Digital Electronics

Unit -1

Introduction to Number Systems

Binary to decimal conversion

The decimal equivalent of the binary number $(1001.0101)_2$ is determined as follows:

- The integer part = 1001
- The decimal equivalent = 1 × 20 + 0 × 21 + 0 × 22 + 1 × 23 = 1 + 0 + 0 + 8 = 9 • The fractional part = .0101
- Therefore, the decimal equivalent = 0 × 2−1 + 1 × 2−2 + 0 × 2−3 + 1 × 2−4 = 0 + 0.25 + 0 + 0.0625 = 0.3125
- Therefore, the decimal equivalent of $(1001.0101)_2$ = 9.3125

Decimal to Binary conversion

We will find the binary equivalent of $(13.375)_{10}$. Solution

- The integer part = 13 Divisor Dividend Remainder 2 13 — 26 1 23 0 21 1 —0 1
- The binary equivalent of (13)10 is therefore $(1101)_2$
- The fractional part = .375
- 0.375 × 2 = 0.75 with a carry of 0
- 0.75 × 2 = 0.5 with a carry of 1
- 0.5 × 2 = 0 with a carry of 1
- The binary equivalent of $(0.375)_{10}$ = $(.011)_2$
- Therefore, the binary equivalent of $(13.375)_{10}$ = $(1101.011)_2$

Octal Numbers

In the octal number system, we have the 7's and 8's complements. The 7's complement of a given octal number is obtained by subtracting each octal digit from 7. For example, the 7's complement of (562)8 would be (215)8. The

8's complement is obtained by adding '1' to the 7's complement. The 8's complement of (562)8 would be (216)8.

Decimal-to-Octal Conversion The process of decimal-to-octal conversion is similar to that of decimal-to-binary conversion. The progressive division in the case of the integer part and the progressive multiplication while working on the fractional part here are by '8' which is the radix of the octal number system. Again, the integer and fractional parts of the decimal number are treated separately. The process can be best illustrated with the help of an example.

Example 1.4 We will find the octal equivalent of $(73.75)_{10}$

Solution

• The integer part = 73

Divisor	Dividend	Remainder
8	73	—
8	9	1
8	1	1
—	0	1

• The octal equivalent of $(73)_{10} = (111)_8$

• The fractional part = 0.75

• 0.75 × 8 = 0 with a carry of 6

• The octal equivalent of $(0.75)_{10} = (.6)_8$

• Therefore, the octal equivalent of $(73.75)_{10} = (111.6)_8$

Octal-to-Decimal Conversion

The decimal equivalent of the octal number $(137.21)_8$ is determined as follows:

• The integer part = 137

• The decimal equivalent = 7 × 80 + 3 × 81 + 1 × 82 = 7 + 24 + 64 = 95 Number Systems 7

• The fractional part = .21

• The decimal equivalent = 2 × 8−1 + 1 × 8−2 = 0.265

• Therefore, the decimal equivalent of $(137.21)_8 = (95.265)_{10}$

Binary–Octal and Octal–Binary Conversions

An octal number can be converted into its binary equivalent by replacing each octal digit with its three-bit binary equivalent. We take the three-bit equivalent because the base of the octal number system is 8 and it is the third power of the base of the binary number system, i.e., 2. All we have then to remember is the three-bit binary equivalents of the basic digits of the octal number system. A binary number can be converted into an equivalent octal number by splitting the integer and fractional parts into groups of three bits, starting from the binary point on both sides. The 0s can be added to complete the outside groups if needed. Example 1.6 Let us find the binary equivalent of $(374.26)_8$ and the octal equivalent of $(1110100.0100111)_2$

Solution

• The given octal number = $(374.26)_8$

• The binary equivalent = $(011\ 111\ 100.010\ 110)_2 = (011111100.010110)_2$

• Any 0s on the extreme left of the integer part and extreme right of the fractional part of the equivalent binary number should be omitted. Therefore, $(011111100.010110)_2 = (11111100.01011)_2$

• The given binary number = $(1110100.0100111)_2$

• $(1110100.0100111)_2 = (1\ 110\ 100.010\ 011\ 1)_2 = (001\ 110\ 100.010\ 011\ 100)_2 = (164.234)_8$

Hexadecimal Number System

The 15's and 16's complements are defined with respect to the hexadecimal number system. The 15's complement is obtained by subtracting each hex digit from 15. For example, the 15's complement of $(3BF)_{16}$ would be $(C40)_{16}$. The 16's complement is obtained by adding '1' to the 15's complement. The 16's complement of $(2AE)_{16}$ would be $(D52)_{16}$.

