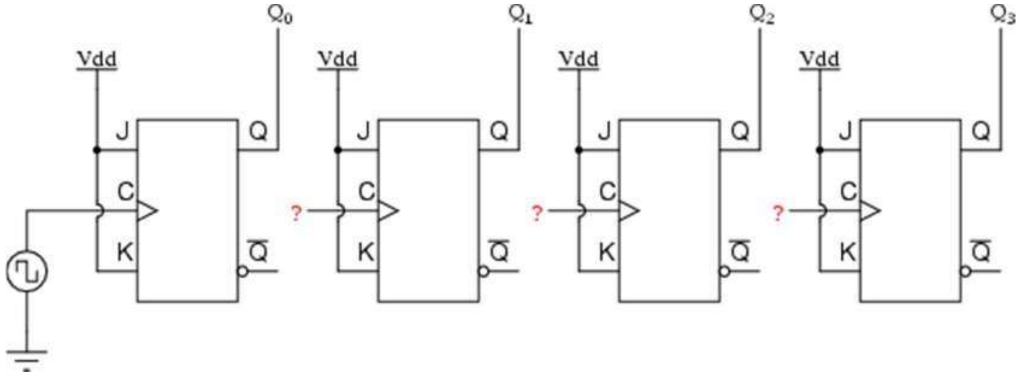
```
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1
```

Note that each bit in this four-bit sequence toggles when the bit before it (the bit having a lesser significance, or place-weight), toggles in a particular direction: from 1 to 0. Small arrows indicate those points in the sequence where a bit toggles, the head of the arrow pointing to the previous bit transitioning from a high (1) state to a low (0) state:



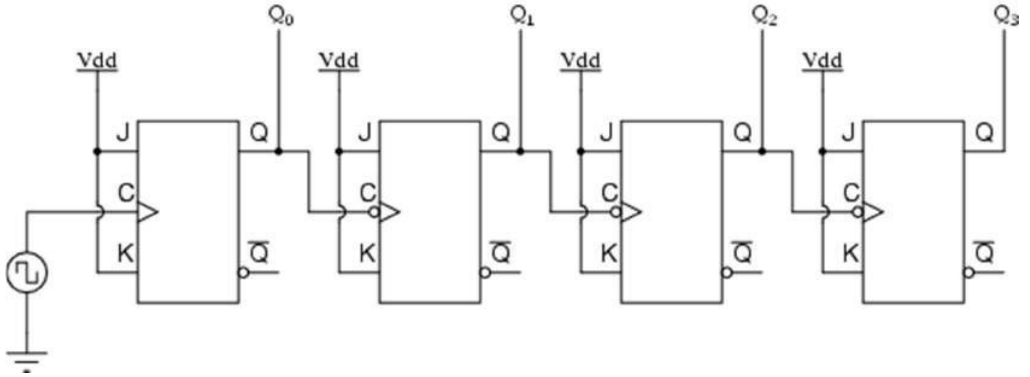
```
0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1
0 1 0 0
0 1 0 1
0 1 1 0
0 1 1 1
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1
```

Starting with four J-K flip-flops connected in such a way to always be in the toggle mode, we need to determine how to connect the clock inputs in such a way so that each succeeding bit toggles when the bit before it transitions from 1 to 0. The Q outputs of each flip-flop will serve as the respective binary bits of the final, four-bit count:

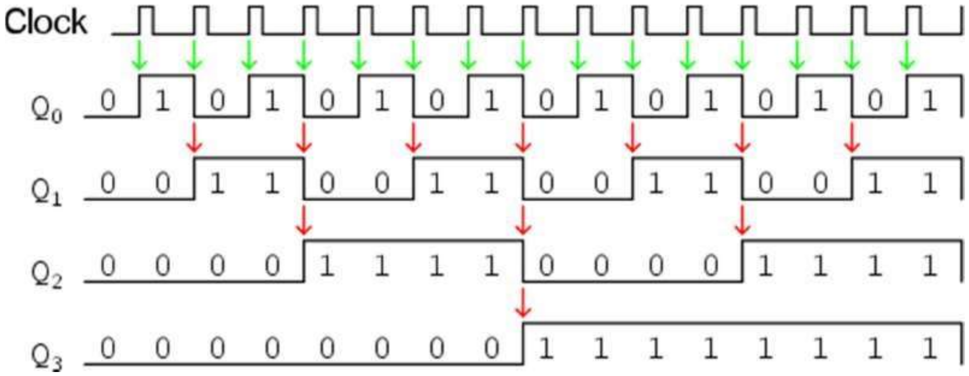


If we used flip-flops with negative-edge triggering (bubble symbols on the clock inputs), we could simply connect the clock input of each flip-flop to the Q output of the flip-flop before it, so that when the bit before it changes from a 1 to a 0, the falling edge of that signal would clock the next flip-flop to toggle the next bit:

A four-bit "up" counter



This circuit would yield the following output waveforms, when clocked by a repetitive source of pulses from an oscillator:

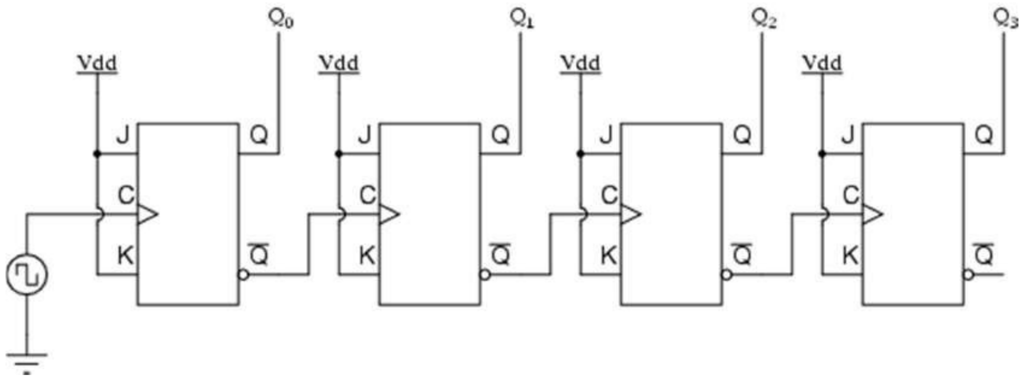

Clock
Q0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
Q1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
Q2 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1
Q3 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1

The first flip-flop (the one with the Q_0 output), has a positive-edge triggered clock input, so it toggles with each rising edge of the clock signal. Notice how the clock signal in this example has a duty cycle less than 50%. I ve shown the signal in this manner for the purpose of demonstrating how the clock signal need not be symmetrical to obtain reliable, clean output bits in our four-bit binary sequence. In the very first flip-flop circuit shown in this chapter, I used the clock signal itself as one of the output bits. This is a bad practice in counter design, though, because it necessitates the use of a square wave signal with a 50% duty cycle (high time = low time) in order to obtain a count sequence where each and every step pauses for the same amount of time. Using one J-K flip-flop for each output bit, however, relieves us of the necessity of having a symmetrical clock signal, allowing the use of practically any variety of high/low waveform to increment the count sequence.

As indicated by all the other arrows in the pulse diagram, each succeeding output bit is toggled by the action of the preceding bit transitioning from high (1) to low (0). This is the pattern necessary to generate an up count sequence.

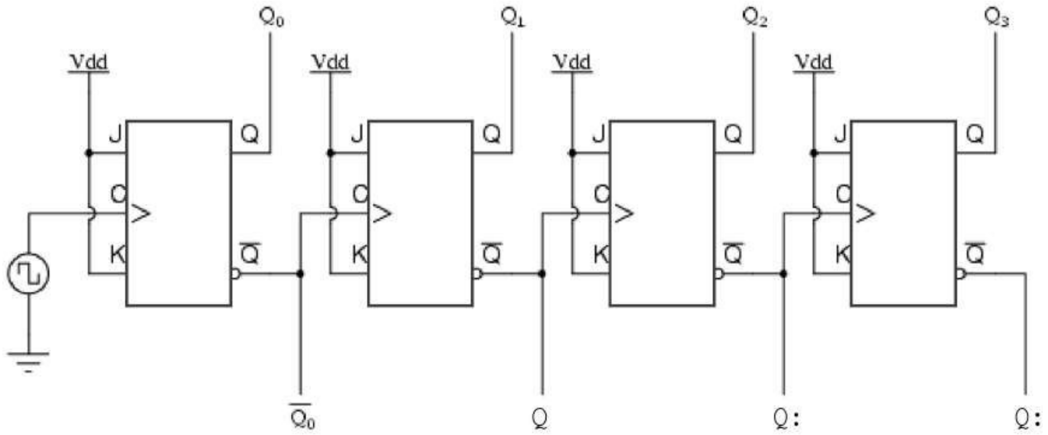
A less obvious solution for generating an up sequence using positive-edge triggered flip- flops is to clock each flip-flop using the Q output of the preceding flip-flop rather than the Q output. Since the Q output will always be the exact opposite state of the Q output on a J-K flip-flop (no invalid states with this type of flip-flop), a high-to-low transition on the Q output will be accompanied by a low-to-high transition on the Q output. In other words, each time the Q output of a flip-flop transitions from 1 to 0, the Q output of the same flip-flop will transition from 0 to 1, providing the positive-going clock pulse we would need to toggle a positive-edge triggered flip-flop at the right moment:

A different way of making a four-bit "up" counter

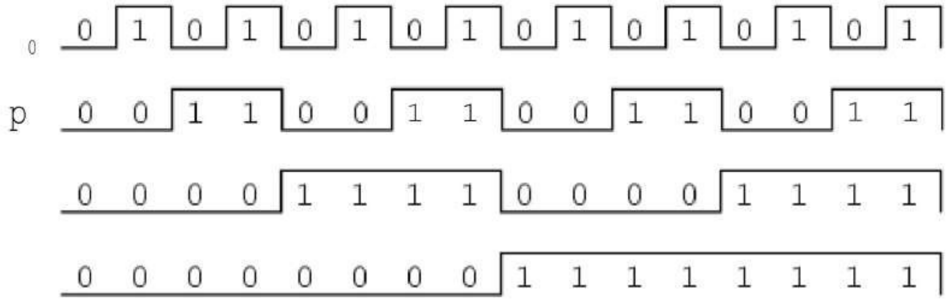


One way we could expand the capabilities of either of these two counter circuits is to regard the Q outputs as another set of four binary bits. If we examine the pulse diagram for such a circuit, we see that the Q outputs generate a down-counting sequence, while the Q outputs generate an up-counting sequence:

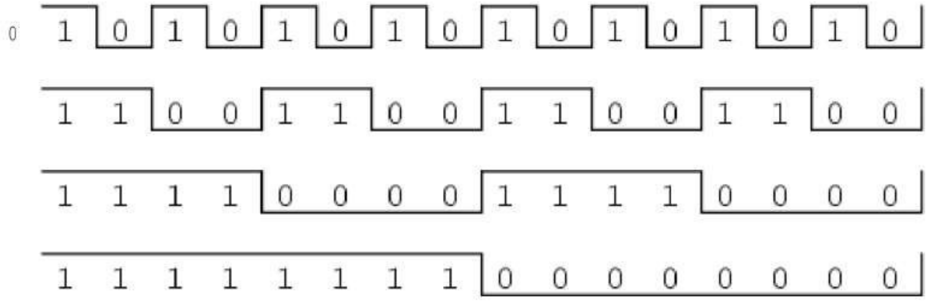
A simuitoneous "up" and ’doon“’ counter



"Up" count sequence



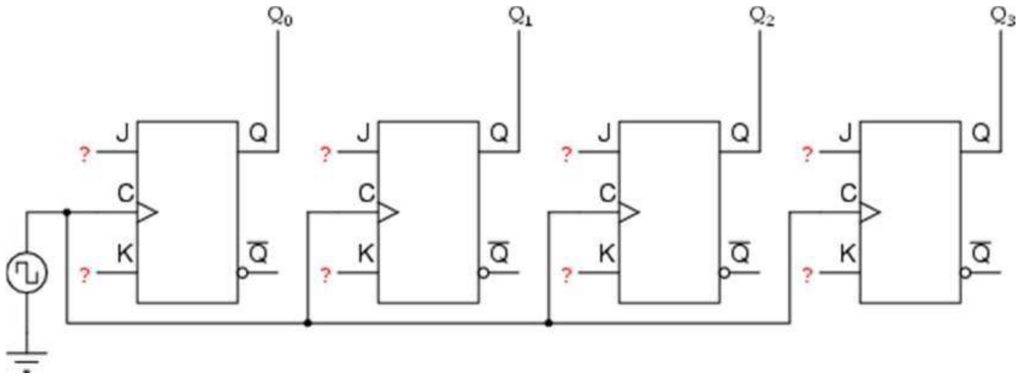
"Down" count sequence



Unfortunately, all of the counter circuits shown thus far share a common problem: the ripple effect. This effect is seen in certain types of binary adder and data conversion circuits, and is due to accumulative propagation delays between cascaded gates. When the Q output of a flip- flop transitions from 1 to 0, it commands the next flip-flop to toggle. If the next flip-flop toggle is a transition from 1 to 0, it will command the flip-flop after it to toggle as well, and so on. However, since there is always some small amount of propagation delay between the command to toggle (the clock pulse) and the actual toggle response (Q and Q outputs changing states), any subsequent flip-flops to be toggled will toggle some time after the first flip-flop has toggled. Thus, when multiple bits toggle in a binary count sequence, they will not all toggle at exactly the same time.

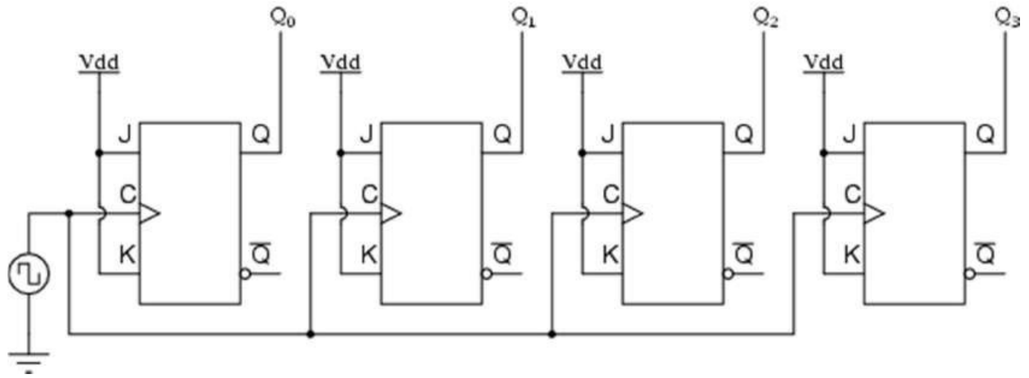
Synchronous Counters:

A synchronous counter, in contrast to an asynchronous counter, is one whose output bits change state simultaneously, with no ripple. The only way we can build such a counter circuit from J-K flip-flops is to connect all the clock inputs together, so that each and every flip-flop receives the exact same clock pulse at the exact same time:



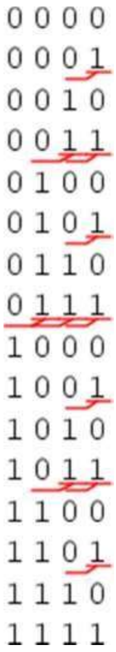
Now, the question is, what do we do with the J and K inputs? We know that we still have to maintain the same divide-by-two frequency pattern in order to count in a binary sequence, and that this pattern is best achieved utilizing the toggle mode of the flip-flop, so the fact that the J and K inputs must both be (at times) high is clear. However, if we simply connect all the J and K inputs to the positive rail of the power supply as we did in the asynchronous circuit, this would clearly not work because all the flip-flops would toggle at the same time: with each and every clock pulse!

This circuit will not function as a counter!



Let s examine the four-bit binary counting sequence again, and see if there are any other patterns that predict the toggling of a bit. Asynchronous counter circuit design is based on the fact that each bit toggle happens at the same time that the preceding bit toggles from a high to a low (from 1 to 0). Since we cannot clock the toggling of a bit based on the toggling of a previous bit in a synchronous counter circuit (to do so would create a ripple effect) we must find some other pattern in the counting sequence that can be used to trigger a bit toggle:

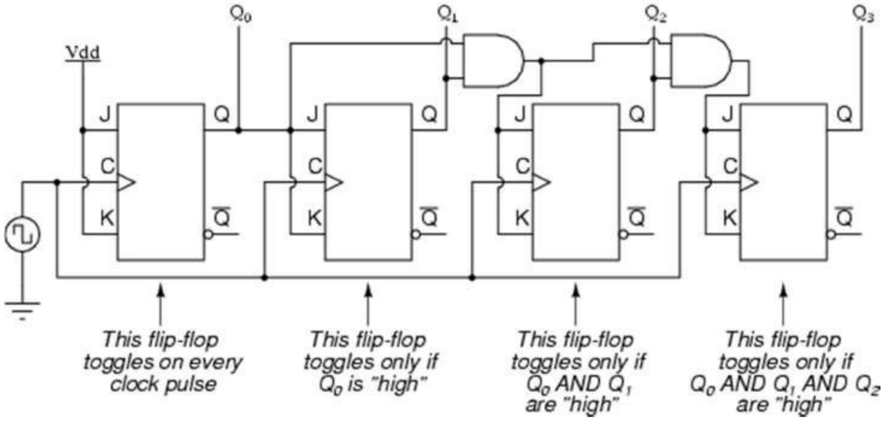
Examining the four-bit binary count sequence, another predictive pattern can be seen. Notice that just before a bit toggles, all preceding bits are high:



0 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 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

This pattern is also something we can exploit in designing a counter circuit. If we enable each J-K flip-flop to toggle based on whether or not all preceding flip-flop outputs (Q) are high, we can obtain the same counting sequence as the asynchronous circuit without the ripple effect, since each flip-flop in this circuit will be clocked at exactly the same time:

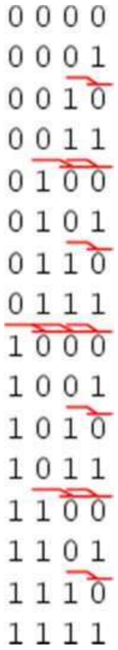
A four-bit synchronous "up" counter



Q₀
Q₁
Q₂
Q₃
Vdd
J
C
K
Q
Q̅
This flip-flop toggles on every clock pulse
This flip-flop toggles only if Q₀ is "high"
This flip-flop toggles only if Q₀ AND Q₁ are "high"
This flip-flop toggles only if Q₀ AND Q₁ AND Q₂ are "high"

The result is a four-bit synchronous up counter. Each of the higher-order flip-flops are made ready to toggle (both J and K inputs high) if the Q outputs of all previous flip-flops are high. Otherwise, the J and K inputs for that flip-flop will both be low, placing it into the latch mode where it will maintain its present output state at the next clock pulse. Since the first (LSB) flip-flop needs to toggle at every clock pulse, its J and K inputs are connected to V_{cc} or V_{dd}, where they will be high all the time. The next flip-flop need only recognize that the first flip-flop s Q output is high to be made ready to toggle, so no AND gate is needed. However, the remaining flip-flops should be made ready to toggle only when all lower-order output bits are high, thus the need for AND gates.

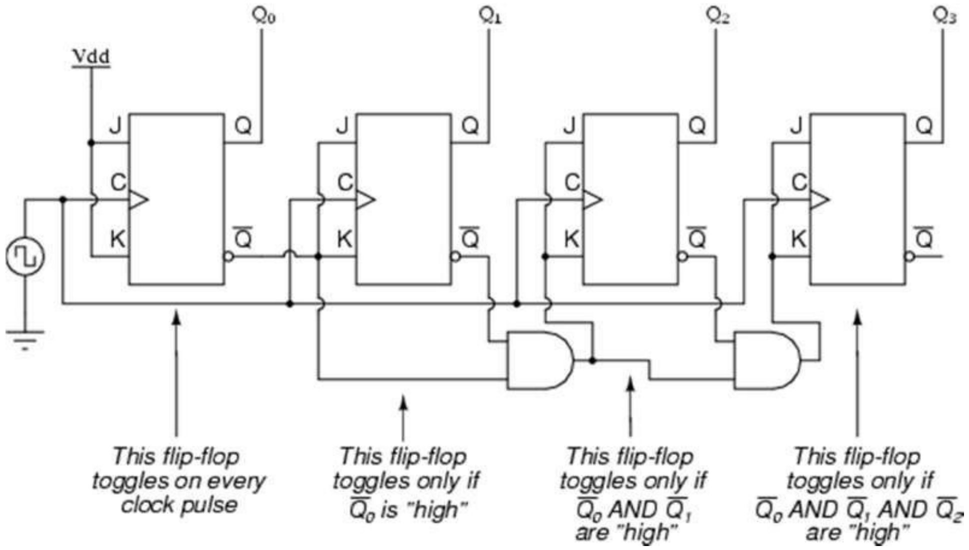
To make a synchronous down counter, we need to build the circuit to recognize the appropriate bit patterns predicting each toggle state while counting down. Not surprisingly, when we examine the four-bit binary count sequence, we see that all preceding bits are low prior to a toggle (following the sequence from bottom to top):



```
0 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 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
```

Since each J-K flip-flop comes equipped with a Q output as well as a Q output, we can use the Q outputs to enable the toggle mode on each succeeding flip-flop, being that each Q will be high every time that the respective Q is low:

A four-bit synchronous "down" counter



This flip-flop toggles on every clock pulse

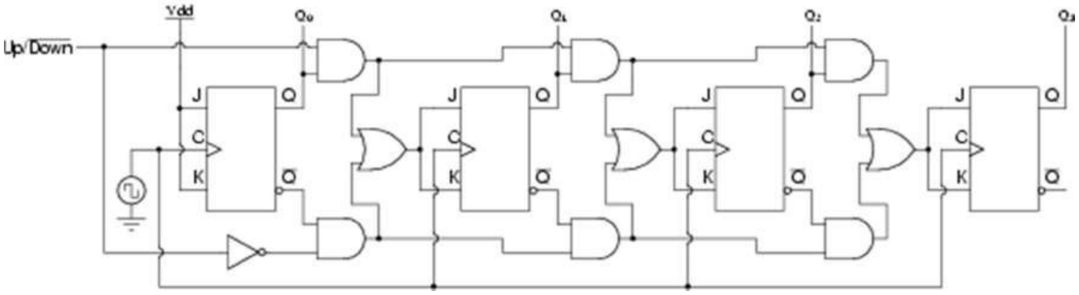
This flip-flop toggles only if $\overline{Q}_0$ is "high"

This flip-flop toggles only if $\overline{Q}_0$ AND $\overline{Q}_1$ are "high"

This flip-flop toggles only if $\overline{Q}_0$ AND $\overline{Q}_1$ AND $\overline{Q}_2$ are "high"

Taking this idea one step further, we can build a counter circuit with selectable between up and down count modes by having dual lines of AND gates detecting the appropriate bit conditions for an up and a down counting sequence, respectively, then use OR gates to combine the AND gate outputs to the J and K inputs of each succeeding flip-flop:

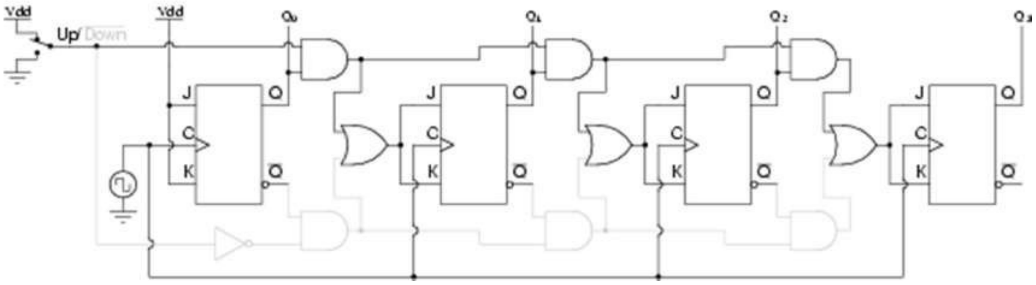
A four-bit synchronous "up/down" counter



This circuit isn t as complex as it might first appear. The Up/Down control input line simply enables either the upper string or lower string of AND gates to pass the Q/Q outputs to the succeeding stages of flip-flops. If the Up/Down control line is high, the top AND gates become enabled, and the circuit functions exactly the same as the first (up) synchronous counter circuit shown in this section. If the Up/Down control line is made low, the bottom AND gates become enabled, and the circuit functions identically to the second (down counter) circuit shown in this section.

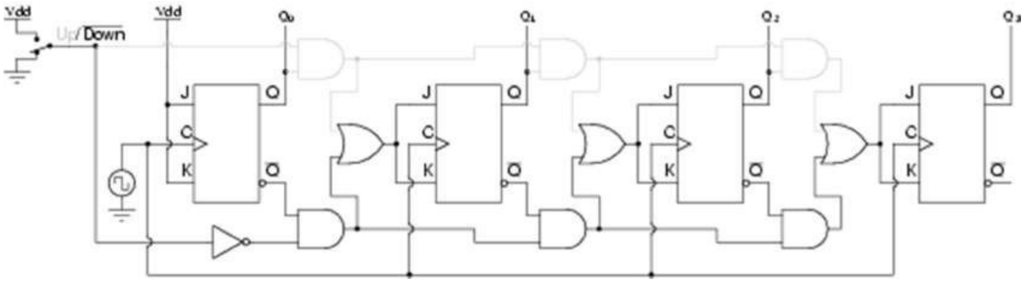
To illustrate, here is a diagram showing the circuit in the up counting mode (all disabled circuitry shown in grey rather than black):

Counter in "up" counting mode



Here, shown in the down counting mode, with the same grey coloring representing disabled circuitry:

Counter in "down" counting mode



Up/down counter circuits are very useful devices. A common application is in machine motion control, where devices called rotary shaft encoders convert mechanical rotation into a series of electrical pulses, these pulses clocking a counter circuit to track total motion.

MOD counters:

Counters of Modulo m”

Counters, either synchronous or asynchronous progress one count at a time in a set binary progression and as a result an n -bit counter functions naturally as a modulo 2^n counter. But we can construct mod counters to count to any value we want by using one or more external logic gates causing it to skip a few output states and terminate at any count resetting the counter back to zero, that is all flip-flops have Q = 0.

In the case of modulo m counters, they do not count to all their possible states, but instead count to the m value and then return to zero. Obviously, m is a number smaller than 2^n , ($m < 2^n$). So how do we get a binary counter to return to zero part way through its count.

Fortunately, as well as counting, up or down, counters can also have additional inputs called CLEAR and PRESET which makes it possible to clear the count to zero, (all Q = 0) or to preset the counter to some initial value. The TTL 74LS74 has active-low Preset and Clear inputs.

Let s assume for simplicity that the CLEAR inputs are all connected together and are active- high inputs allowing the flip-flops to operate normally when the Clear input is equal to 0 (LOW). But if the Clear input is at logic level 1 (HIGH), then the next positive edge of the clock signal will reset all the flip-flops into the state Q = 0, regardless of the value of the next clock signal.

Note also that as all the Clear inputs are connected together, a single pulse can also be used to clear the outputs (Q) of all the flip-flops to zero before counting starts to ensure that the count actually starts from zero. Also some larger bit counters have an additional ENABLE or INHIBIT input pin which allows the counter to stop the count at any point in the counting cycle and hold its present state, before being allowed to continue counting again. This means the counter can be stopped and started at will without resetting the outputs to zero.

MOD-5 Counter Design using JK flip-flop:
Step 1:

Flip flop required

are $2^n \geq N$
Mod 5 hence N=5

n= no. of flip-flop
i.e. 3 flip flop are required Step 2:

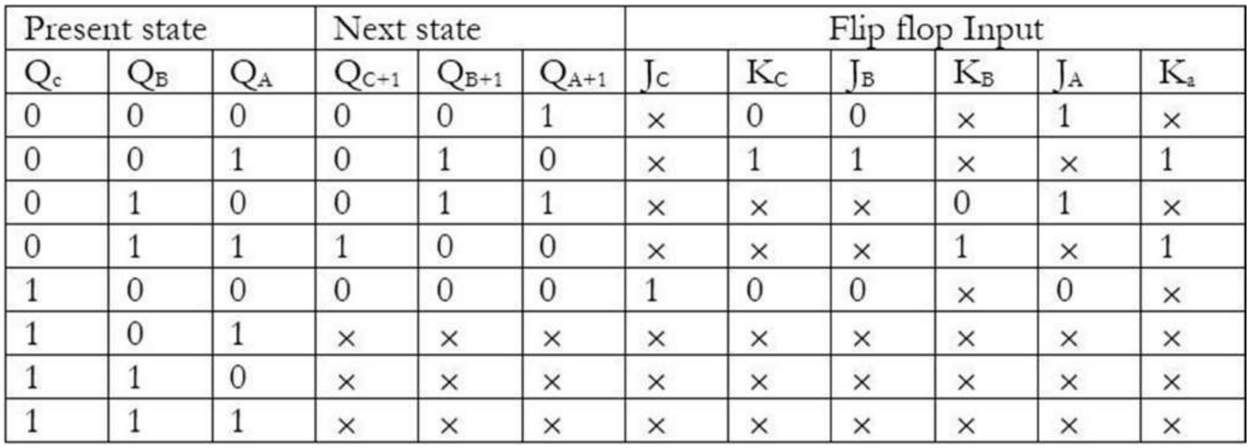
Type of flip flop to be used: JK flip flop Step 3:

1) Excitation table for JK flip flop

Q_n	Q_{n+1}	J	K
0	0	0	×
0	1	1	×
1	0	×	1

Now, we can derive excitation table for counter using above table as follows:

2) Excitation table for counter

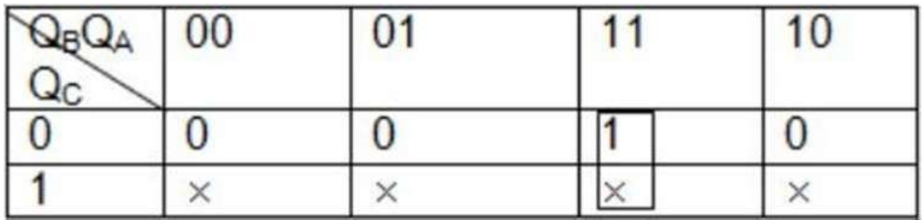


Present state			Next state			Flip flop Input					
Q_c	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_a
0	0	0	0	0	1	×	0	0	×	1	×
0	0	1	0	1	0	×	1	1	×	×	1
0	1	0	0	1	1	×	×	×	0	1	×
0	1	1	1	0	0	×	×	×	1	×	1
1	0	0	0	0	0	1	0	0	×	0	×
1	0	1	×	×	×	×	×	×	×	×	×
1	1	0	×	×	×	×	×	×	×	×	×
1	1	1	×	×	×	×	×	×	×	×	×

Step 4 K-map

simplification For

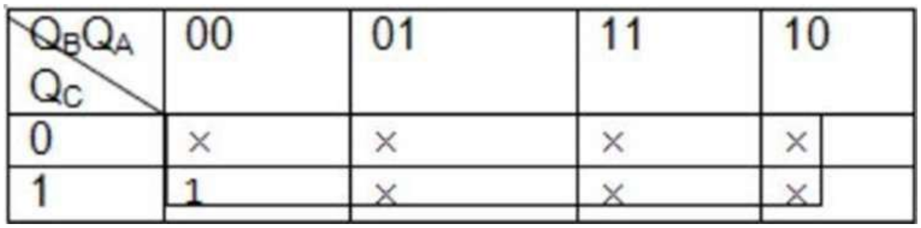
J_C



Q_BQ_A / Q_C	00	01	11	10
0	0	0	1	0
1	×	×	×	×

$J_c = Q_BQ_A$

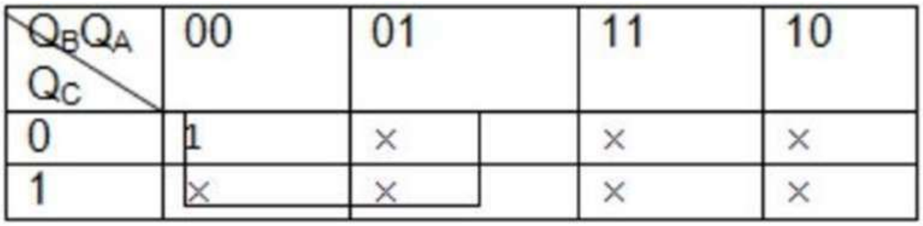
For K_C



Q_BQ_A Q_C	00	01	11	10
0	×	×	×	×
1	1	×	×	×

K_C=1

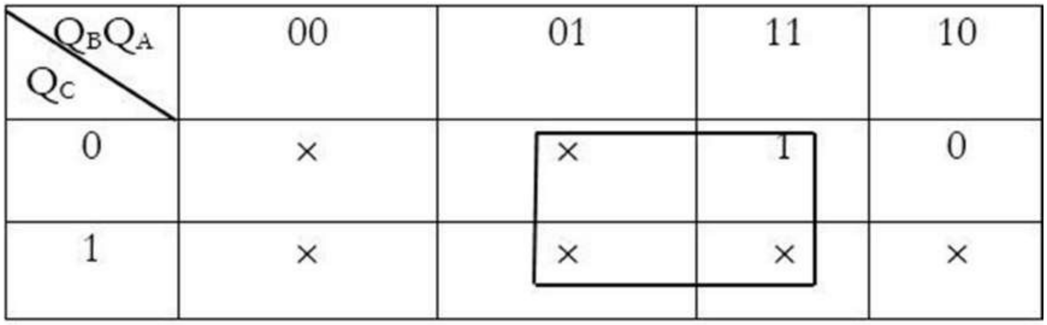
For J_B



Q_BQ_A Q_C	00	01	11	10
0	1	×	×	×
1	×	×	×	×

J_B=Q_A

ForK_B



Q_BQ_A Q_C	00	01	11	10
0	×	×	1	0
1	×	×	×	×

K_B=Q_A

For J_A

Q_BQ_A / Q_C	00	01	11	10
0	1	×	×	1
1	0	×	×	×

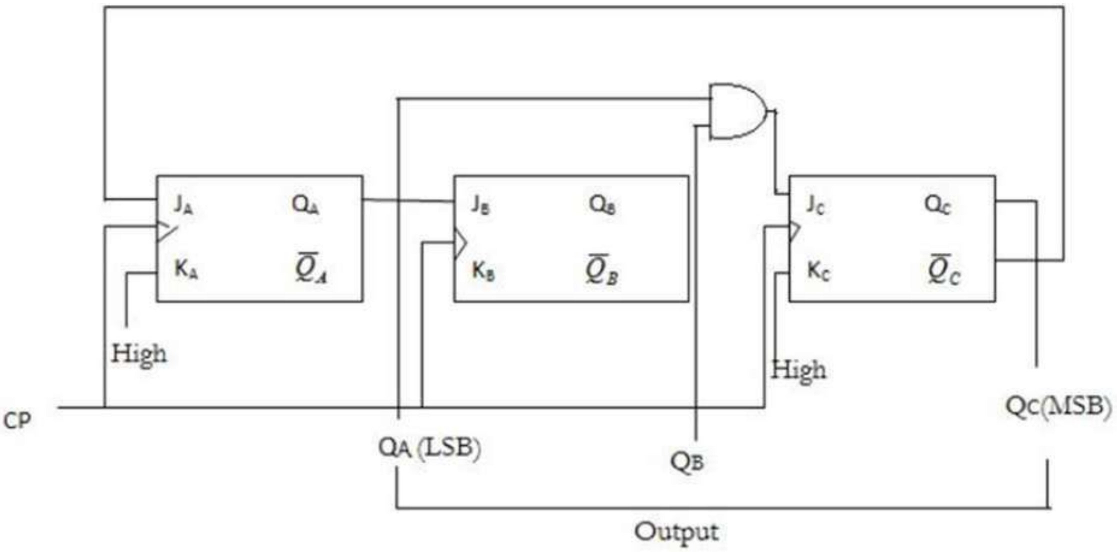
J_A=Q_C'

For K_A

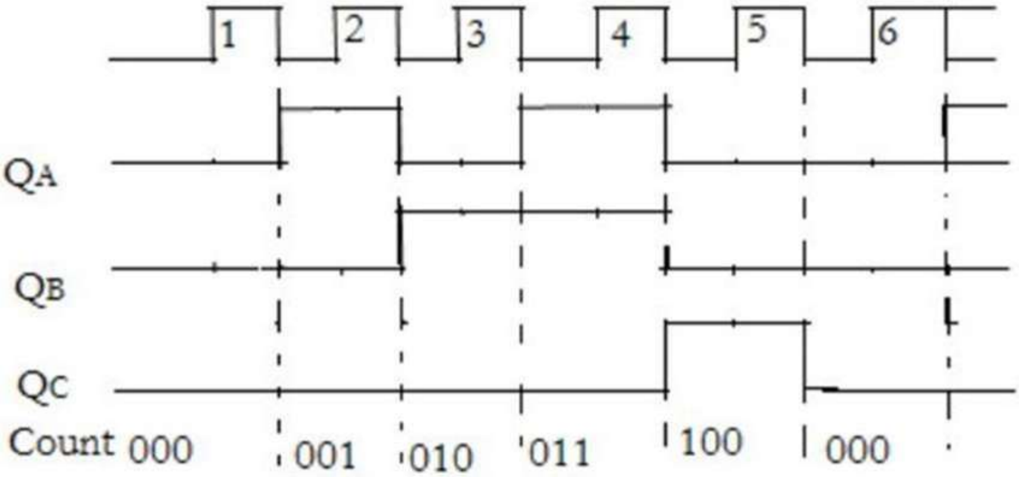
Q_BQ_A / Q_C	00	01	11	10
0	×	1	1	×
1	×	×	×	×

K_A=1

Step 5 Logic Diagram



Step 6: Timing Diagram



Multiple Choice Question:

1. Why is the extent of propagation delay in synchronous counter much lesser than that of asynchronous counter?
a. Due to clocking of all flip flops at the same instant
b. Due to increase in number of states
c. Due to absence of connection between output of preceding flip flop and clock of next one
d. Due to absence of mode control operation

2. Which flip flops serve to be the fundamental building blocks of counters? a. S-R flip
flops
b. J-K flip flops
c. T flip flops
d. D flip flops

3. Which type of triggering phenomenon is exhibited by Counters? a. Edge
b. Level
c. Pulse
d. All of the above

4. If the number of states in a counter are 2^n, then the value of 'n' is ________
a. Less than the number of flip flops
b. Greater than the number of flip flops
c. Equal to the number of flip flops
d. Unpredictable

Sample Question:

1. Draw the circuit for 4 bit Johnson counter using D flip-flop and explain operation. Draw the timing diagram for this 4 bit Johnson counter. How does this timing diagram differ from the ring counters?
2. Design a 4-Bit asynchronous up counter using flip-flop.
3. What is ripple counter?
4. Explain the shift registers with timing diagram.

Assignment:

1. Design a synchronous MOD 10 up counter using D flip-flop and draw the timing diagram.
2. Design a MOD 6 synchronous binary up counter using JK flip flop with other necessary logic gates.

References:

1. Digital Principles & Applications, A.P.Malvino, D.P.Leach & Saha, 7th Ed., 2011, Tata McGraw
2. Fundamentals of Digital Circuits, A. Anand Kumar, 2nd Edition, 2009, PHI Learning Pvt. Ltd.
3. Digital Electronics, Principles and applications, Roger L Tokheim, 2003, Tata McGraw Hill.
4. Digital Systems: Principles and Applications, R.J. Tocci, N.S.Widmer, 2001, PHI Learning.
5. Digital Electronics, An introduction to theory and practice, W H Gothmann, 1982, PHI Learning.
6. http://www.tutorialspoint.com/computer_logical_organization/combinational_circuits.htm
7. http://en.wikipedia.org/wiki/Carry-lookahead_adde
8. MANO, M. M. AND C. R. KIME. Logic and Computer Design Fundamentals, 3rd ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2004
9. Digital Circuits & Design, S Salivahanan& S A Arivazhagan
10. http://www.exploreroots.com
11. http://www.electronicshub.org
12. http://www.electronics-tutorials.ws
13. https://www.allaboutcircuits.com

Module IV

Analog to Digital Converters (A/D)

Parameters of ADC

- Resolution: In ADC, the original analog signal has essentially an infinite resolution as the signal is continuous. The digital representation of this signal would, of course, reduce this resolution as digital quantities are discrete and vary in equal steps. The resolution of an ADC is the smallest change that can be distinguished in the analog input. Resolution = FSR (Full Scale Range) / 2n□

- Conversion Time: The A/D conversion another critical parameter is conversion time. This is defined as the total time required converting an analog signal into its digital output.□

- Accuracy: It is the comparison of the actual output and the expected output.□

- Linearity: The output should be the linear function of the input.□

- Full-scale output value: The maximum bit output achieved from the respective input□

Understanding Flash ADCs

Abstract: Flash analog-to-digital converters, also known as parallel ADCs, are the fastest way to convert an analog signal to a digital signal. Flash ADCs are ideal for applications requiring very large bandwidth, but they consume more power than other ADC architectures and are generally limited to 8-bit resolution. This tutorial will discuss flash converters and compare them with other converter types.

Introduction

Flash analog-to-digital converters, also known as parallel ADCs, are the fastest way to convert an analog signal to a digital signal. Flash ADCs are suitable for applications requiring very large bandwidths. However, these converters consume considerable power, have relatively low resolution, and can be quite expensive. This limits them to high-frequency applications that typically cannot be addressed any other way. Typical examples include data acquisition, satellite communication, radar processing, sampling oscilloscopes, and high-density disk drives.

This tutorial will discuss flash converters and compare them with other converter types.

Architectural Details

Flash ADCs are made by cascading high-speed comparators. Figure 1 shows a typical flash ADC block diagram. For an N-bit converter, the circuit employs 2^N-1 comparators. A resistive-divider with 2^N resistors provides the reference voltage. The reference voltage for each comparator is one least significant bit (LSB) greater than the reference voltage for the comparator immediately below it. Each comparator produces a 1 when its analog input voltage is higher than the reference voltage applied to it. Otherwise, the comparator output is 0. Thus, if the analog input is between V_{X4} and V_{X5}, comparators X_1 through X_4 produce 1s and the remaining comparators produce 0s. The point where the code changes from ones to zeros is the point at which the input signal becomes smaller than the respective comparator reference-voltage levels.