Hexadecimal-to-Decimal Conversion

The decimal equivalent of the hexadecimal number $(1E0.2A)_{16}$ is determined as follows:

• The integer part = 1E0

• The decimal equivalent = $0 \times 160 + 14 \times 161 + 1 \times 162 = 0 + 224 + 256 = 480$

• The fractional part = 2A

• The decimal equivalent = 2 × 16−1 + 10 × 16−2 = 0.164

• Therefore, the decimal equivalent of (1E0.2A) $_{16}$ = (480.164) $_{10}$

Example 1.2 Find the decimal equivalent of the following binary numbers expressed in the 2's complement format:

(a) 00001110; (b) 10001110.

Solution

(a) The MSB bit is '0', which indicates a plus sign. The magnitude bits are 0001110.
The decimal equivalent = 0×20 +1×21 +1×22 +1×23 +0×24 +0×25 +0×26 = 0+2+4+8+0+0+0 = 14
Therefore, 00001110 represents +14

(b) The MSB bit is '1', which indicates a minus sign. The magnitude bits are therefore given by the 2's complement of 0001110, i.e., 1110010 The decimal equivalent = 0×20 +1×21 +0×22 +0×23 +1×24 +1×25 +1×26 = 0+2+0+0+16+32+64 = 114

Therefore, 10001110 represents −114

Decimal-to-Hexadecimal Conversion

The process of decimal-to-hexadecimal conversion is also similar. Since the hexadecimal number system has a base of 16, the progressive division and multiplication factor in this case is 16.

The process is illustrated further with the help of an example.

Example 1.5 Let us determine the hexadecimal equivalent of (82.25)$_{10}$

Solution

• The integer part = 82 Divisor Dividend Remainder 16 82 — 16 5 2 —0 5
• The hexadecimal equivalent of (82) $_{10}$ = (52)$_{16}$
• The fractional part = 0.25 • 0.25 × 16 = 0 with a carry of 4
• Therefore, the hexadecimal equivalent of (82.25) $_{10}$ = (52.4) $_{16}$

Hex–Binary and Binary–Hex Conversions

A hexadecimal number can be converted into its binary equivalent by replacing each hex digit with its four-bit binary equivalent. We take the four-bit equivalent because the base of the hexadecimal number system is 16 and it is the fourth power of the base of the binary number system. All we have then to remember is the four-bit binary equivalents of the basic digits of the hexadecimal number system. A given binary number can be converted into an equivalent hexadecimal number by splitting the integer and fractional

parts into groups of four bits, starting from the binary point on both sides. The 0s can be added to complete the outside groups if needed.

Example 1.7 Let us find the binary equivalent of $(17E.F6)_{16}$ and the hex equivalent of $(1011001110.011011101)_2$.

Solution

• The given hex number = $(17E.F6)_{16}$

• The binary equivalent = $(0001\ 0111\ 1110.1111\ 0110)_2$ = $(000101111110.11110110)_2$ = $(101111110.1111011)_2$

• The 0s on the extreme left of the integer part and on the extreme right of the fractional part have been omitted.

• The given binary number = $(1011001110.011011101)_2$ = $(10\ 1100\ 1110.0110\ 1110\ 1)_2$

• The hex equivalent = $(0010\ 1100\ 1110.0110\ 1110\ 1000)_2$ = $(2CE.6E8)_{16}$

Hex–Octal and Octal–Hex Conversions

For hexadecimal–octal conversion, the given hex number is firstly converted into its binary equivalent which is further converted into its octal equivalent. An alternative approach is firstly to convert the given hexadecimal number into its decimal equivalent and then convert the decimal number into an equivalent octal number. The former method is definitely more convenient and straightforward. For octal–hexadecimal conversion, the octal number may first be converted into an equivalent binary number and then the binary number transformed into its hex equivalent. The other option is firstly to convert the given octal number into its decimal equivalent and then convert the decimal number into its hex equivalent. The former approach is definitely the preferred one. Two types of conversion are illustrated in the following example.

Example 1.8

Let us find the octal equivalent of $(2F.C4)_{16}$ and the hex equivalent of $(762.013)_8$

Solution

• The given hex number = $(2F.C4)_{16}$

• The binary equivalent = $(0010\ 1111.1100\ 0100)_2$ = $(00101111.11000100)_2$ = $(101111.110001)_2$ = $(101\ 111.110\ 001)_2$ = $(57.61)_8$.

• The given octal number = $(762.013)_8$.

• The octal number = $(762.013)_8$ = $(111\ 110\ 010.000\ 001\ 011)_2$ = $(111110010.000001011)_2$ = $(0001\ 1111\ 0010.0000\ 0101\ 1000)_2$ = $(1F2.058)_{16}$

Excess – 3 Code

The excess-3 code is another important BCD code. It is particularly significant for arithmetic operations as it overcomes the shortcomings encountered while using the 8421 BCD code to add two decimal digits whose sum exceeds 9. The excess-3 code has no such limitation, and it considerably simplifies arithmetic operations.

Table 2.2 lists the excess-3 code for the decimal numbers 0–9. The excess-3 code for a given decimal number is determined by adding '3' to each decimal digit in the given number and then replacing each digit of the newly found decimal number by 22 Digital Electronics

Table 2.2 Excess-3 code equivalent of decimal numbers.

Decimal Number	Excess-3 code	Decimal number	Excess-3 code
0	0011	5	1000
1	0100	6	1001
2	0101	7	1010
3	0110	8	1011
4	0111	9	1100

its four-bit binary equivalent. It may be mentioned here that, if the addition of '3' to a digit produces a carry, as is the case with the digits 7, 8 and 9, that carry should not be taken forward. The result of addition should be taken as a single entity and subsequently replaced with its excess-3 code equivalent. As an example, let us find the excess-3 code for the decimal number 597:

• The addition of '3' to each digit yields the three new digits/numbers '8', '12' and '10'.
• The corresponding four-bit binary equivalents are 1000, 1100 and 1010 respectively.

• The excess-3 code for 597 is therefore given by: 1000 1100 1010 = 100011001010.

Also, it is normal practice to represent a given decimal digit or number using the maximum number of digits that the digital system is capable of handling. For example, in four-digit decimal arithmetic, 5 and 37 would be written as 0005 and 0037 respectively. The corresponding 8421 BCD equivalents would be 0000000000000101 and 0000000000110111 and the excess-3 code equivalents would be 0011001100111000 and 0011001101101010. Corresponding to a given excess-3 code, the equivalent decimal number can be determined by first splitting the number into four-bit groups, starting from the radix point, and then subtracting 0011 from each four-bit group. The new number is the 8421 BCD equivalent of the given excess-3 code, which can subsequently be converted into the equivalent decimal number. As an example, following these steps, the decimal equivalent of excess-3 number 01010110.10001010 would be 23.57.

Another significant feature that makes this code attractive for performing arithmetic operations is that the complement of the excess-3 code of a given decimal number yields the excess-3 code for 9's complement of the decimal number. As adding 9's complement of a decimal number B to a decimal number A achieves A – B, the excess-3 code can be used effectively for both addition and subtraction of decimal numbers.

Example 2.3

Find
(a) The excess-3 equivalent of $(237.75)_{10}$
(b) The decimal equivalent of the excess-3 numbers 1100101000011.01110101.

Solution

(a) Integer part = 237.

The excess-3 code for $(237)_{10}$ is obtained by replacing 2, 3 and 7 with the four-bit binary equivalents of 5, 6 and 10 respectively. This gives the excess-3 code for $(237)_{10}$ as: 0101 0110 1010 = 010101101010.