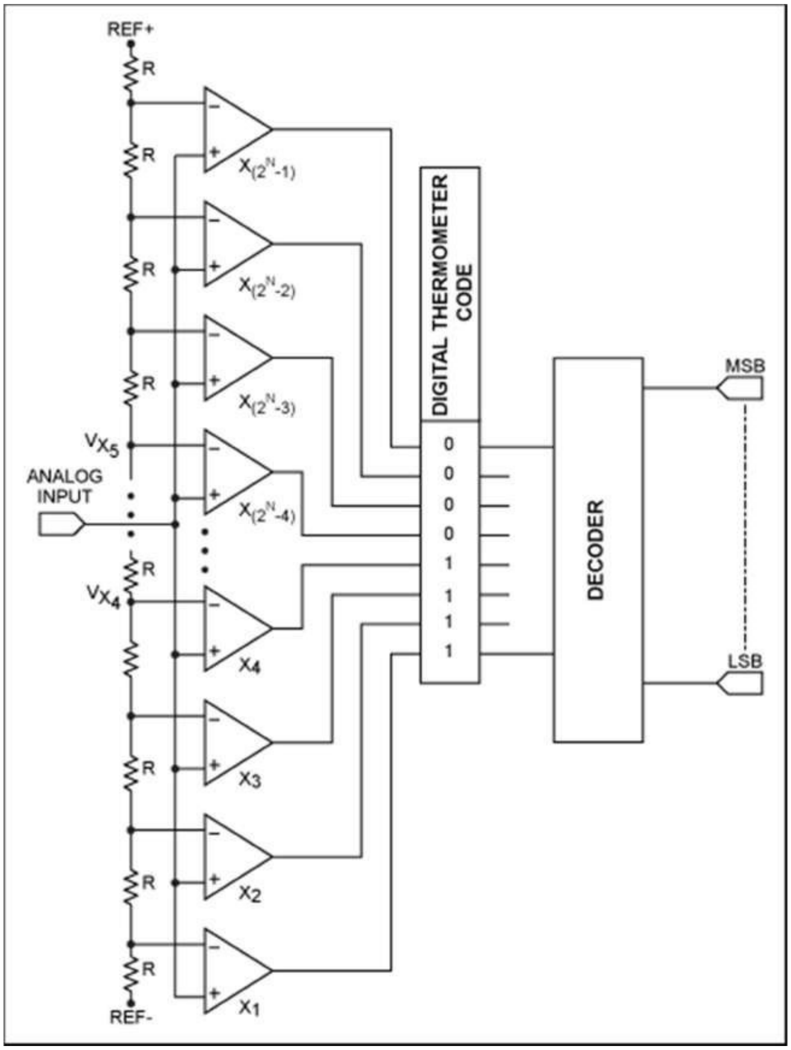


figure 1. Flash ADC architecture. If the analog input is between V_{X4} and V_{X5}, comparators X_1 through X_4 produce 1s and the remaining comparators produce 0s.

This architecture is known as thermometer code encoding. This name is used because the design is similar to a mercury thermometer, in which the mercury column always rises to the appropriate temperature and no mercury is present above that temperature. The thermometer code is then decoded to the appropriate digital output code.

The comparators are typically a cascade of wideband low-gain stages. They are low gain because at high frequencies it is difficult to obtain both wide bandwidth and high gain. The comparators are designed for low-voltage offset, so that the input offset of each comparator is smaller than an LSB of the ADC. Otherwise, the comparator's offset could falsely trip the comparator, resulting in a digital output code that is not representative of a thermometer code. A regenerative latch at each comparator output stores the result. The latch has positive feedback, so that the end state is forced to either a 1 or a 0.

Given these basics, some adjustments are needed to optimize the flash converter architecture.

Sparkle Codes

Normally, the comparator outputs will be a thermometer code, such as 00011111. Errors can cause an output like 00010111, meaning that there is a spurious zero in the result. This out-of-sequence 0 is called a sparkle, which is caused by imperfect input settling or comparator timing mismatch. The magnitude of the error can be quite large. Modern converters like the MAX109/MAX104 employ an input track-and-hold in front of the ADC along with an encoding technique that suppresses sparkle codes.

Metastability

When the digital output from a comparator is ambiguous (neither a 1 nor a 0), the output is defined as metastable. Metastability can be reduced by allowing more time for regeneration. Gray-code encoding, which allows only 1 bit in the output to change at a time, can greatly improve metastability. . Thus, the comparator outputs are first converted to gray-code encoding and then later decoded to binary, if desired.

Another problem occurs when a metastable output drives two distinct circuits. It is possible for one circuit to declare the input a 1, while the other circuit thinks that it is a 0. This can create major errors. To avoid this conflict, only one circuit should sense a potentially mestatable output.

Input Signal-Frequency Dependence

When the input signal changes before all the comparators have completed their tasks, the ADC's performance is adversely impacted. The most serious impact is a drop-off in signal-to-noise ratio (SNR) plus distortion (SINAD) as the frequency of the analog input frequency increases.

Measuring spurious-free dynamic range (SFDR) is another good way to observe converter performance. The "effective bits" achieved by the ADC is a function of input frequency; it can be improved by adding a track-and-hold (T/H) circuit in front of the ADC. The T/H circuit allows dramatic improvement, especially when input frequencies approach the Nyquist frequency, as shown in Figure 2 (taken from the MAX104 data sheet). Parts without T/H show a significant drop-off in SFDR.

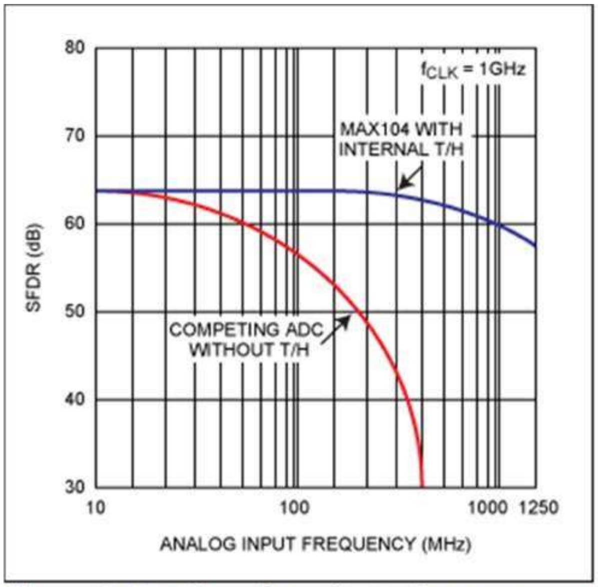


Figure 2. Spurious-free dynamic range as a function of input frequency.

SNR is degraded when there is jitter in the sampling clock. This becomes noticeable for high analog-input frequencies. To achieve accurate results, it is critical to provide the ADC with a low-jitter, sampling clock source.

Architectural Trade-Offs

ADCs can be implemented by employing a variety of architectures. The principal trade-offs among these alternatives are:

- The time it takes to complete a conversion (conversion time). For flash converters, the conversion time does not change materially with increased resolution. The conversion time for successive approximation register (SAR) or pipelined converters, however, increases approximately linearly with an increase in resolution (Figure 3a). For integrating ADCs, the conversion time doubles with every bit increase in resolution.

- Component matching requirements in the circuit. Flash ADC component matching typically limits resolution to around 8 bits. Calibration and trimming are sometimes used to improve the matching available on chip. Component matching requirements double with every bit increase in resolution. This pattern applies to flash, successive approximation, or pipelined converters, but not to integrating converters. For integrating converters, component matching does not materially increase with an increase in resolution (Figure 3b).

- Die size, cost, and power. For flash converters, every bit increase in resolution almost doubles the size of the ADC core circuitry. The power also doubles. In contrast, a SAR, pipelined, or sigma-delta ADC die size will increase linearly with an increase in resolution; an integrating converter core die size will not materially change with an increase in resolution (Figure 3c). Finally, it is well known that an increase in die size increases cost.

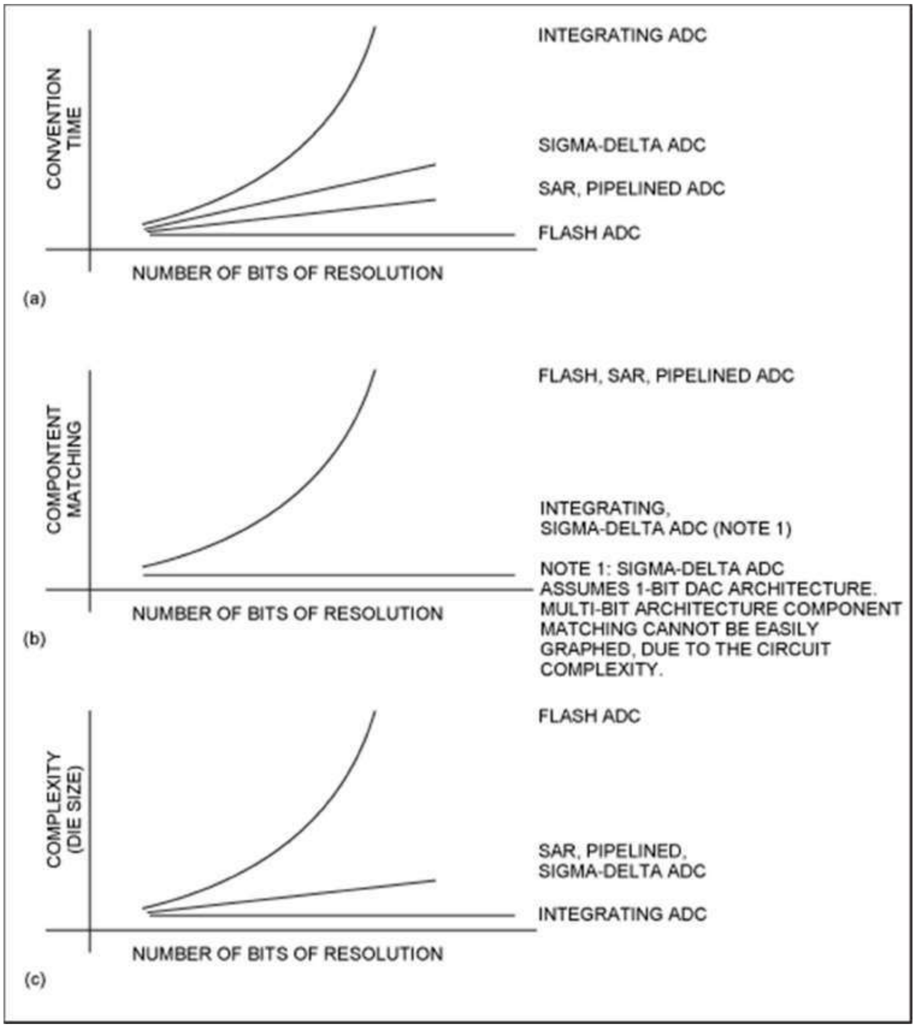
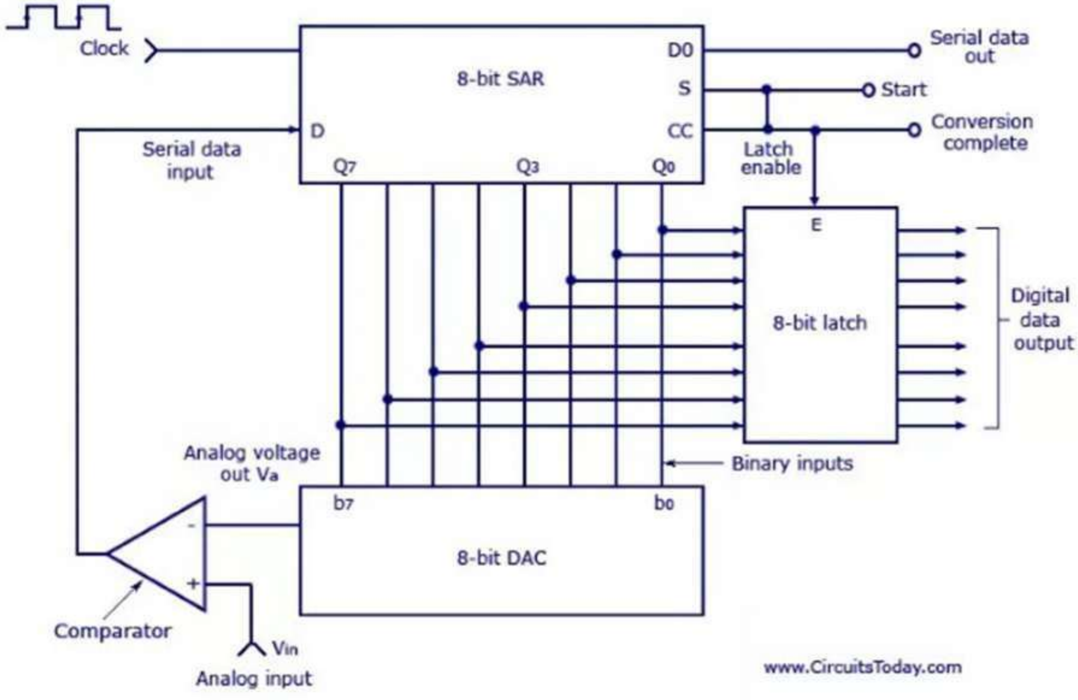


Figure 3. Architectural trade-offs.

This type of converter is used to convert analog voltage to its corresponding digital output. The function of the analog to digital converter is exactly opposite to that of a DIGITAL TO ANALOG CONVERTER. Like a D/A converter, an A/D converter is also specified as 8, 10, 12 or 16 bit. Though there are many types of A/D converters, we will be discussing only about the successive approximation type.

Successive Approximation Type Analog to Digital Converter

A successive approximation A/D converter consists of a comparator, a successive approximation register (SAR), output latches, and a D/A converter. The circuit diagram is shown below.



Successive Approximation Type Analog to Digital Converter

At the start of a conversion cycle, the SAR is reset by making the start signal (S) high. The MSB of the SAR (Q7) is set as soon as the first transition from LOW to HIGH is introduced. The output is given to the D/A converter which produces an analog equivalent of the MSB and is compared with the analog input V_{in}.

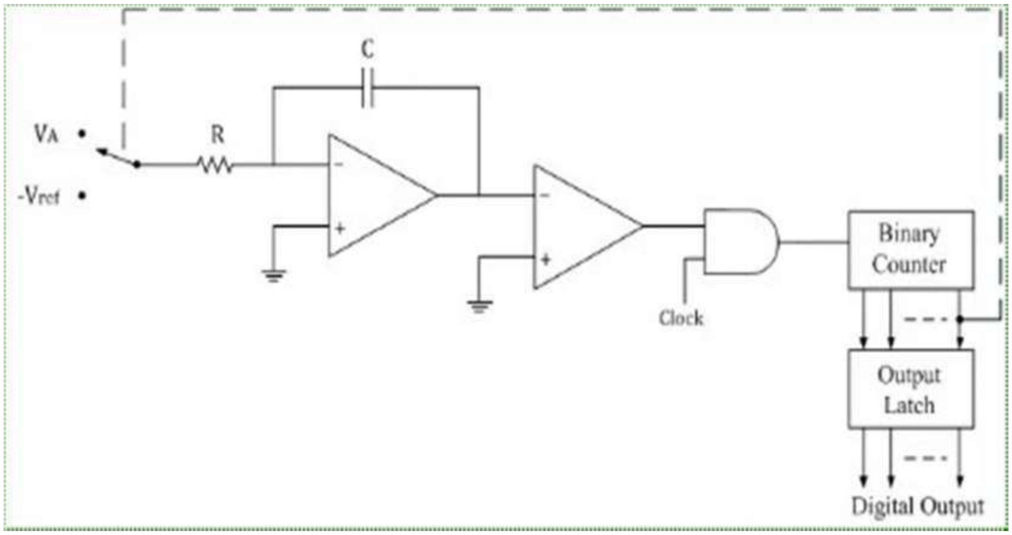
If comparator output is LOW, D/A output will be greater than V_{in} and the MSB will be cleared by the SAR. If comparator output is HIGH, D/A output will be less than V_{in} and the MSB will be set to the next position (Q7 to Q6) by the SAR.

According to the comparator output, the SAR will either keep or reset the Q6 bit. This process goes on until all the bits are tried. After Q0 is tried, the SAR makes the conversion complete (CC) signal HIGH to show that the parallel output lines contain valid data. The CC signal in turn enables the latch, and digital data appear at the output of the latch. As the SAR determines each bit, digital data is also available serially. As shown in the figure above, the CC signal is connected to the start conversion input in order to convert the cycle continuously.

The biggest advantage of such a circuit is its high speed. It may be more complex than an A/D converter, but it offers better resolution.

DUAL SLOPE TYPE ADC

In dual slope type ADC, the integrator generates two different ramps, one with the known analog input voltage VA and another with a known reference voltage Vref. Hence it is called a s dual slope A to D converter. The logic diagram for the same is shown below.



Operation:

The binary counter is initially reset to 0000; the output of integrator reset to 0V and the input to the ramp generator or integrator is switched to the unknown analog input voltage VA.

The analog input voltage VA is integrated by the inverting integrator and generates a negative ramp output. The output of comparator is positive and the clock is passed through the AND gate. This results in counting up of the binary counter.

The negative ramp continues for a fixed time period t1, which is determined by a count detector for the time period t1. At the end of the fixed time period t1, the ramp output of integrator is given by

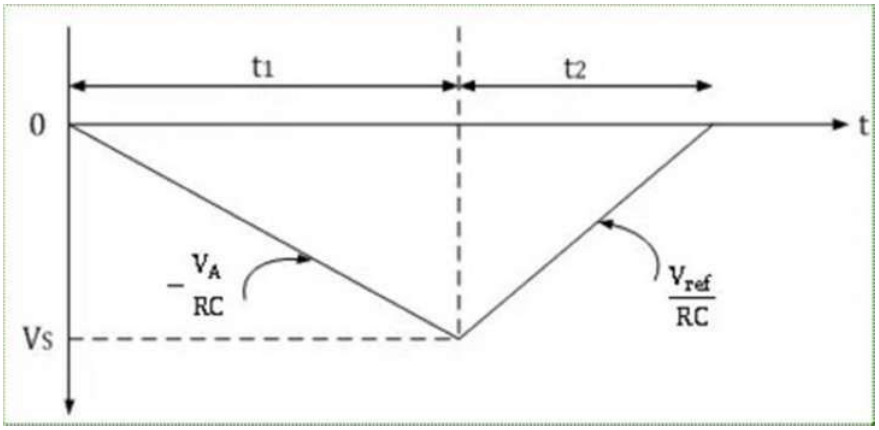
∴ VS=-VA/RC×t1

When the counter reaches the fixed count at time period t1, the binary counter resets to 0000 and switches the integrator input to a negative reference voltage –Vref.

Now the ramp generator starts with the initial value –Vs and increases in positive direction until it reaches 0V and the counter gets advanced. When Vs reaches 0V, comparator output becomes negative (i.e. logic 0) and the AND gate is deactivated. Hence no further clock is applied through AND gate. Now, the conversion cycle is said to be completed and the positive ramp voltage is given by

∴ VS=Vref/RC×t2

Where Vref & RC are constants and time period t2 is variable. The dual ramp output waveform is shown below.



Since ramp generator voltage starts at 0V, decreasing down to –Vs and then increasing up to 0V, the amplitude of negative and positive ramp voltages can be equated as follows.
∴ $Vref/RC \times t2=-VA/RC \times t1$
∴ $t2=-t1 \times VA/Vref$
∴ $VA=-Vref \times t1/t2$

Thus the unknown analog input voltage VA is proportional to the time period t2, because Vref is a known reference voltage and t1 is the predetermined time period.
The actual conversion of analog voltage VA into a digital count occurs during time t2. The binary counter gives corresponding digital value for time period t2. The clock is connected to the counter at the beginning of t2 and is disconnected at the end of t2. Thus the counter counts digital output as Digital output=(counts/sec) t2
∴ Digital output=(counts/sec)[$t1 \times VA/Vref$]
For example, consider the clock frequency is 1 MHz, the reference voltage is -1V, the fixed time period t1 is 1ms and the RC time constant is also 1 ms. Assuming the unknown analog input voltage amplitude as VA = 5V, during the fixed time period t1 , the integrator output Vs is
∴ $VS=-VA/RC \times t1=(-5)/1ms \times 1ms=-5V$
During the time period t2, ramp generator will integrate all the way back to 0V.
∴ $t2=VS/Vref \times RC=(-5)/(-1) \times 1ms=5ms=5000\mu s$
Hence the 4-bit counter value is 5000, and by activating the decimal point of MSD seven segment displays, the display can directly read as 5V.