Fractional part = .75. The excess-3 code for $(.75)_{10}$ is obtained by replacing 7 and 5 with the four-bit binary equivalents of 10 and 8 respectively. That is, the excess-3 code for $(.75)_{10}$ = .10101000.

Combining the results of the integral and fractional parts, the excess-3 code for $(237.75)_{10}$ = 010101101010.10101000.

(b) The excess-3 code = 1100101000011.01110101 = 1100 1010 0011.0111 0101. Subtracting 0011 from each four-bit group, we obtain the new number as: 1001 0111 0000.0100 0010. Therefore, the decimal equivalent = $(970.42)_{10}$.

Logic Gates:

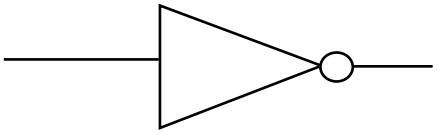
Not Gate:

A not gate is a gate whose output is the reverse of the potential output.

Truth Table:

A	NOT
0	1
1	0

Indication:



OR gate

The operation of an OR gate represents that of a normal arithmetic addition.

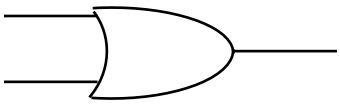
Truth Table

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

Here, In OR gate, the carry is not being considered but is used as the ultimate output.

1 + 1 = 1 0

Indication:



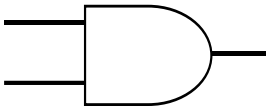
AND

AND gate is nothing but multiplication

Truth Table

A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

Indication:



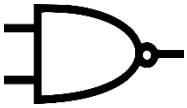
NAND

A NAND gate is nothing more but a combination of NOT and AND gate.

Truth Table

A	B	NAND
0	0	1
0	1	1
1	0	1
1	1	0

Indication



NOR Gate

Just like NAND gate, NOR gate is a combination of NOT gate and OR gate.

Thus, reverses the output of OR gate.

Truth Table

A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

Indication

EX-OR gate

The Exclusive-OR gate or XOR gate is achieved by combining standard logic gates together. XOR gate is used extensively in error detection circuits, computational logic comparators and arithmetic logic circuits. The Exclusive OR gate gives an output only if its two inputs are dissimilar, namely if one of them is high (one) and the other is low (zero).

Truth Table

The Boolean expression representing the 2 input XOR gate is written as

Y = A (+) B = A'B + AB'

The Boolean expression for the three-input logic gate is given by

Y = A (+) B (+) C = A'BC+AB'C+A'BC'+ ABC

A	B	EX-OR
0	0	0
0	1	1
1	0	1
1	1	0

Indication

EX-NOR

A simple NOR gate symbol can be denoted by a standard OR gate with an inversion bubble connected. The logic symbol of an Exclusive-NOR gate is an XOR gate (Exclusive-OR gate) with the "inversion bubble" or the circle in front. Therefore, the Exclusive-NOR gate is the complementary form of the Exclusive-OR gate.

The Boolean expression for the XNOR gate with 2 input is given by

Y = A' (+) B' = A'B' + AB

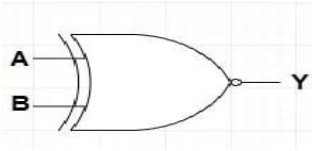
The Boolean expression for the XNOR gate with 3 input is given by

Y = A' (+) B' (+) C' = A'B'C + A'BC + AB'C + ABC'

Truth Table

A	B	EX-NOR
0	0	1
0	1	0
1	0	0
1	1	1

Indication



De Morgan's Theorems

De Morgan's First Theorem

De Morgan's First theorem proves that when two (or more) input variables are AND'ed and negated, they are equivalent to the OR of the complements of the individual variables. Thus, the equivalent of the NAND function will be a negative-OR function, proving that $\overline{A.b} = A' + B'$. We can show this operation using the following table.

Verifying De Morgan's First Theorem using Truth Table

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

De Morgan's Second Theorem

De Morgan's Second theorem proves that when two (or more) input variables are OR'ed and negated, they are equivalent to the AND of the complements of the individual variables. Thus the equivalent of the NOR function is a negative-AND function proving that A+B = A.B, and again we can show operation this using the following truth table.