Digital to Analog Converters (D/A)

Parameters (Characteristics) of DAC

- Resolution: It is determined by the number of bits in the input binary word. A 12-bit converter has a resolution of 1 part in 212.□

- Full-scale output voltage: The maximum output voltage of a converter (when all input is 1) will always have a value 1 LSB less than the named value.□

- Accuracy: The actual output voltage of a DAC is different from the ideal value; the factors that contribute to the lack of linearity also contribute to the lack of accuracy. The accuracy of a DAC is the measure of the difference between actual output voltage and the expected output voltage. For an example, a DAC with□ ±0.2% accuracy and full scale (maximum) output voltage of 10V will produce a maximum error for an output voltage is 20 mV. [0.2/100 * 10V = 0.002*10 V = 20mV]

- Linearity: An ideal DAC should be linear i.e. the output voltage should be a linear function of the input code. All DAC depart somewhat from the ideal linearity. Typical factors responsible for introducing non- linearity are the non-exact value of resistors and non-ideal electronic switches that introduce extra resistance to the circuit. The non-linearity (linearity error) is the amount by which the actual output differs from the ideal straight line output.□

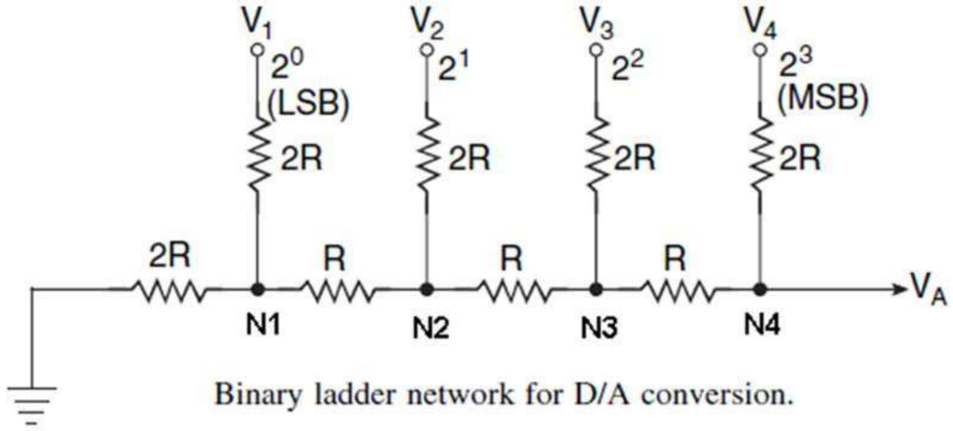
- Settling time:When the output of DAC changes from one value to another, it typically overshoots the new value and may oscillate briefly around that new value before it settles to a constant value. It is the time interval between the instant when the analog input passes a specified value and the time instant when the analog output enters for the last time a specified error band about its final value.□

- Monotonicity: A converter is said to be monotonic if its output voltage value continuous to increase with a continuously increasing input value.□

- Temperature Coefficient: It is defined as the degree of inaccuracy that the temperature change can cause in any of the parameters of the DAC□

R-2R ladder D/A converter

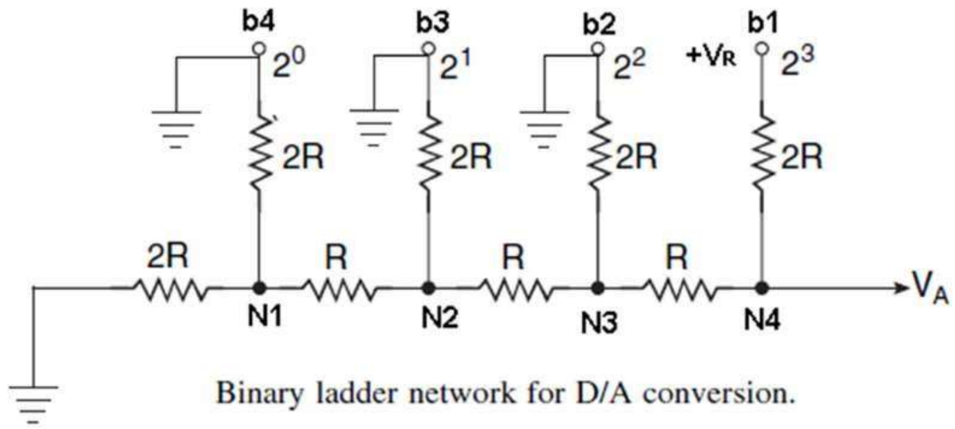
To overcome huge range of resistor used in weighted resistor D/A converter, R-2R ladder D/A converter is introduced. In my previous post I discussed about weighted resistor D/A converter. But the vital problem in weighted register D/A converter is use of huge range of different resistance. Suppose we have to design 8-bit weighted register D/A converter then we need the resistance value $2^0R+2^1R+ .+2^7R$. So the largest resistor corresponding to bit b_8 is 128 times the value of the smallest resistor correspond to b_1. But in case of R-2R ladder D/A converter, Resistors of only two value (R and 2R) are used. Now in bellow see the simple ladder network.



Binary ladder network for D/A conversion.

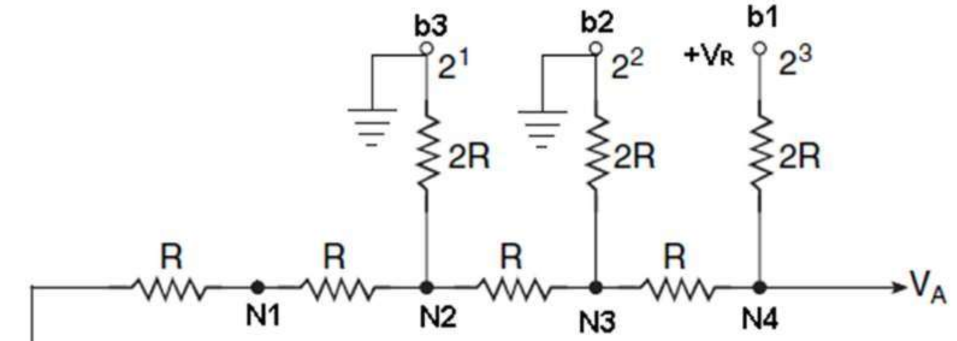
In ladder circuit the output voltage is also weighted sum of the corresponding digital input. Let take an example to understand how it works? As we can see the above network is a 4-bit ladder network so we take an example to convert analog signal correspond of 1000 digital bit. For 1000 bit we can see only MSB got 1 and rest all bits got 0.

See the bellow picture to understand how it work if it got 1000.



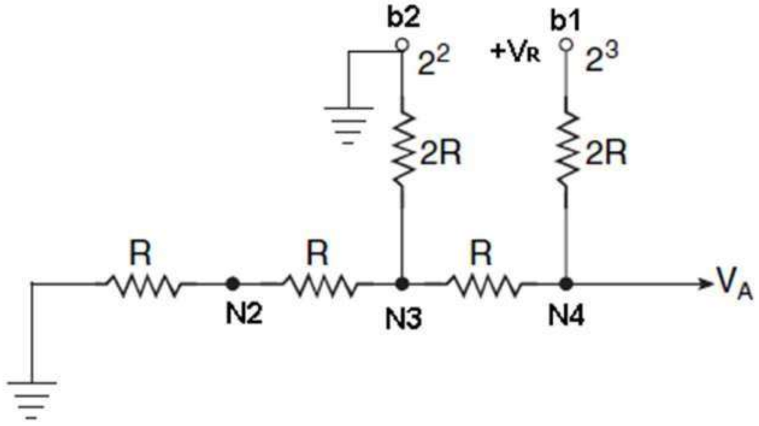
Binary ladder network for D/A conversion.

Now see at node1 (N1) resistor 2R connecting in b4 parallel with resistor 2R. And those 2R parallel 2R resistors make equivalent register of R shown in bellow diagram.



Binary ladder network for D/A conversion.

Now for N2 same thing happen B3 series with 2R and parallel with R + R resistors. It will also make equivalent resistor R at N3. See the bellow diagram.



Binary ladder network for D/A conversion.

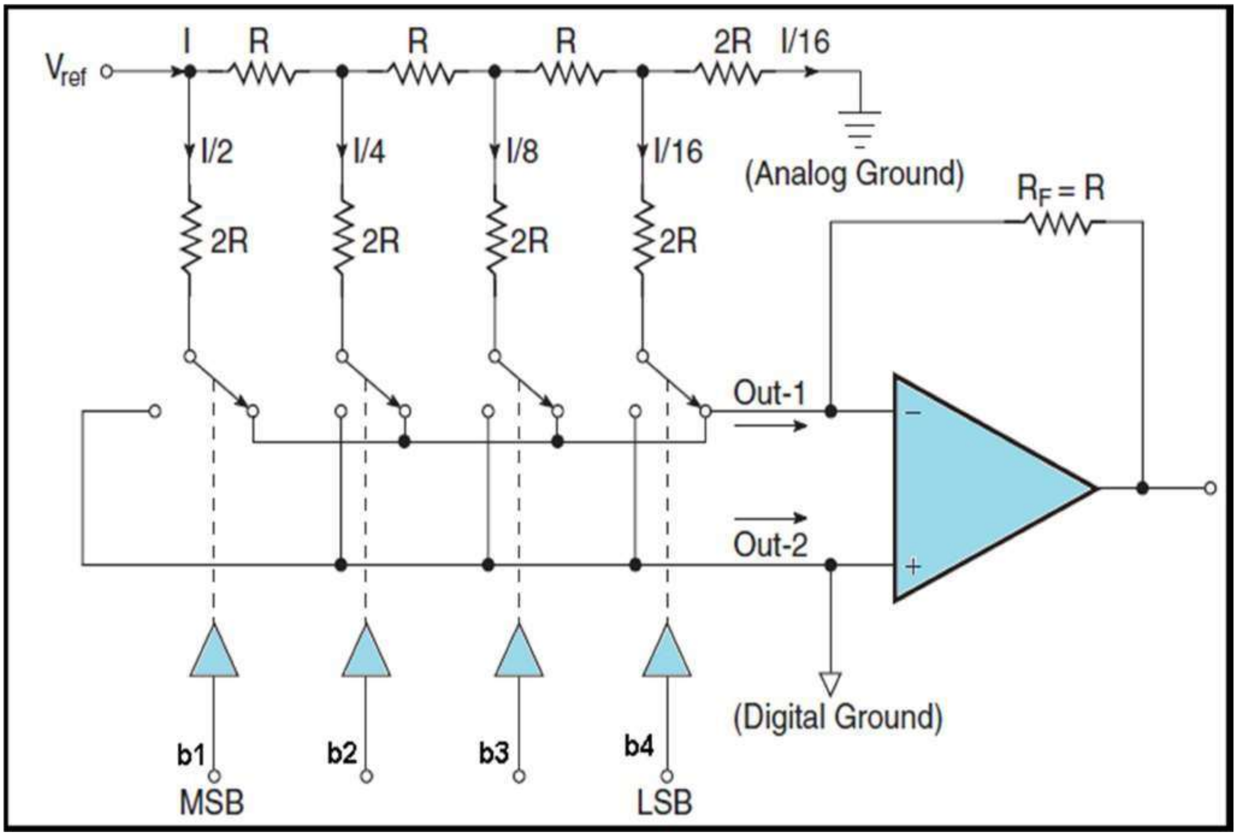
Repeating the same process we got equivalent of R resistor at N4.

Now at N4, if we calculate the output analog equivalent voltage then we will get
$V_A = V_R$*2R/(R+R+2R)
$= V_R/2$
Thus when bit 1000 the output is $V_R/2$. Similarly it can be found that using above process for bit 0100 the output will be $V_R/4$, for bit 0010 output will be VR/8 and for bit 0001 output will be VR/16.
By using superposition theorem we can find in any n-bit ladder network the output voltage will be

$V_A = V_R/2 + V^1{}_R/2 + V^2{}_R/2 + {}^3 \; . + V_R{}^n/2$

Where n is the total number of bits at the input.

Now see the practical circuit arrangement of 4-bit R-2R ladder D/A converter using op amp.



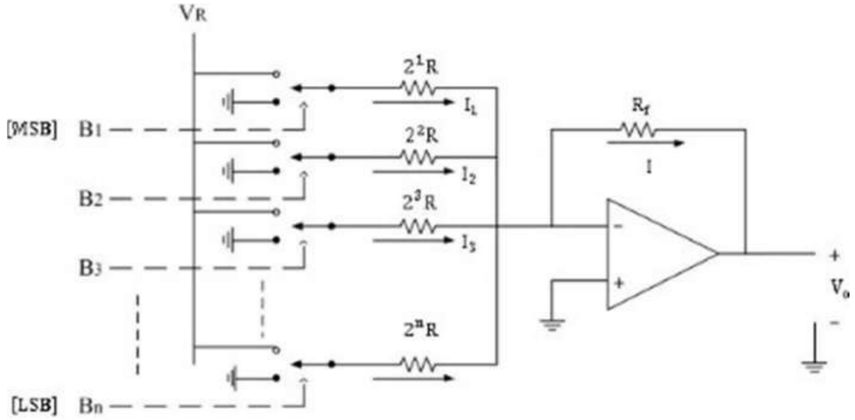
The inverting input terminal of the op amp work as a summing amplifier for the ladder inputs. So we can get out put voltage by bellow equation.

$V0 = V_R{}^*(R_F/R)[b1/2 + b2/2 + b3/2 + b4/2\ ]\ {}^1 \quad {}^2 \quad {}^3 \quad {}^4$

Binary Weighted Resistor DAC

In the weighted resistor type DAC, each digital level is converted into an equivalent analog voltage or current.

The following figure shows the circuit diagram of the binary weighted resistor type DAC.



It consists of parallel binary weighted resistor bank and a feedback resistor Rf.
The switch positions decides the binary word (i.e. B1 B2B3 Bn). In the circuit op-amp is used as current to voltage converter.

Analysis:
Let us analyze the circuit using normal analysis concepts used in op-amp. When the switches are closed the respective currents are flowing through resistors as shown in the circuit diagram above. Since input current to the op-amp is zero, the addition current flows through feedback resistor.

∴ I=I1+I2+I3+ .... +In

The inverting terminal of op-amp is virtually at ground potential.

$$\therefore V_o = -IR_f$$

$$\therefore V_o = -[I_1 + I_2 + I_3 + \ldots\ldots\ldots\ldots + I_n]R_f$$

$$\therefore V_o = -\left[B_1 \frac{V_R}{2^1 R} + B_2 \frac{V_R}{2^2 R} + B_3 \frac{V_R}{2^3 R} + \ldots\ldots + B_n \frac{V_R}{2^n R}\right] R_f$$

$$V_o = -\frac{R_f}{R} V_R[B_1.2^{-1} + B_2.2^{-2} + B_3..2^{-3} + \ldots\ldots + B_n..2^{-n}]$$

$$If\, R_f = R$$

$$\therefore V_o = -V_R[B_1.2^{-1} + B_2.2^{-2} + B_3..2^{-3} + \ldots\ldots + B_n..2^{-n}]$$

Consider the example of 3-bit DAC.
When input binary sequence is B1 B2 B3 = 001

$$\therefore \quad V_o = -V_R[B_1.2^{-1} + B_2.2^{-2} + B_3.2^{-3}]$$

$$\therefore \quad V_o = -V_R[0.2^{-1} + 0.2^{-2} + 1.2^{-3}]$$

$$\therefore \quad V_o = -V_R[0 + 0 + \frac{1}{8}]$$

$$\therefore \quad V_o = -\frac{V_R}{8}$$

When input binary sequence is B1 B2 B3 = 101

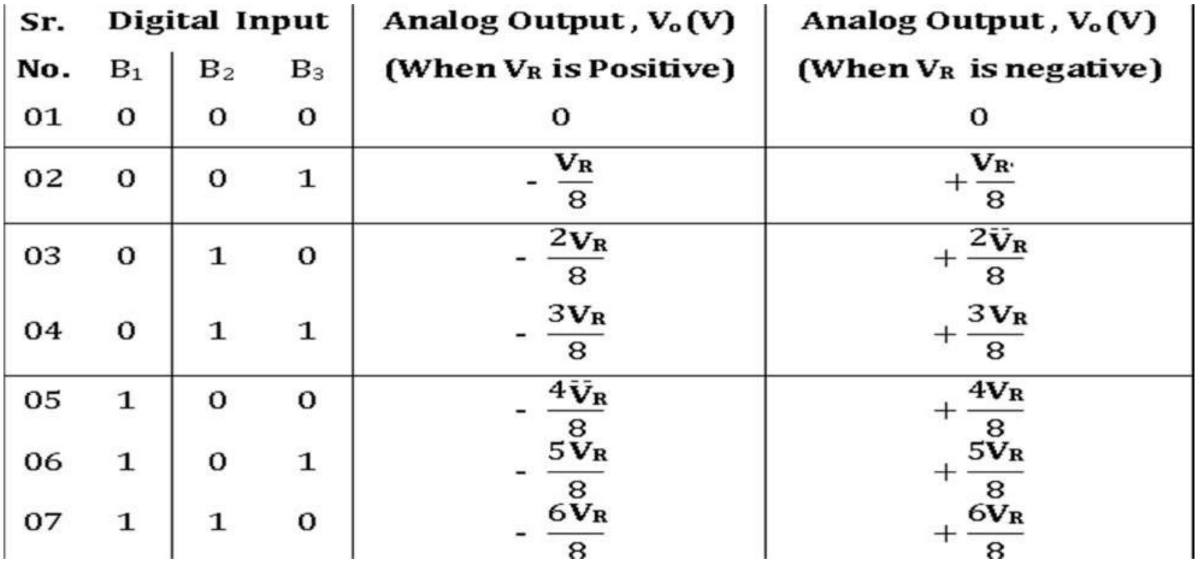
$$\therefore \quad V_o = -V_R[B_1.2^{-1} + B_2.2^{-2} + B_3.2^{-3}]$$

$$\therefore \quad V_o = -V_R[1.2^{-1} + 0.2^{-2} + 1.2^{-3}]$$

$$\therefore \quad V_o = -V_R[\frac{1}{2} + 0 + \frac{1}{8}]$$

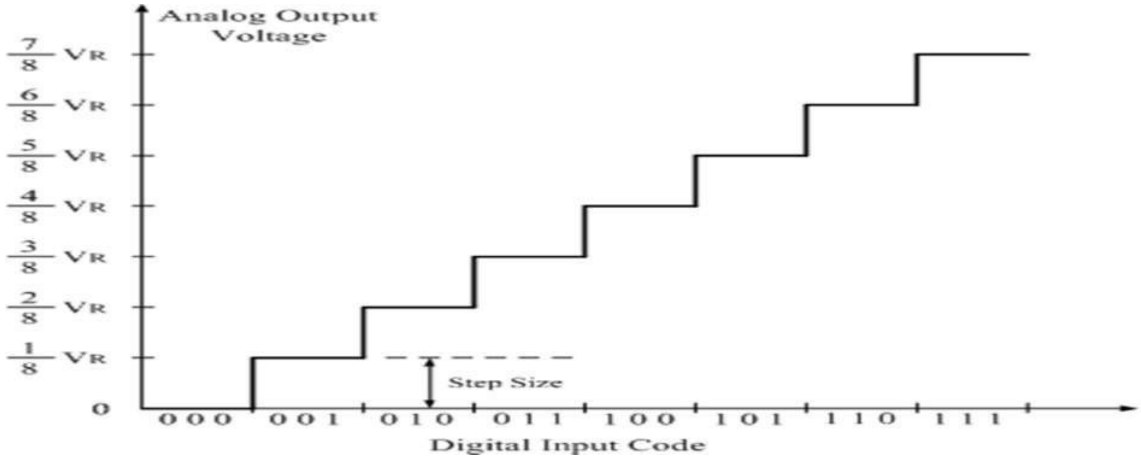
$$\therefore \quad V_o = -\frac{5V_R}{8}$$

If the reference voltage is positive i.e. + VR, then the output voltage is positive.
Similarly for other six combinations of digital input, the analog output voltage Vo is calculated as follows



Sr. No.	Digital Input B₁	B₂	B₃	Analog Output , V₀ (V) (When V_R is Positive)	Analog Output , V₀ (V) (When V_R is negative)
01	0	0	0	0	0
02	0	0	1	$-\frac{V_R}{8}$	$+\frac{V_R}{8}$
03	0	1	0	$-\frac{2V_R}{8}$	$+\frac{2V_R}{8}$
04	0	1	1	$-\frac{3V_R}{8}$	$+\frac{3V_R}{8}$
05	1	0	0	$-\frac{4V_R}{8}$	$+\frac{4V_R}{8}$
06	1	0	1	$-\frac{5V_R}{8}$	$+\frac{5V_R}{8}$
07	1	1	0	$-\frac{6V_R}{8}$	$+\frac{6V_R}{8}$

The following figure shows the staircase output voltage waveform obtained for R-2R ladder DAC (when VR is positive).



Disadvantages:

1) When number of binary input increases, it is not easy to maintain the resistance ratio.
2) Very wide ranges of different values of resistors are required.

For high accuracy of conversion, the values of resistances must be accurate.

3) Different current flows through resistors, so their wattage ratings are also different.
4) Accuracy and stability of conversion depends primarily on the absolute accuracy of the resistors and tracking of each other with temperature. eg. For 10 digit converter small resistance value = 10 kΩ and large resistance value = 5.12 MΩ

It is very difficult and expensive to obtain stable precise resistances of such value.

5) Since ‘R’ is very large, op-amp bias currents gives a drop which offsets output.
6) Resistances of switches may be comparable with smallest resistor.

Logic Family(TTL,CMOS,ECL)

Logic Families indicate the type of logic circuit used in the IC. The main types of logic families are:

- TTL(Transistor Transistor Logic)
- CMOS (Complementary MOS) □ ECL (Emitter Coupled Logic)

Characteristics of Logic Families

The main characteristics of Logic families include:

- Speed
- Fan-in
- Fan-out
- Noise Immunity
- Power Dissipation

Speed: Speed of a logic circuit is determined by the time between the application of input and change in the output of the circuit.

Fan-in: It determines the number of inputs the logic gate can handle.

Fan-out: Determines the number of circuits that a gate can drive.

Noise Immunity: Maximum noise that a circuit can withstand without affecting the output.

Power: When a circuit switches from one state to the other, power dissipates.

Transistor transistor logic (TTL)

Transistor tr ansistor logic (TTL) is a class of digital circuits built from bipolar junction transistors (BJTs) and resistors. It is called transistor transistor logic because transistors perform both the logic function (e.g., AND) and the amplifying function (compare with resistor transistor logic (RTL) and diode transistor logic (DTL)).

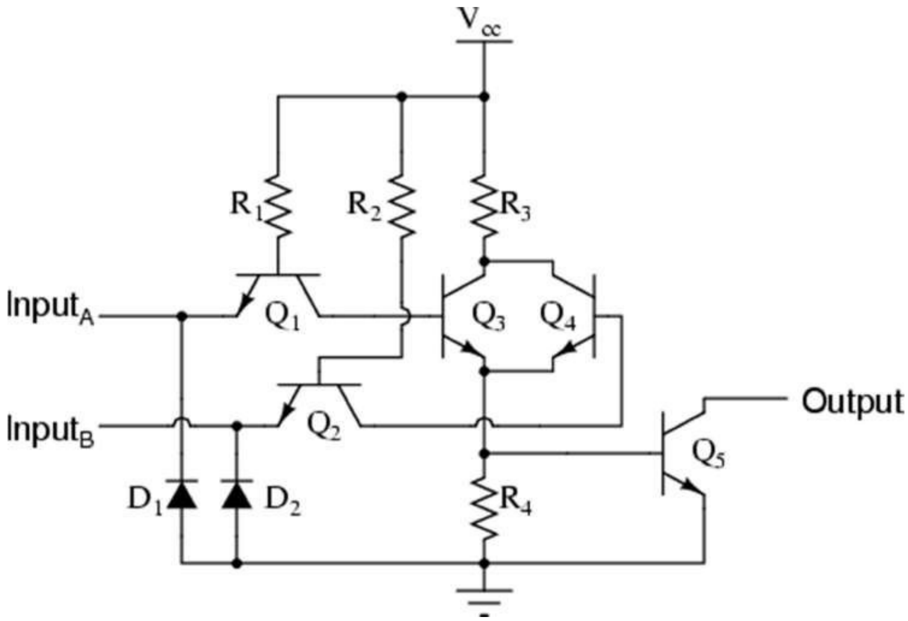
TTL integrated circuits (ICs) were widely used in applications such as computers, industrial controls, test equipment and instrumentation, consumer electronics, and synthesizers. The designation TTL is sometimes used to mean TTL-compatible logic levels, even when not associated directly with TTL integrated circuits, for example as a label on the inputs and outputs of electronic instruments~~.~~[1]

After their introduction in integrated circuit form in 1963 by Sylvania, TTL integrated circuits were manufactured by several semiconductor companies. The 7400 series (also called 74xx) by Texas Instruments became particularly popular. TTL manufacturers offered a wide range of logic gate, flip-flops, counters, and other circuits. Several variations of the original TTL circuit design were developed. The variations offered interchangeable functions that had higher speed or lower power dissipation to allow design optimization. TTL devices were originally made in ceramic and plastic dual-in-line (DIP) packages, and flat-pack form. TTL chips are now also made in surface-mount packages.

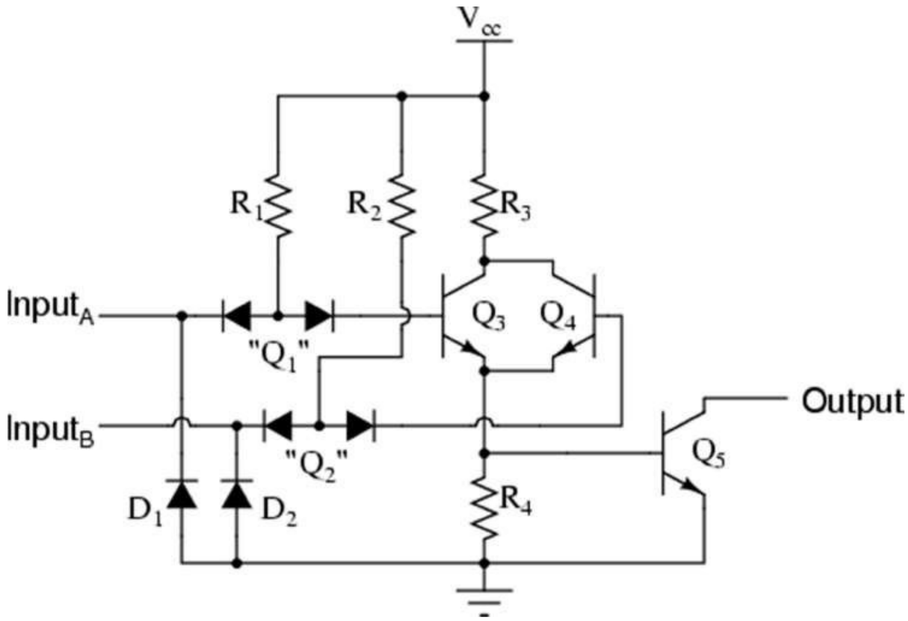
TTL became the foundation of computers and other digital electronics. Even after much larger scale integrated circuits made multiple-circuit-board processors obsolete, TTL devices still found extensive use as the glue logic interfacing more densely integrated components.

TTL NOR and OR gates

Let s examine the following TTL circuit and analyze its operation:



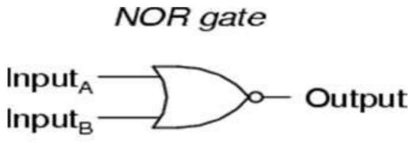
Transistors Q_1 and Q_2 are both arranged in the same manner that we ve seen for transistor Q_1 in all the other TTL circuits. Rather than functioning as amplifiers, Q_1 and Q_2 are both being used as two-diode steering networks. We may replace Q_1 and Q_2 with diode sets to help illustrate:



If input A is left floating (or connected to V_{cc}), current will go through the base of transistor Q_3, saturating it. If input A is grounded, that current is diverted away from Q_3 s base through the left steering diode of Q_1, thus forcing Q_3 into cutoff. The same can be said for input B and transistor Q_4: the logic level of input B determines Q_4 s conduction: either saturated or cutoff.

Notice how transistors Q_3 and Q_4 are paralleled at their collector and emitter terminals. In essence, these two transistors are acting as paralleled switches, allowing current through resistors R_3 and R_4 according to the logic levels of inputs A and B. If any input is at a high (1) level, then at least one of the two transistors (Q_3 and/or Q_4) will be saturated, allowing current through resistors R_3 and R_4, and turning on the final output transistor Q_5 for a low (0) logic level output. The only way the output of this circuit can ever assume a high (1) state isif both Q_3 and Q_4 are cutoff, which means both inputs would have to be grounded, or low (0).

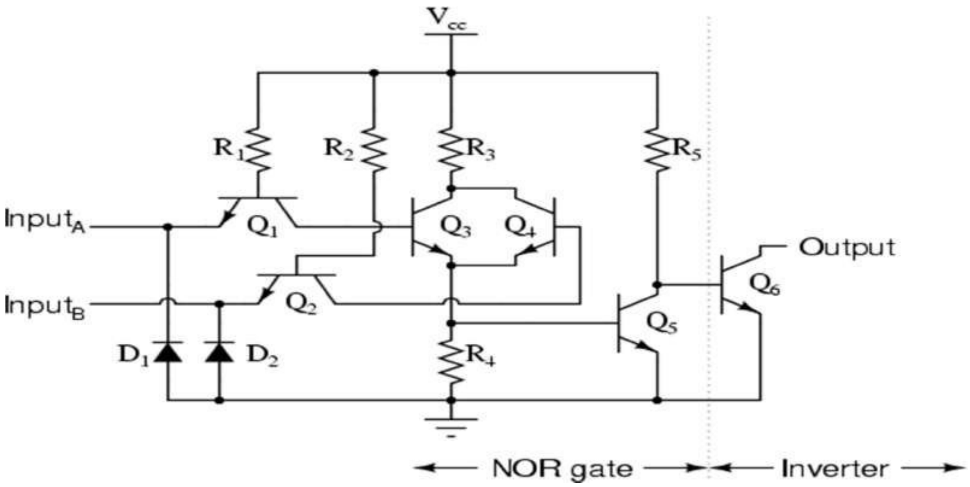
This circuit s truth table, then, is equivalent to that of the NOR gate:



A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

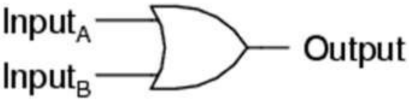
In order to turn this NOR gate circuit into an OR gate, we would have to invert the output logic level with another transistor stage, just like we did with the NAND-to-AND gate example:

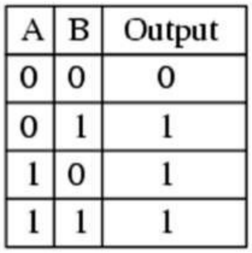
OR gate with open-collector output



The truth table and equivalent gate circuit (an inverted-output NOR gate) are shown here:

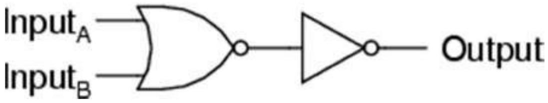
OR gate





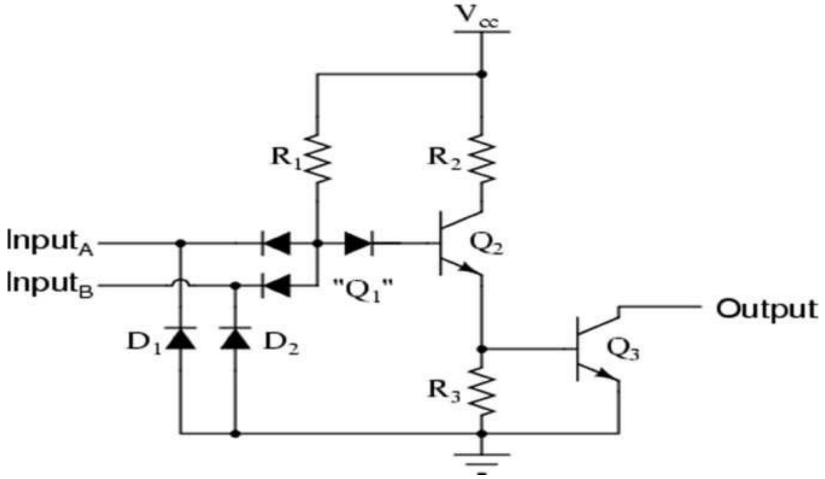
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

Equivalent circuit



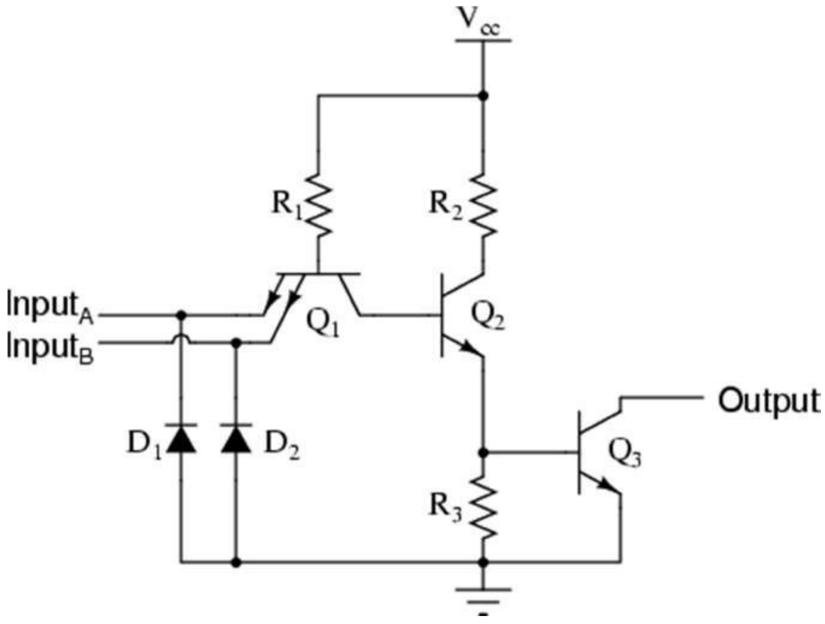
TTL NAND and AND gates

A two-input inverter circuit



This schematic illustrates a real circuit, but it isn t called a two-input inverter. Through analysis we will discover what this circuit s logic function is and correspondingly what it should be designated as.

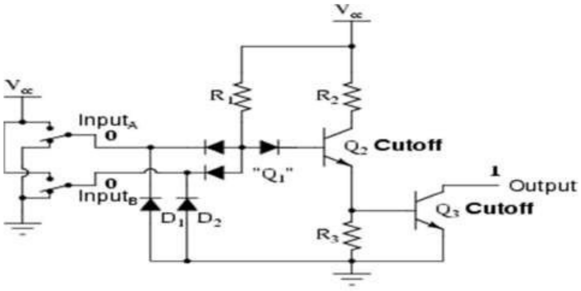
Just as in the case of the inverter and buffer, the steering diode cluster marked Q_1 is actually formed likea transistor, even though it isn t used in any amplifying capacity. Unfortunately, a simple NPN transistor structure is inadequate to simulate the three PN junctions necessary in this diode network, so a different transistor (and symbol) is needed. This transistor has one collector, one base, and two emitters, and in the circuit it looks like this:



In the single-input (inverter) circuit, grounding the input resulted in an output that assumed the high (1) state. In the case of the open-collector output configuration, this high state was simply floating. Allowing the

input to float (or be connected to V_{cc}) resulted in the output becoming grounded, which is the low or 0 state. Thus, a 1 in resulted in a 0 out, and vice versa.

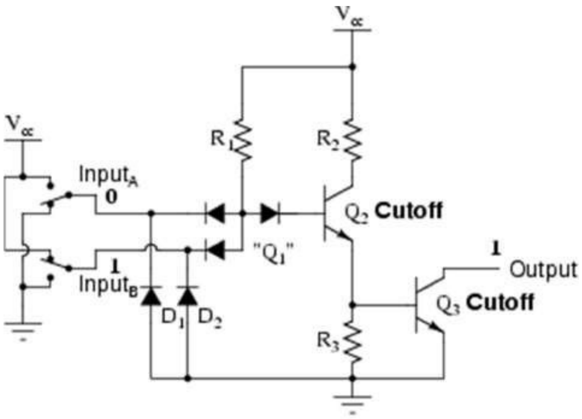
Since this circuit bears so much resemblance to the simple inverter circuit, the only difference being a second input terminal connected in the same way to the base of transistor Q_2, we can say that each of the inputs will have the same effect on the output. Namely, if either of the inputs are grounded, transistor Q_2 will be forced into a condition of cutoff, thus turning Q_3 off and floating the output (output goes high). The following series of illustrations shows this for three input states (00, 01, and 10):



$\text{Input}_A = 0$

$\text{Input}_B = 0$

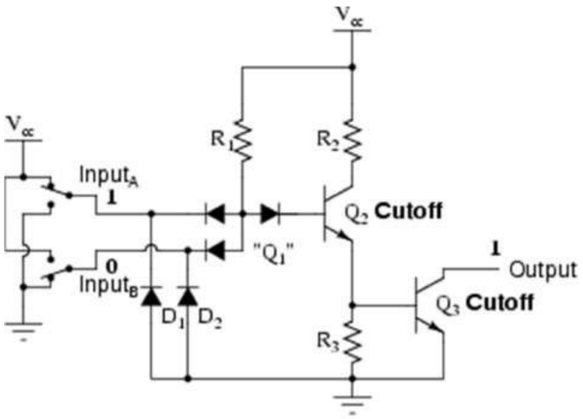
Output = 1



$\text{Input}_A = 0$

$\text{Input}_B = 1$

Output = 1

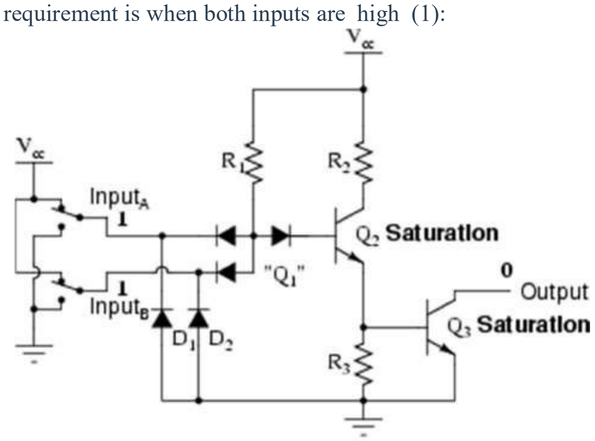


$Input_A$ = 1

$Input_B$ = 0

Output = 1

In any case where there is a grounded (low) input, the output is guaranteed to be floating (high). Conversely, the only time the output will ever go low is if transistor Q_3 turns on, which means transistor Q_2must be turned on (saturated), which means neither input can be diverting R_1 current away from the base of Q_2. The only condition that will satisfy this requirement is when both inputs are high (1):



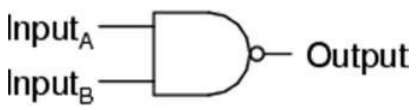
$Input_A$ = 1

$Input_B$ = 1

Output = 0

Collecting and tabulating these results into a truth table, we see that the pattern matches that of the NAND gate:

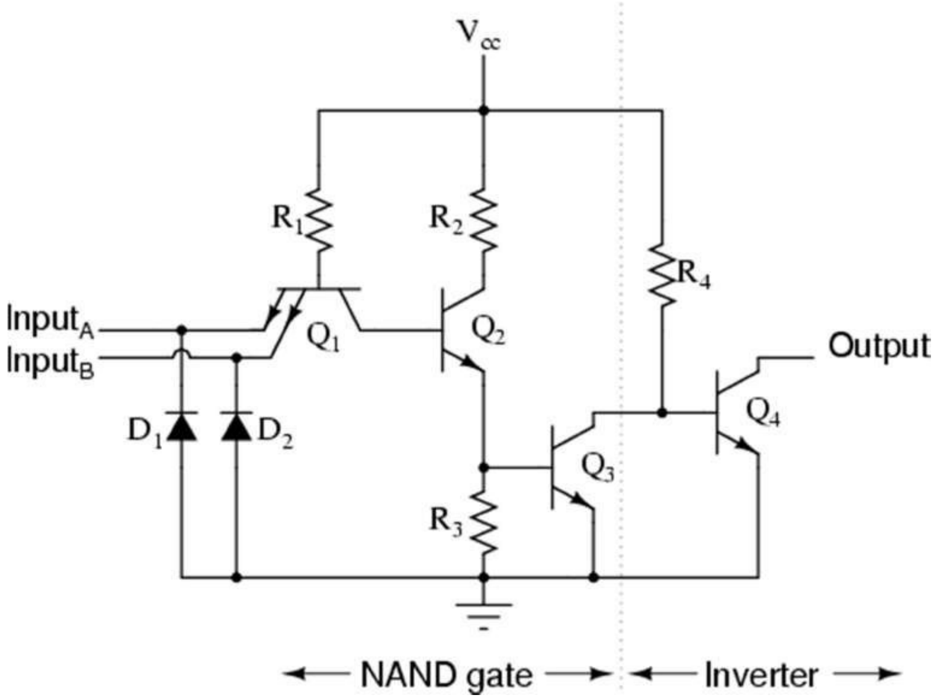
NAND gate



A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

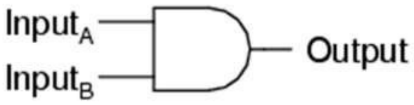
In the earlier section on NAND gates, this type of gate was created by taking an AND gate and increasing its complexity by adding an inverter (NOT gate) to the output. However, when we examine this circuit, we see that the NAND function is actually the simplest, most natural mode of operation for this TTL design. To create an AND function using TTL circuitry, we need to increase the complexity of this circuit by adding an inverter stage to the output, just like we had to add an additional transistor stage to the TTL inverter circuit to turn it into a buffer:

AND gate with open-collector output



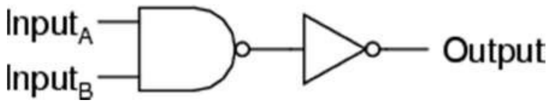
The truth table and equivalent gate circuit (an inverted-output NAND gate) are shown here:

AND gate



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Equivalent circuit

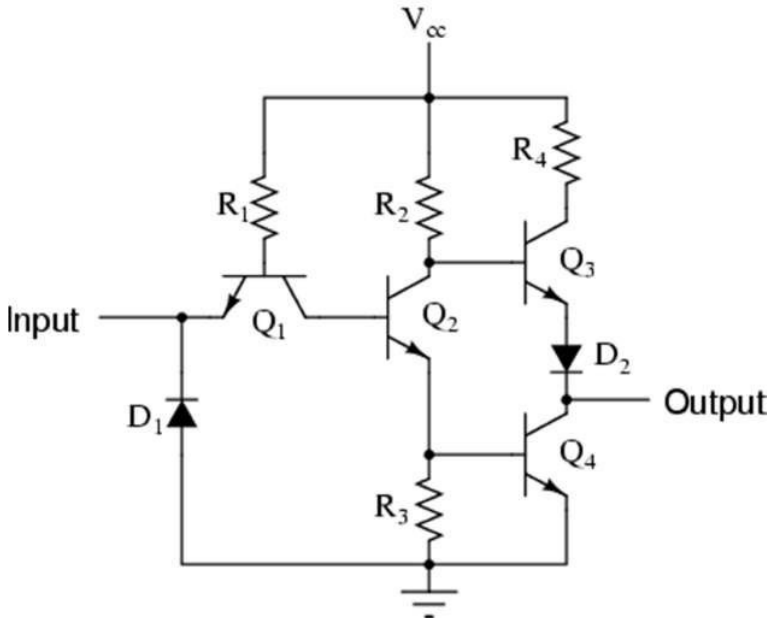


The NOT Gate

The single-transistor inverter circuit illustrated earlier is actually too crude to be of practical use as a gate. Real inverter circuits contain more than one transistor to maximize voltage gain (so as to ensure that the final output transistor is either in full cutoff or full saturation), and other components designed to reduce the chance of accidental damage.

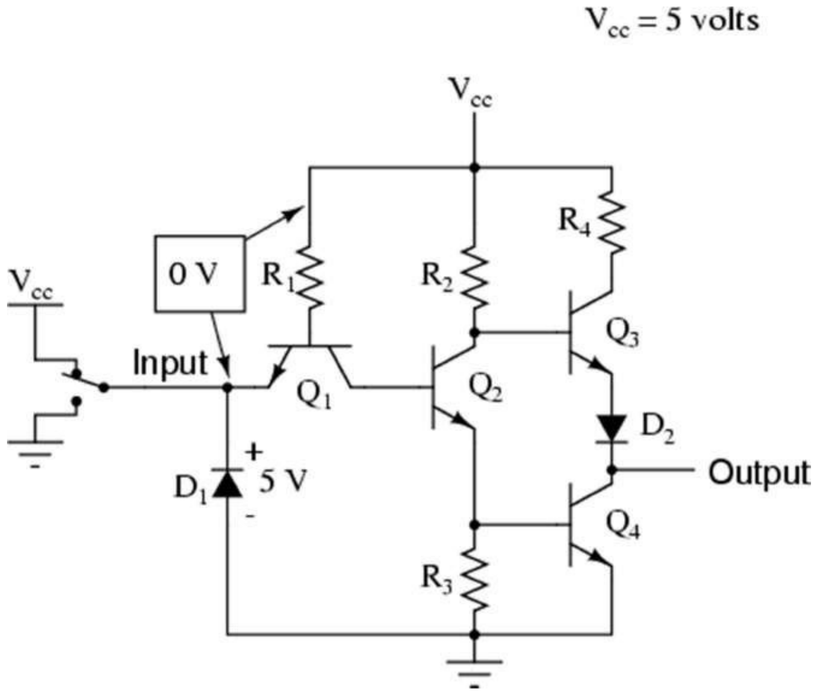
Shown here is a schematic diagram for a real inverter circuit, complete with all necessary components for efficient and reliable operation:

Practical inverter (NOT) circuit

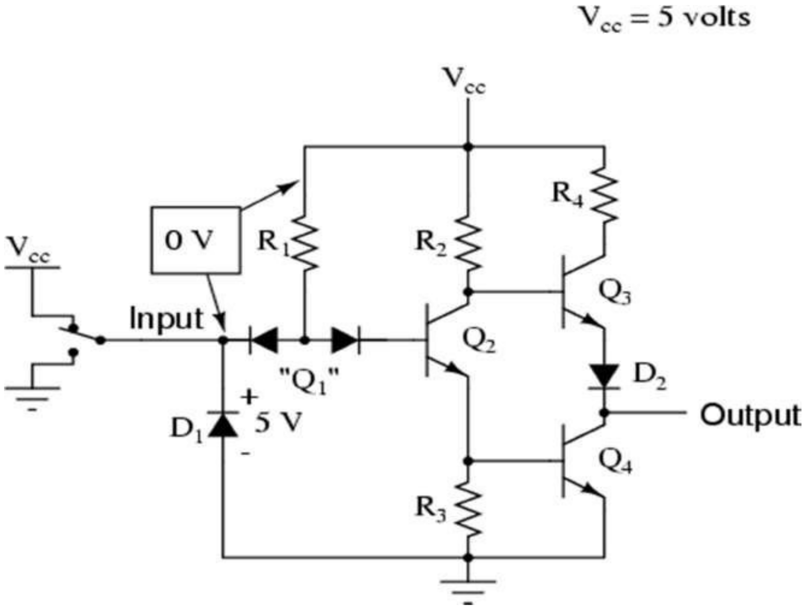


This circuit is composed exclusively of resistors, diodes and bipolar transistors. Bear in mind that other circuit designs are capable of performing the NOT gate function, including designs substituting field-effect transistors for bipolar (discussed later in this chapter).

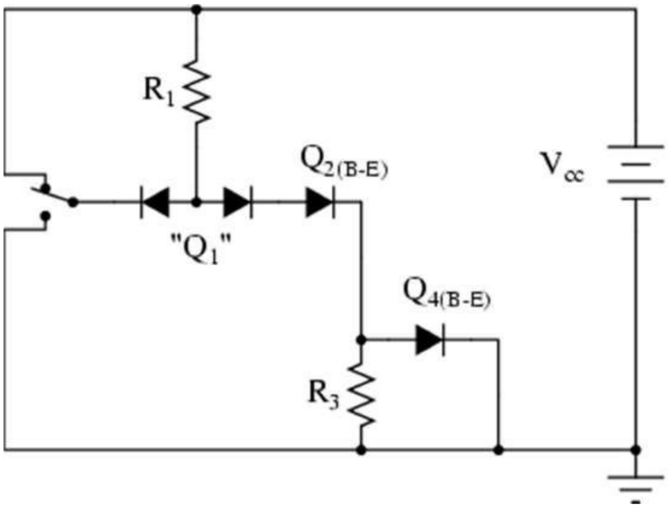
Let s analyze this circuit for the condition where the input is high, or in a binary 1 state. We can simulate this by showing the input terminal connected to V_{cc} through a switch:



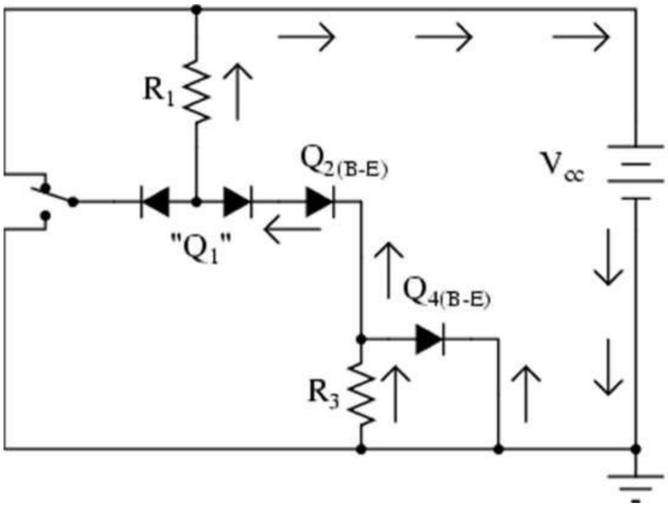
In this case, diode D$_1$ will be reverse-biased, and therefore not conduct any current. In fact, the only purpose for having D$_1$ in the circuit is to prevent transistor damage in the case of a negative voltage being impressed on the input (a voltage that is negative, rather than positive, with respect to ground). With no voltage between the base and emitter of transistor Q$_1$, we would expect no current through it, either. However, as strange as it may seem, transistor Q$_1$ is not being used as is customary for a transistor. In reality, Q$_1$ is being used in this circuit as nothing more than a back-to-back pair of diodes. The following schematic shows the real function of Q$_1$:



The purpose of these diodes is to steer current to or away from the base of transistor Q_2, depending on the logic level of the input. Exactly how these two diodes are able to steer current isn t exactly obvious at first inspection, so a short example may be necessary for understanding. Suppose we had the following diode/resistor circuit, representing the base-emitter junctions of transistors Q_2 and Q_4 as single diodes, stripping away all other portions of the circuit so that we can concentrate on the current steered through the two back-to-back diodes:

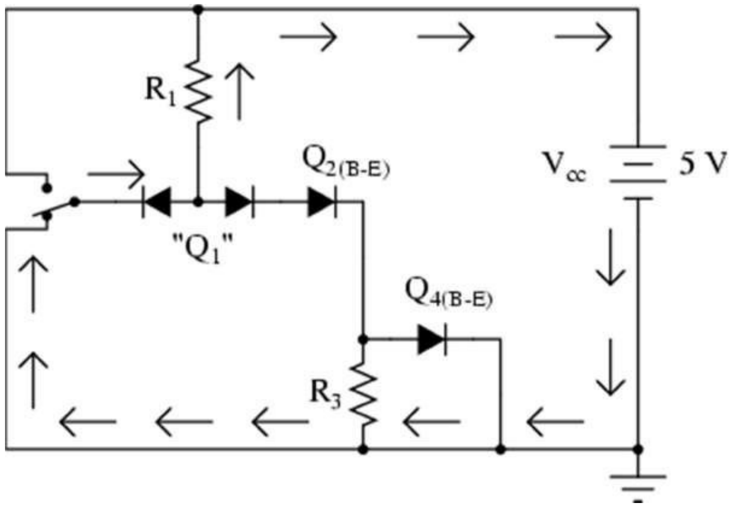


With the input switch in the up position (connected to V_{cc}), it should be obvious that there will be no current through the left steering diode of Q_1, because there isn t any voltage in the switch-diode-R_1-switch loop to motivate electrons to flow. However, there will be current through the right steering diode of Q_1, as well as through Q_2 s base-emitter diode junction and Q_4 s base-emitter diode junction:



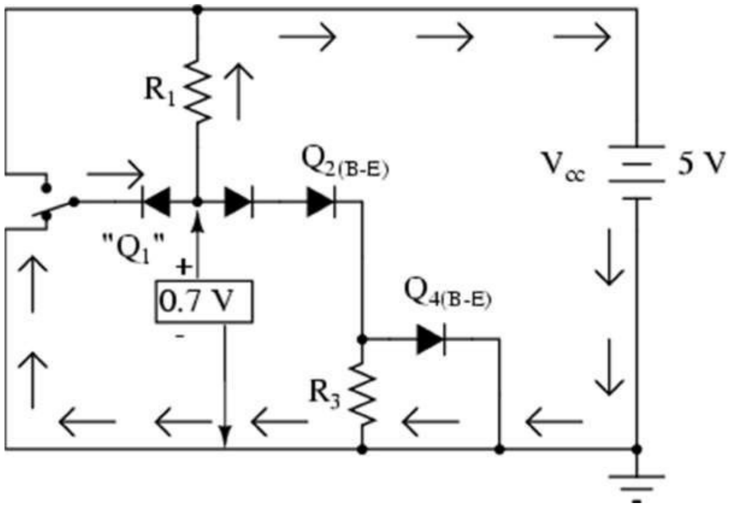
This tells us that in the real gate circuit, transistors Q_2 and Q_4 will have base current, which will turn them on to conduct collector current. The total voltage dropped between the base of Q_1 (the node joining the two back-to- back steering diodes) and ground will be about 2.1 volts, equal to the combined voltage drops of three PN junctions: the right steering diode, Q_2 s base-emitter diode, and Q_4 s base-emitter diode.

Now, let s move the input switch to the down position and see what happens:



If we were to measure current in this circuit, we would find that all of the current goes through the left steering diode of Q_1 and none of it through the right diode. Why is this? It still appears as though there is a complete path for current through Q_4 s diode, Q_2 s diode, the right diode of the pair, and R_1, so why will there be no current through that path?

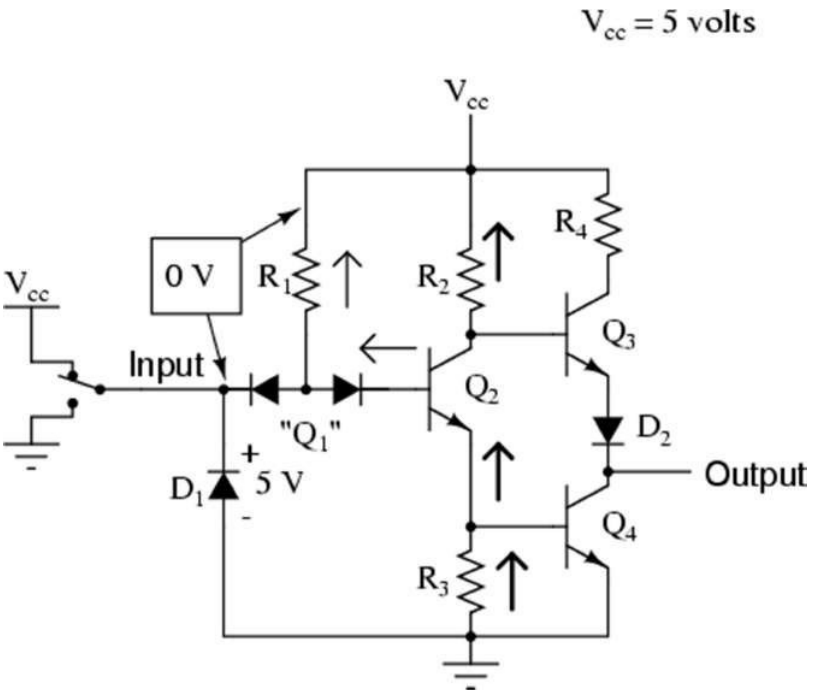
Remember that PN junction diodes are very nonlinear devices: they do not even begin to conduct current until the forward voltage applied across them reaches a certain minimum quantity, approximately 0.7 volts for silicon and 0.3 volts for germanium. And then when they begin to conduct current, they will not drop substantially more than 0.7 volts. When the switch in this circuit is in the down position, the left diode of the steering diode pair is fully conducting, and so it drops about 0.7 volts across it and no more.



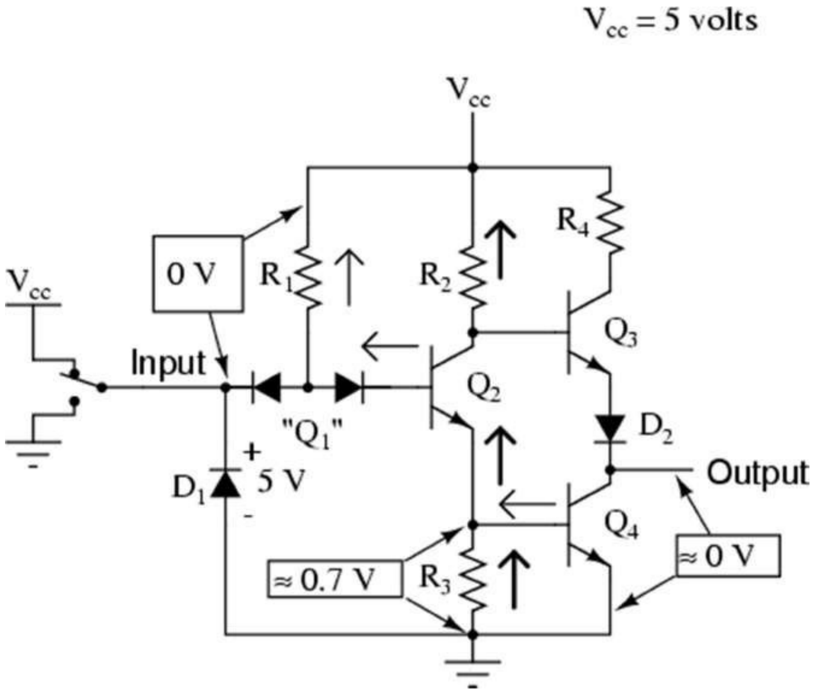
Recall that with the switch in the up position (transistors Q_2 and Q_4 conducting), there was about 2.1 volts dropped between those same two points (Q_1 s base and ground), which also happens to be the minimum voltage necessary to forward-bias three series-connected silicon PN junctions into a state of conduction. The 0.7 volts provided by the left diode s forward voltage drop is simply insufficient to allow any electron flow through the series string of the right diode, Q_2 s diode, and the R_3//Q_4 diode parallel subcircuit, and so no electrons flow through that path. With no current through the bases of either transistor Q_2 or Q_4, neither one will be able to conduct collector current: transistors Q_2 and Q_4 will both be in a state of cutoff.

Consequently, this circuit configuration allows 100 percent switching of Q_2 base current (and therefore control over the rest of the gate circuit, including voltage at the output) by diversion of current through the left steering diode.

In the case of our example gate circuit, the input is held high by the switch (connected to V_{cc}), making the left steering diode (zero voltage dropped across it). However, the right steering diode is conducting current through the base of Q_2, through resistor R_1:

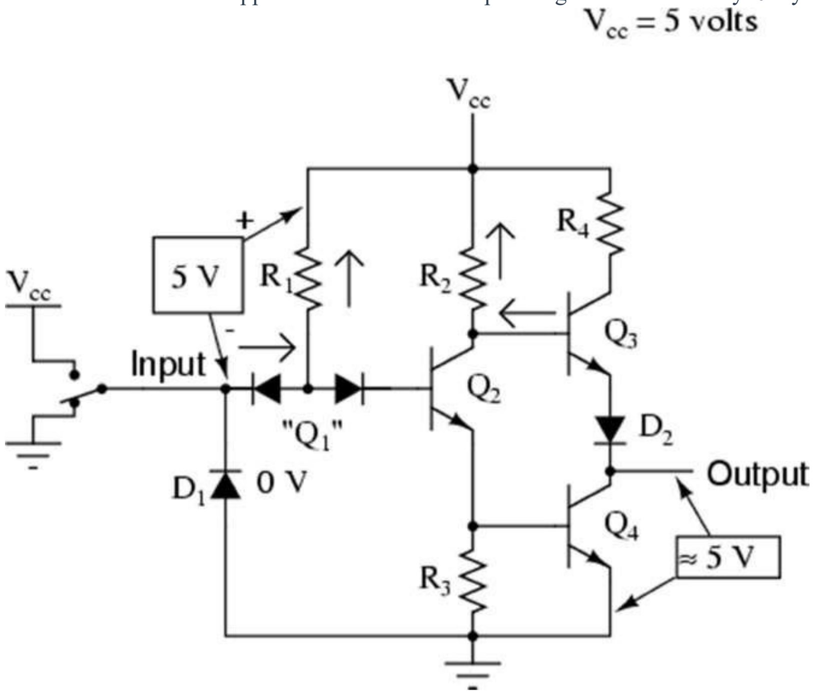


With base current provided, transistor Q_2 will be turned on. More specifically, it will be saturated by virtue of the more-than-adequate current allowed by R_1 through the base. With Q_2 saturated, resistor R_3 will be dropping enough voltage to forward-bias the base-emitter junction of transistor Q_4, thus saturating it as well:



With Q_4 saturated, the output terminal will be almost directly shorted to ground, leaving the output terminal at a voltage (in reference to ground) of almost 0 volts, or a binary 0 (low) logic level. Due to the presence of diode D_2, there will not be enough voltage between the base of Q_3 and its emitter to turn it on, so it remains in cutoff.

Let s see now what happens if we reverse the input s logic level to a binary 0 by actuating the input switch:



Now there will be current through the left steering diode of Q_1 and no current through the right steering diode. This eliminates current through the base of Q_2, thus turning it off. With Q_2 off, there is no longer a path for Q_4 base current, so Q_4 goes into cutoff as well. Q_3, on the other hand, now has sufficient voltage dropped between its base and ground to forward-bias its base-emitter junction and saturate it, thus raising the output terminal voltage to a high state. In actuality, the output voltage will be somewhere around 4 volts depending on the degree of saturation and any load current, but still high enough to be considered a high (1) logic level.

With this, our simulation of the inverter circuit is complete: a 1 in gives a 0 out, and vice versa.

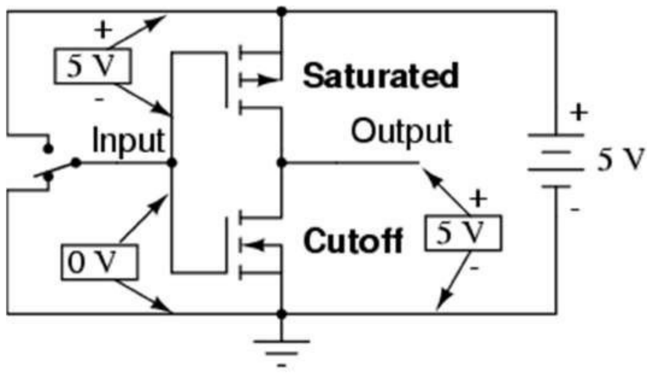
The astute observer will note that this inverter circuit s input will assume a high state of left floating (not connected to either V_{cc} or ground). With the input terminal left unconnected, there will be no current through the left steering diode of Q_1, leaving all of R_1 s current to go through Q_2 s base, thus saturating Q_2 and driving the circuit output to a low state:

CMOS Gate Circuitry

Field-effect transistors, particularly the insulated-gate variety, may be used in the design of gate circuits. Being voltage-controlled rather than current-controlled devices, IGFETs tend to allow very simple circuit designs. Take for instance, the following inverter circuit built using P- and N-channel IGFETs:

Notice the V_{dd} label on the positive power supply terminal. This label follows the same convention as V_{cc} in TTL circuits: it stands for the constant voltage applied to the drain of a field effect transistor, in reference to ground.

Let s connect this gate circuit to a power source and input switch, and examine its operation. Please note that these IGFET transistors are E-type (Enhancement-mode), and so are normally-off devices. It takes an applied voltage between gate and drain (actually, between gate and substrate) of the correct polarity to bias them on.



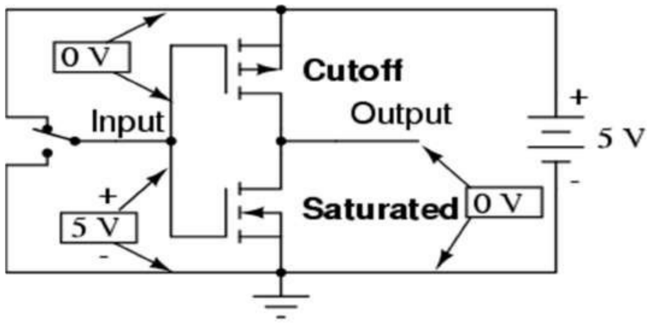
Input = "low" (0)

Output = "high" (1)

The upper transistor is a P-channel IGFET. When the channel (substrate) is made more positive than the gate (gate negative in reference to the substrate), the channel is enhanced and current is allowed between source and drain. So, in the above illustration, the top transistor is turned on.

The lower transistor, having zero voltage between gate and substrate (source), is in its normal mode: off. Thus, the action of these two transistors are such that the output terminal of the gate circuit has a solid connection to V_{dd} and a very high resistance connection to ground. This makes the output high (1) for the low (0) state of the input.

Next, we ll move the input switch to its other position and see what happens:



Input = "high" (1)

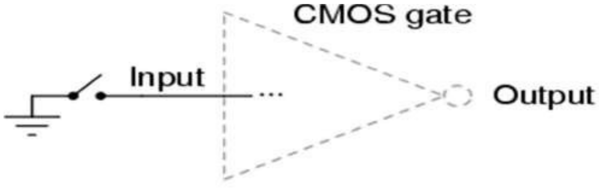
Output = "low" (0)

Now the lower transistor (N-channel) is saturated because it has sufficient voltage of the correct polarity applied between gate and substrate (channel) to turn it on (positive on gate, negative on the channel). The upper transistor, having zero voltage applied between its gate and substrate, is in its normal mode: off. Thus, the output of this gate circuit is now low (0). Clearly, this circuit exhibits the behavior of an inverter, or NOT gate.

Using field-effect transistors instead of bipolar transistors has greatly simplified the design of the inverter gate. Note that the output of this gate never floats as is the case with the simplest TTL circuit: it has a natural totem- pole configuration, capable of both sourcing and sinking load current. Key to this gate circuit s elegant design is the complementary use of both P- and N-channel IGFETs. Since IGFETs are more commonly known as MOSFETs (Metal-Oxide-Semiconductor Field Effect Transistor), and this circuit uses both P- and N-channel transistors together, the general classification given to gate circuits like this one is CMOS: Complementary Metal Oxide Semiconductor.

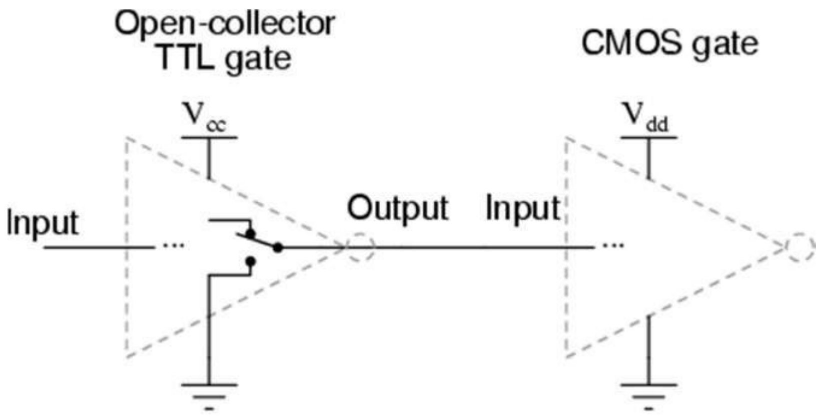
CMOS circuits aren t plagued by the inherent nonlinearities of the field-effect transistors, because as digital circuits their transistors always operate in either the saturated or cutoff modes and never in the active mode. Their inputs are, however, sensitive to high voltages generated by electrostatic (static electricity) sources, and may even be activated into high (1) or low (0) states by spurious voltage sources if left floating. For this reason, it is inadvisable to allow a CMOS logic gate input to float under any circumstances. Please note that this is very different from the behavior of a TTL gate where a floating input was safely interpreted as a high (1) logic level.

This may cause a problem if the input to a CMOS logic gate is driven by a single-throw switch, where one state has the input solidly connected to either V_{dd} or ground and the other state has the input floating (not connected to anything):



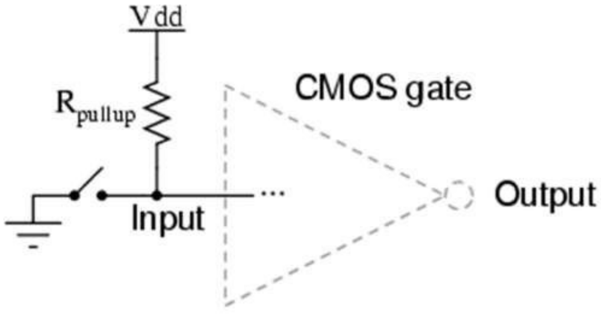
When switch is closed, the gate sees a definite "low" (0) input. However, when switch is open, the input logic level will be uncertain because it's floating.

Also, this problem arises if a CMOS gate input is being driven by an open-collector TTL gate. Because such a TTL gate s output floats when it goes high (1), the CMOS gate input will be left in an uncertain state:



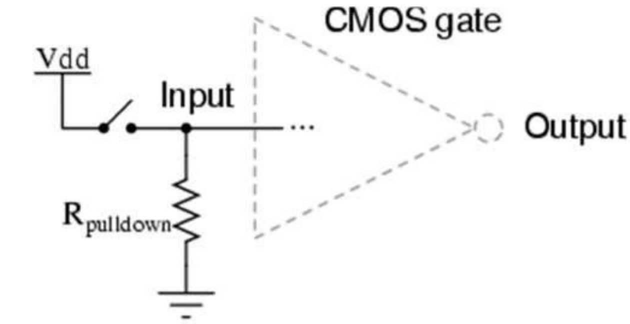
When the open-collector TTL gate's output is "high" (1), the CMOS gate's input will be left floating and in an uncertain logic state.

Fortunately, there is an easy solution to this dilemma, one that is used frequently in CMOS logic circuitry. Whenever a single-throw switch (or any other sort of gate output incapable of both sourcing and sinking current) is being used to drive a CMOS input, a resistor connected to either V_{dd} or ground may be used to provide a stable logic level for the state in which the driving device's output is floating. This resistor's value is not critical: 10 kΩ is usually sufficient. When used to provide a "high" (1) logic level in the event of a floating signal source, this resistor is known as a pullup resistor:



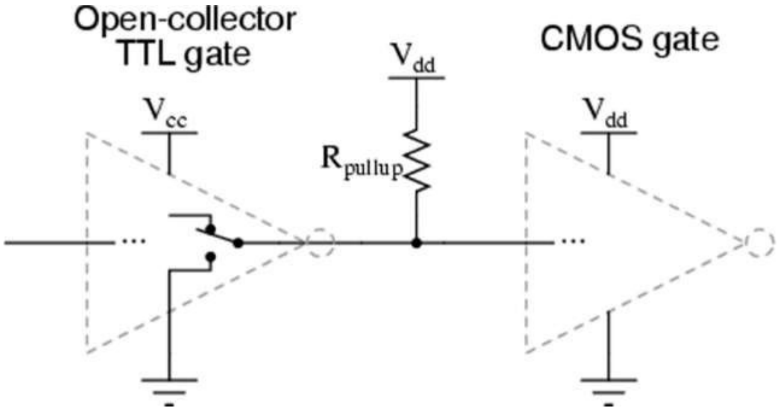
When switch is closed, the gate sees a definite "low" (0) input. When the switch is open, R_{pullup} will provide the connection to Vdd needed to secure a reliable "high" logic level for the CMOS gate input.

When such a resistor is used to provide a "low" (0) logic level in the event of a floating signal source, it is known as a pulldown resistor. Again, the value for a pulldown resistor is not critical:



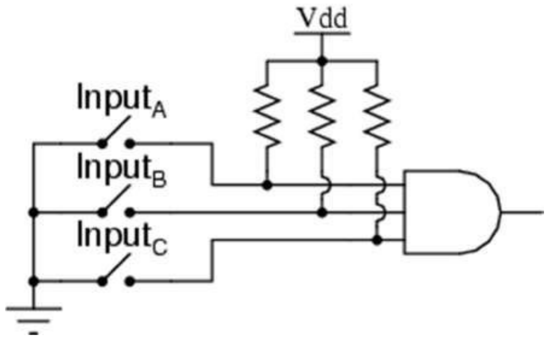
When switch is closed, the gate sees a definite "high" (1) input. When the switch is open, $R_{pulldown}$ will provide the connection to ground needed to secure a reliable "low" logic level for the CMOS gate input.

Because open-collector TTL outputs always sink, never source, current, pullup resistors are necessary when interfacing such an output to a CMOS gate input:



Although the CMOS gates used in the preceding examples were all inverters (single-input), the same principle of pullup and pulldown resistors applies to multiple-input CMOS gates. Of course, a separate pullup or pulldown resistor will be required for each gate input:

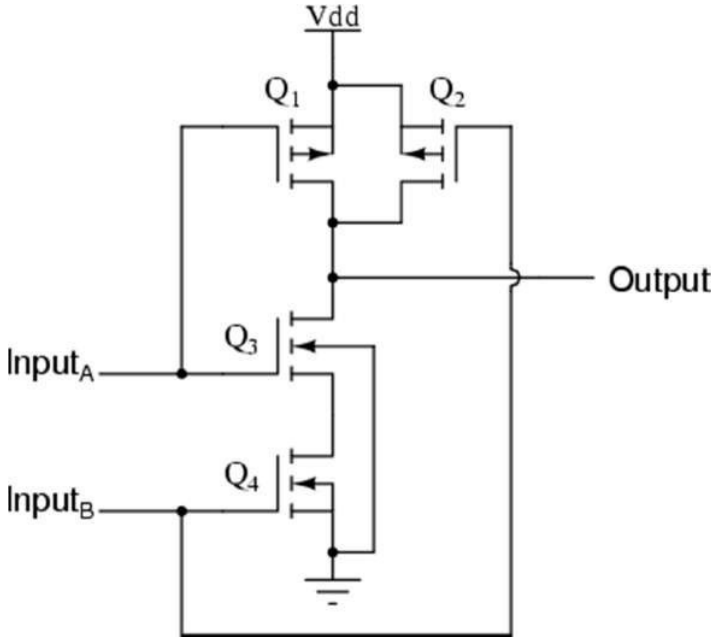
Pullup resistors for a 3-input CMOS AND gate



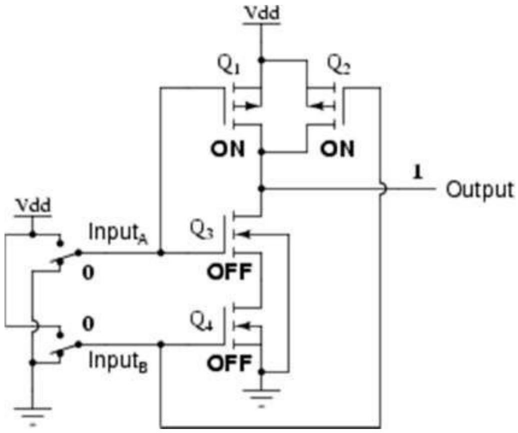
This brings us to the next question: how do we design multiple-input CMOS gates such as AND, NAND, OR, and NOR? Not surprisingly, the answer(s) to this question reveal a simplicity of design much like that of the CMOS inverter over its TTL equivalent.

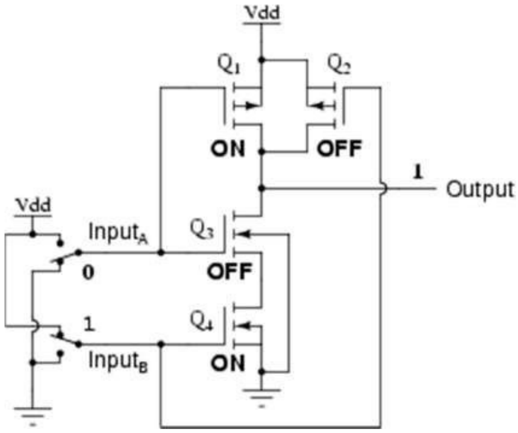
For example, here is the schematic diagram for a CMOS NAND gate:

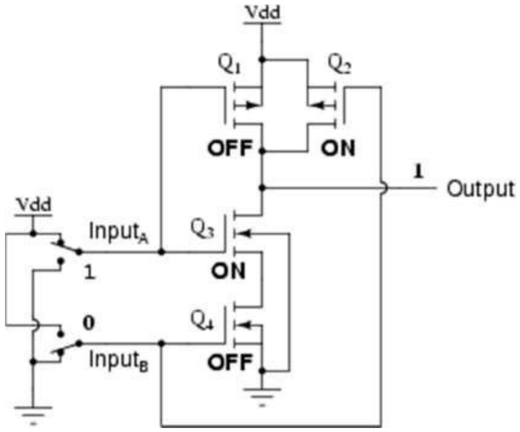
CMOS NAND gate

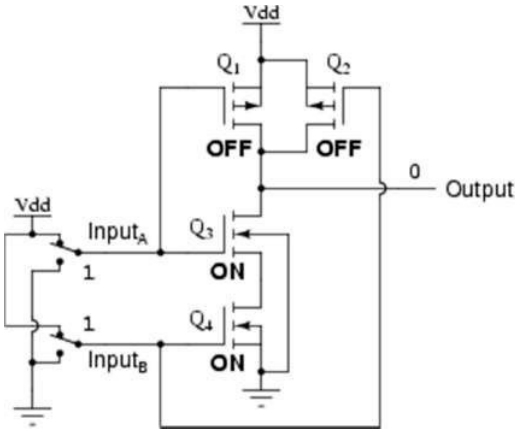


Notice how transistors Q_1 and Q_3 resemble the series-connected complementary pair from the inverter circuit. Both are controlled by the same input signal (input A), the upper transistor turning off and the lower transistor turning on when the input is high (1), and vice versa. Notice also how transistors Q_2 and Q_4 are similarly controlled by the same input signal (input B), and how they will also exhibit the same on/off behavior for the same input logic levels. The upper transistors of both pairs (Q_1 and Q_2) have their source and drain terminals paralleled, while the lower transistors (Q_3 and Q_4) are series-connected. What this means is that the output will go high (1) if either top transistor saturates, and will go low (0) only if both lower transistors saturate. The following sequence of illustrations shows the behavior of this NAND gate for all four possibilities of input logic levels (00, 01, 10, and 11):



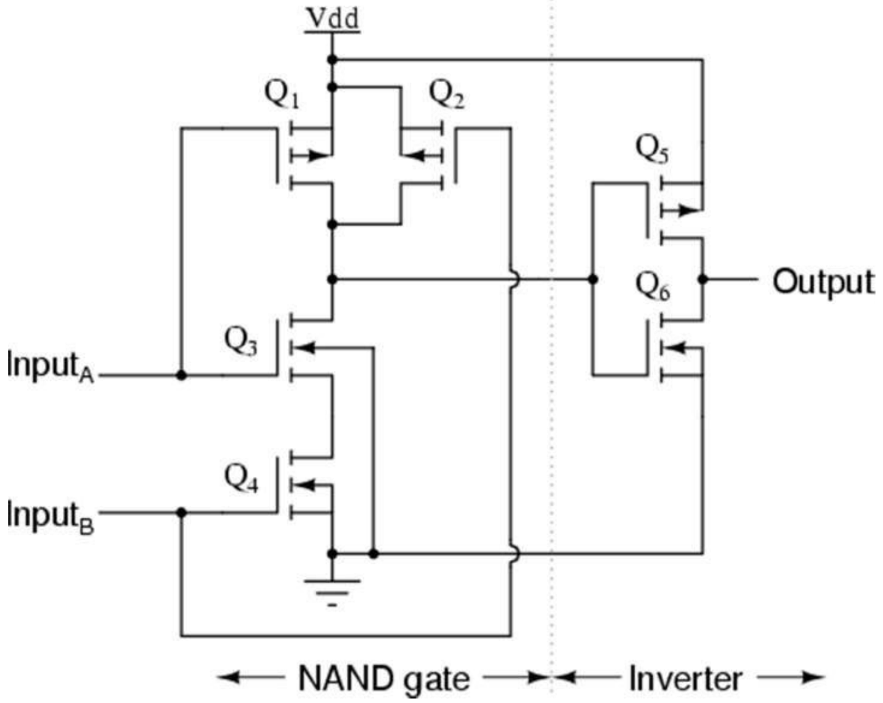




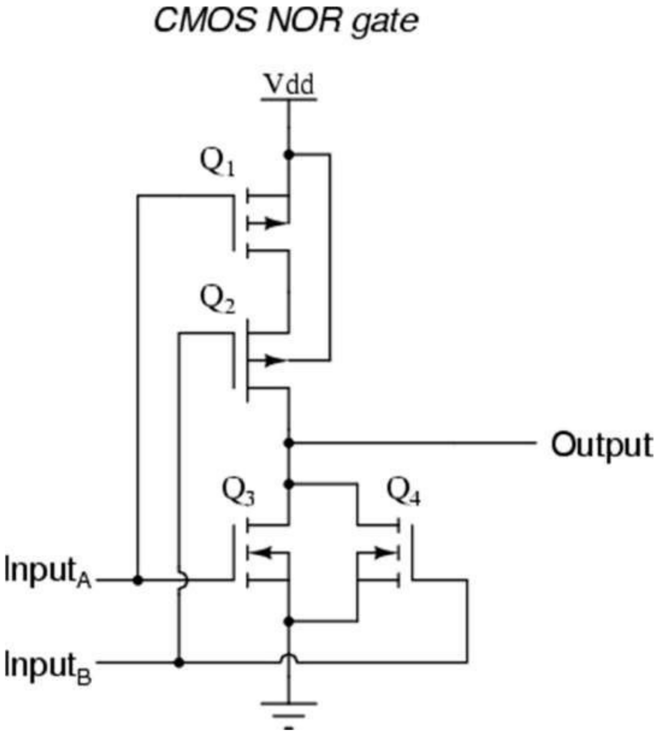


As with the TTL NAND gate, the CMOS NAND gate circuit may be used as the starting point for the creation of an AND gate. All that needs to be added is another stage of transistors to invert the output signal:

CMOS AND gate

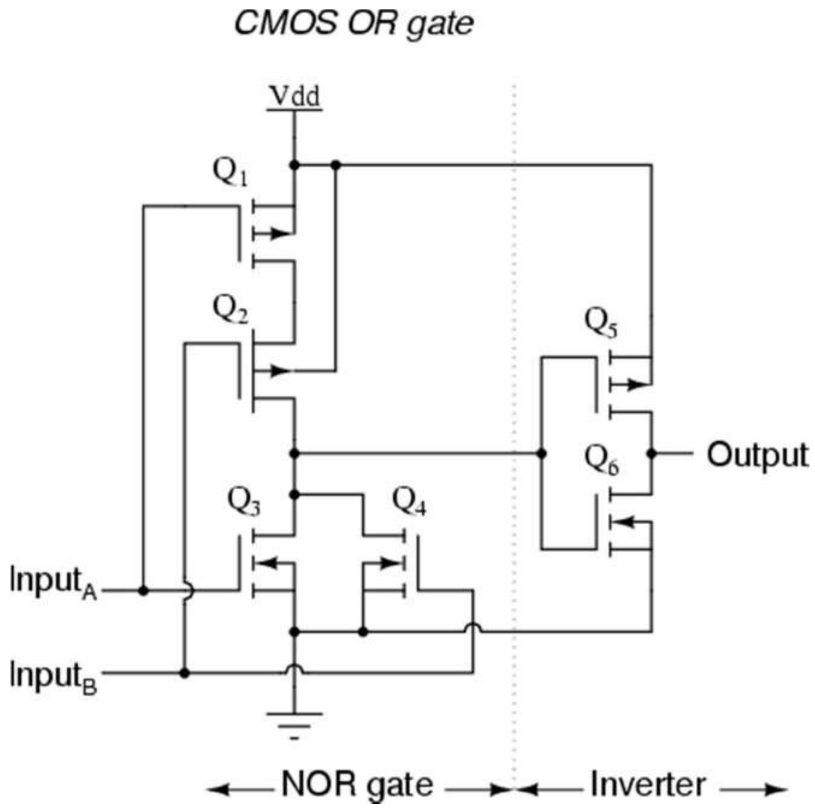


A CMOS NOR gate circuit uses four MOSFETs just like the NAND gate, except that its transistors are differently arranged. Instead of two paralleled sourcing (upper) transistors connected to V_{dd} and two series- connected sinking (lower) transistors connected to ground, the NOR gate uses two series-connected sourcing transistors and two parallel-connected sinking transistors like this:



As with the NAND gate, transistors Q_1 and Q_3 work as a complementary pair, as do transistors Q_2 and Q_4. Each pair is controlled by a single input signal. If either input A or input B are high (1), at least one of the lower transistors (Q_3 or Q_4) will be saturated, thus making the output low (0). Only in the event of both inputs being low (0) will both lower transistors be in cutoff mode and both upper transistors be saturated, the conditions necessary for the output to go high (1). This behavior, of course, defines the NOR logic function.

The OR function may be built up from the basic NOR gate with the addition of an inverter stage on the output:



Since it appears that any gate possible to construct using TTL technology can be duplicated in CMOS, why do these two families of logic design still coexist? The answer is that both TTL and CMOS have their own unique advantages.

First and foremost on the list of comparisons between TTL and CMOS is the issue of power consumption. In this measure of performance, CMOS is the unchallenged victor. Because the complementary P- and N-channel MOSFET pairs of a CMOS gate circuit are (ideally) never conducting at the same time, there is little or no current drawn by the circuit from the V_{dd} power supply except for what current is necessary to source current to a load. TTL, on the other hand, cannot function without some current drawn at all times, due to the biasing requirements of the bipolar transistors from which it is made.

There is a caveat to this advantage, though. While the power dissipation of a TTL gate remains rather constant regardless of its operating state(s), a CMOS gate dissipates more power as the frequency of its input signal(s) rises. If a CMOS gate is operated in a static (unchanging) condition, it dissipates zero power (ideally). However, CMOS gate circuits draw transient current during every output state switch from low to high and vice versa. So, the more often a CMOS gate switches modes, the more often it will draw current from the V_{dd} supply, hence greater power dissipation at greater frequencies.

A CMOS gate also draws much less current from a driving gate output than a TTL gate because MOSFETs are voltage-controlled, not current-controlled, devices. This means that one gate can drive many more CMOS inputs than TTL inputs. The measure of how many gate inputs a single gate output can drive is called fanout.

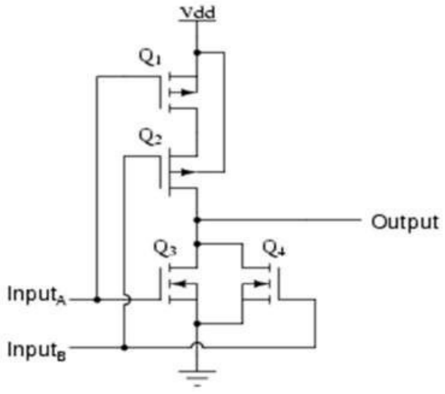
Another advantage that CMOS gate designs enjoy over TTL is a much wider allowable range of power supply voltages. Whereas TTL gates are restricted to power supply (V_{cc}) voltages between 4.75 and 5.25 volts, CMOS gates are typically able to operate on any voltage between 3 and 15 volts! The reason behind this disparity in power supply voltages is the respective bias requirements of MOSFET versus bipolar junction transistors.

MOSFETs are controlled exclusively by gate voltage (with respect to substrate), whereas BJTs are current- controlled devices. TTL gate circuit resistances are precisely calculated for proper bias currents assuming a 5 volt regulated power supply. Any significant variations in that power supply voltage will result in the transistor bias currents being incorrect, which then results in unreliable (unpredictable) operation. The only effect that variations in power supply voltage have on a CMOS gate is the voltage definition of a high (1) state. For a CMOS gate operating at 15 volts of power supply voltage (V_{dd}), an input signal must be close to 15 volts in order to be considered high (1). The voltage threshold for a low (0) signal remains the same: near 0 volts.

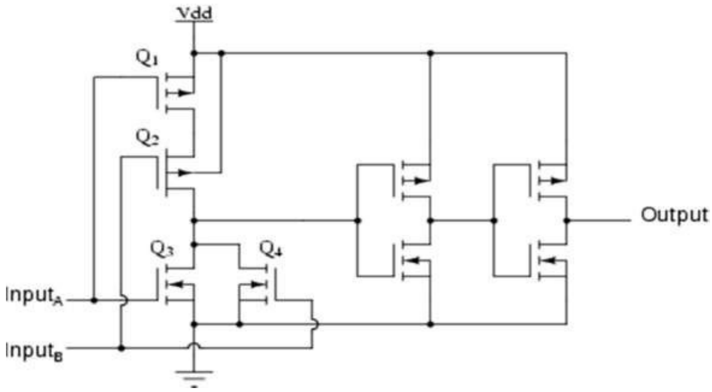
One decided disadvantage of CMOS is slow speed, as compared to TTL. The input capacitances of a CMOS gate are much, much greater than that of a comparable TTL gate owing to the use of MOSFETs rather than BJTs and so a CMOS gate will be slower to respond to a signal transition (low-to-high or vice versa) than a TTL gate, all other factors being equal. The RC time constant formed by circuit resistances and the input capacitance of the gate tend to impede the fast rise- and fall-times of a digital logic level, thereby degrading high-frequency performance.

A strategy for minimizing this inherent disadvantage of CMOS gate circuitry is to buffer the output signal with additional transistor stages, to increase the overall voltage gain of the device. This provides a faster- transitioning output voltage (high-to-low or low-to-high) for an input voltage slowly changing from one logic state to another. Consider this example, of an unbuffered NOR gate versus a buffered, or B-series, NORgate:

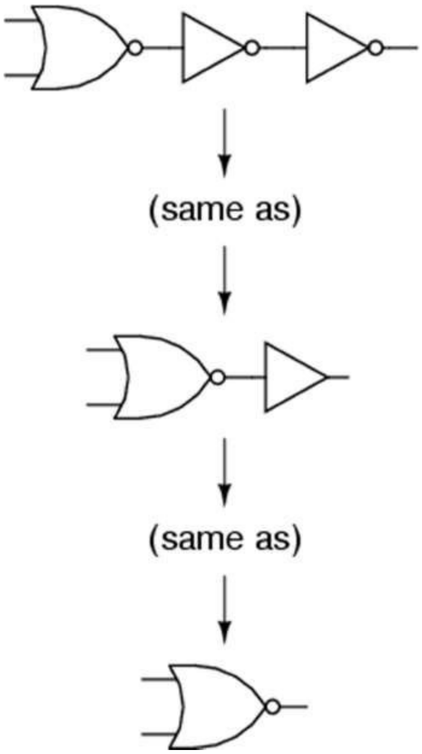
"Unbuffered" NOR gate



"B-series" (buffered) NOR gate



In essence, the B-series design enhancement adds two inverters to the output of a simple NOR circuit. This serves no purpose as far as digital logic is concerned, since two cascaded inverters simply cancel:



However, adding these inverter stages to the circuit does serve the purpose of increasing overall voltage gain, making the output more sensitive to changes in input state, working to overcome the inherent slowness caused by CMOS gate input capacitance.

- REVIEW:
- CMOS logic gates are made of IGFET (MOSFET) transistors rather than bipolar junction transistors.
- CMOS gate inputs are sensitive to static electricity. They may be damaged by high voltages, and they may assume any logic level if left floating.
- Pullup and pulldown resistors are used to prevent a CMOS gate input from floating if being driven by a signal source capable only of sourcing or sinking current.
- CMOS gates dissipate far less power than equivalent TTL gates, but their power dissipation increases with signal frequency, whereas the power dissipation of a TTL gate is approximately constant over a wide range of operating conditions.
- CMOS gate inputs draw far less current than TTL inputs, because MOSFETs are voltage-controlled, not current-controlled, devices.
- CMOS gates are able to operate on a much wider range of power supply voltages than TTL: typically 3 to 15 volts versus 4.75 to 5.25 volts for TTL.
- CMOS gates tend to have a much lower maximum operating frequency than TTL gates due to input capacitances caused by the MOSFET gates.
- B-series CMOS gates have buffered outputs to increase voltage gain from input to output, resulting in faster output response to input signal changes. This helps overcome the inherent slowness of CMOS gates due to MOSFET input capacitance and the RC time constant thereby engendered.

ECL: Emitter-Coupled Logic

The key to reducing propagation delay in a bipolar logic family is to prevent a gate s transistors from saturating. Section BJT.3 shows how Schottky diodes can prevent saturation in TTL gates. However, it is also possible to prevent saturation by using a radically different circuit structure, called current-mode logic (CML) or emitter- coupled logic (ECL).

Unlike the other logic families in this chapter, ECL does not produce a large voltage swing between the LOW and HIGH levels. Instead, it has a small voltage swing, less than a volt, and it internally switches current between two possible paths, depending on the output state.

The first ECL logic family was introduced by General Electric in 1961. The concept was later refined by Motorola and others to produce the still popular 10K and 100K ECL families. These families are extremely fast, offering propagation delays as short as 1 ns. The newest ECL family, ECLinPS (literally, ECL in picoseconds), offers maximum delays under 0.5 ns (500 ps), including the signal delay getting on and off of the IC package. Throughout the evolution of digital circuit technology, some type of ECL has always been the fastest technology for discrete, packaged logic components.

Still, commercial ECL families aren t nearly as popular as CMOS and TTL, mainly because they consume much more power. In fact, high power consumption made the design of ECL supercomputers, such as the Cray-1 and Cray-2, as much of a challenge in cooling technology as in digital design. Also, ECL has a poor speed- power product, does not provide a high level of integration, has fast edge rates requiring design for transmission-line effects in most applications, and is not directly compatible with TTL and CMOS. Nevertheless, ECL still finds its place as a logic and interface technology in very high-speed communications gear, including fiber-optic transceiver interfaces for gigabit Ethernet and Asynchronous Transfer Mode (ATM) networks.

ECL.1 Basic ECL Circuit

The basic idea of current-mode logic is illustrated by the inverter/buffer circuit in Figure ECL-1 on the next page. This circuit has both an inverting output (OUT1) and a noninverting output (OUT2). Two transistors are connected as a differential amplifier with a common emitter resistor. The supply voltages for this example are VCC = 5.0, VBB = 4.0, and VEE = 0 V, and the input LOW and HIGH levels are defined to be 3.6 and 4.4 V. This circuit actually produces output LOW and HIGH levels that are 0.6 V higher (4.2 and 5.0 V), but this is corrected in real ECL circuits.

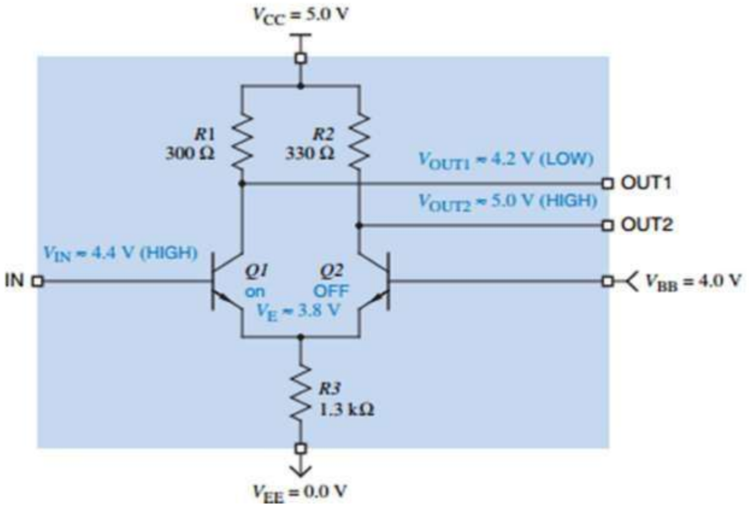


Figure ECL-1
Basic ECL inverter/ buffer circuit with input HIGH.

When VIN is HIGH, as shown in the figure, transistor Q1 is on, but not saturated, and transistor Q2 is OFF. This is true because of a careful choice of resistor values and voltage levels. Thus, VOUT2 is pulled to 5.0 V (HIGH) through R2, and it can be shown that the voltage drop across R1 is about 0.8 V, so that VOUT1 is about 4.2 V (LOW).

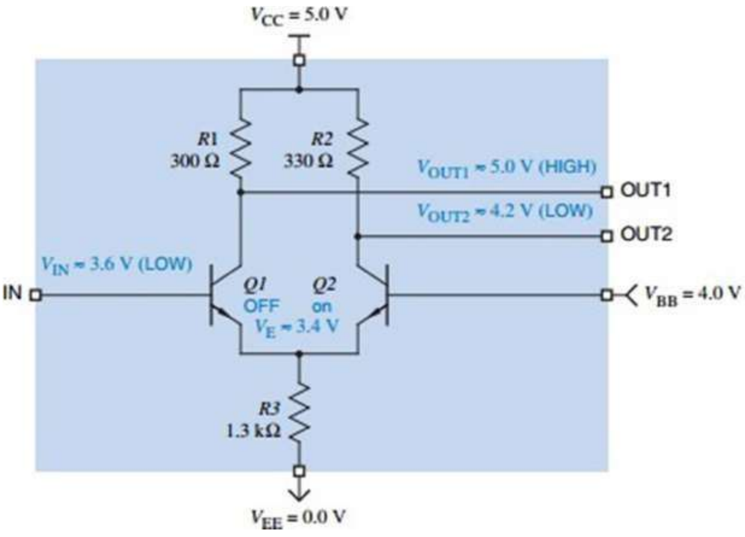


Figure ECL-2
Basic ECL inverter/ buffer circuit with input LOW.

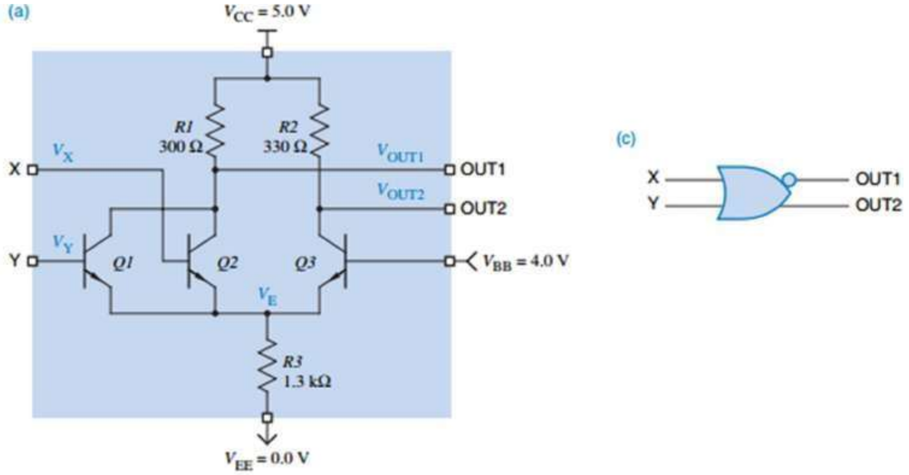
When VIN is LOW, as shown in Figure ECL-2, transistor Q2 is on, but not saturated, and transistor Q1 is OFF. Thus, VOUT1 is pulled to 5.0 V through R1, and it can be shown that VOUT2 is about 4.2 V.

The outputs of this inverter are called differential outputs because they are always complementary, and it is possible to determine the output state by looking at the difference between the output voltages (VOUT1 − VOUT2) rather than their absolute values. That is, the output is 1 if (VOUT1 − VOUT2) > 0, and it is 0 if (VOUT1 − VOUT2) < 0. It is possible to build input circuits with two wires per logical input that define the logical signal value in this way; these are called differential inputs.

Differential signals are used in most ECL interfacing and clock distribution applications because of their low skew and high noise immunity. They are low skew because the timing of a 0-to-1 or 1-to-0 transition does not depend critically on voltage thresholds, which may change with temperature or between devices. Instead, the timing depends only on when the voltages cross over relative to each other. Similarly, the relative definition of 0 and 1 provides outstanding noise immunity, since noise created by variations in the power supply or coupled from external sources tends to be a common-mode signal that affect both differential signals similarly, leaving the difference value unchanged.

It is also possible, of course, to determine the logic value by sensing the absolute voltage level of one input signal, called a single-ended input. Singleended signals are used in most ECL logic applications to avoid the obvious expense of doubling the number of signal lines. The basic ECL inverter in Figure ECL-2 has a single-ended input. It always has both outputs available internally; the circuit is actually either an inverter or a noninverting buffer, depending on whether we use OUT1 or OUT2.
To perform logic with the basic circuit of Figure ECL-2, we simply place additional transistors in parallel with Q1, similar to the approach in a TTL NOR gate. For example, Figure ECL-3 on the next page shows a 2-input ECL OR/ NOR gate. If any input is HIGH, the corresponding input transistor is active, and VOUT1 is LOW (NOR output). At the same time, Q3 is OFF, and VOUT2 is HIGH (OR output).

Recall that the input levels for the inverter/buffer are defined to be 3.6 and 4.4 V, while the output levels that it produces are 4.2 and 5.0 V. This is obviously a problem. We could put a diode in series with each output to lower it by 0.6 V to match the input levels, but that still leaves another problem the outputs have poor fanout. A HIGH output must supply base current to the inputs that it drives, and this current creates an additional voltage drop across R1 or R2, reducing the output voltage (and we don t have much margin to work with). These problems are solved in commercial ECL families, such as the 10K family described next.



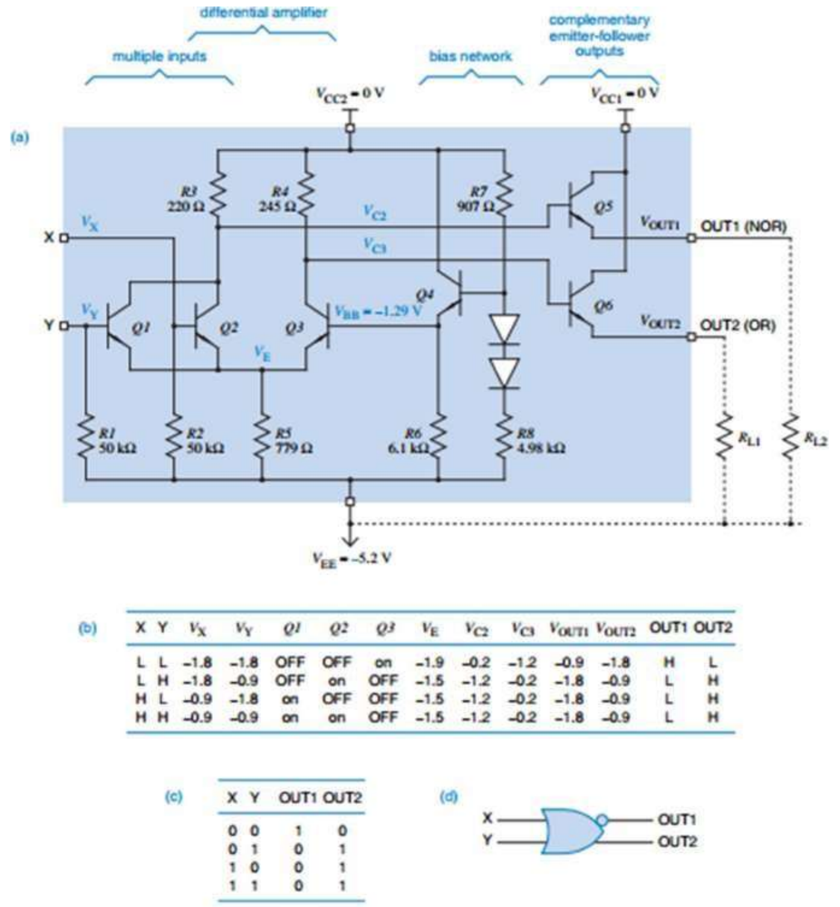
(b)

X	Y	V_X	V_Y	Q1	Q2	Q3	V_E	V_{OUT1}	V_{OUT2}	OUT1	OUT2
L	L	3.6	3.6	OFF	OFF	on	3.4	5.0	4.2	H	L
L	H	3.6	4.4	OFF	on	OFF	3.8	4.2	5.0	L	H
H	L	4.4	3.6	on	OFF	OFF	3.8	4.2	5.0	L	H
H	H	4.4	4.4	on	on	OFF	3.8	4.2	5.0	L	H

(d)

X	Y	OUT1	OUT2
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	1

Figure ECL-3 ECL 2-input OR/NOR gate: (a) circuit diagram; (b) function table; (c) logic symbol; (d) truth table.



X	Y	V_X	V_Y	Q1	Q2	Q3	V_E	V_{C2}	V_{C3}	V_{OUT1}	V_{OUT2}	OUT1	OUT2
L	L	−1.8	−1.8	OFF	OFF	on	−1.9	−0.2	−1.2	−0.9	−1.8	H	L
L	H	−1.8	−0.9	OFF	on	OFF	−1.5	−1.2	−0.2	−1.8	−0.9	L	H
H	L	−0.9	−1.8	on	OFF	OFF	−1.5	−1.2	−0.2	−1.8	−0.9	L	H
H	H	−0.9	−0.9	on	on	OFF	−1.5	−1.2	−0.2	−1.8	−0.9	L	H

X	Y	OUT1	OUT2
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	1

Figure ECL-5 Two-input 10K ECL OR/NOR gate: (a) circuit diagram; (b) function table; (c) truth table; (d) logic symbol.

Figure ECL-5(a) is the circuit for an ECL OR/NOR gate, one section of a quad OR/NOR gate with part number 10102. A pull-down resistor on each input ensures that if the input is left unconnected, it is treated as LOW. The bias network has component values selected to generate VBB = −1.29 V for proper operation of the differential amplifier. Each output transistor, using the emitterfollower configuration, maintains its emitter voltage at one diode- drop below its base voltage, thereby achieving the required output-level shift. Figure ECL-5(b) summarizes the electrical operation of the gate. The emitter-follower outputs used in ECL 10K require external pull-down resistors, as shown in the figure. The 10K family is designed to use external rather than internal pull-down resistors for good reason. The rise and fall times of ECL output transitions are so fast (typically 2 ns) that any connection longer than a few inches must be treated as a transmission line and must be terminated as discussed in Section Zo. Rather than waste power with an internal pull-down resistor, ECL 10K allows the designer to select an external resistor that satisfies both pull-down and transmission-line termination requirements. The simplest termination, sufficient for short connections, is to connect a resistor in the range of 270 Ω to 2 kΩ from each output to VEE. A typical ECL 10K gate has a propagation delay of 2 ns, comparable to 74AS TTL. With its outputs left unconnected, a 10K gate consumes about 26 mW of power, also comparable to a 74AS TTL gate, which consumes about 20 mW. However, the termination required by ECL 10K also consumes power, from 10 to 150 mW per output depending on the type of termination circuit. A 74AS TTL output may or may not require a power-consuming termination circuit, depending on the physical characteristics of the application.