Verifying De Morgan's Second Theorem using Truth Table

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

In addition

Refer (This) for laws of algebra

Universal building blocks (NOT, OR, AND) Binary addition and subtraction

Binary Addition

Truth Table

A+B	Sum	Carry
0+0	0	0
0+1	1	0
1+0	1	0
1+1	0	1

Assume two integers 3 & 5,

Converting that to Binary equivalent, we get

```
  3 = 0011
+5 = 0101
-------
  8 = 1000
```

In order to understand the addition process of two binary equivalents,

We must take into consideration the left-overs or carry.

For instance, we start from RHS,

```
  3 = 0011
+5 = 0101
-------
  8 = 1000
```

1 + 1 = 2, But in Binary digits we only have 0 and 1 as possible options, and 2 in Binary is 1 0. Now, we consider the right digit first i.e., 0 and 1 as carry. So,

```
  3 = 0011
+5 = 0101
-------
       0 (1 carry)
```

Moving on to next,

```
  3 = 0011
+5 = 0101
-------
```

1 + 0 = 1. And we have a carry of 1. Now, 1 + 1(carry) = 2 (1 0) again. So, we write 0 and take 1 as the carry.

```
  3 = 0011
+5 = 0101
---------
     00 (1 carry)
```

Next, we have 0 + 1 = 1

```
  3 = 0011
+5 = 0101
---------
```

Same as 1 + 0, in 0 + 1, too, we write down 0 and take 1 as carry.

```
  3 = 0011
+5 = 0101
---------
    000
```

Lastly, we have 0 + 0. And 0 + 0 is 0. Then adding it with the carry, 0+1 = 1.

```
  3 = 0011
+5 = 0101
---------
   1000
```

Thus, 3+5 = 8, and 8's Binary equivalent is 1000.

Binary Subtraction

A-B	Difference	Borrow
0-0	0	0
0-1	1	1
1-0	1	0
1-1	0	0

Just like addition, binary subtraction is nothing more than a normal subtraction, but with Binary digits. In binary subtraction, instead of carry we deal with borrows.

Assume the same instance

5 = 0011

-3 = 0101

2 = 0010

In the first RHS,

5 = 0101

-3 = 0011

0010

The digit 1 – 1 = 0. Well, we simply write that down.

Onto the second one, 0-1 = -1. Here is where the "Borrow" concept comes into play.

(Borrow 1 from its left neighbour) 1 0 – 1 = 1 (2-1=1).

5 = 0101

-3 = 0011

10

5 = 00(10)1

-3 = 0011

0010

1's Complement

A 1's complement is nothing but the inversion of binary digits.

For instance, assume a 4-bits binary digit 0101,

Its 1's complement equivalent is 1010.

2's Complement

On the other hand, 2's complement is the addition of one to the 1's complement.

For instance,

Given binary digits: 1010

1's complement: 0101

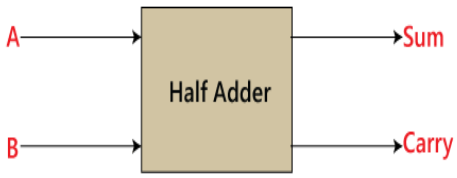
2s complement: + 1

Thus, = 0110

Half adder

The Half-Adder is a basic building block of adding two numbers as two inputs and produce out two outputs. The adder is used to perform OR operation of two single bit binary numbers. The augent and addent bits are two input states, and 'carry' and 'sum 'are two output states of the half adder.

Block Diagram



Truth Table

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

In the above table,

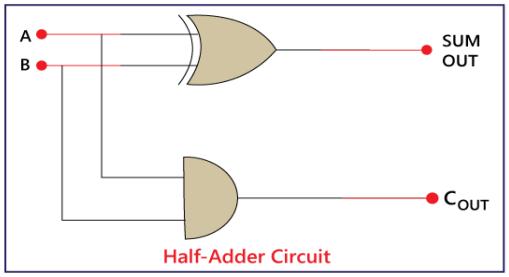
'A' and' B' are the input states, and 'sum' and 'carry' are the output states.

The carry output is 0 in case where both the inputs are not 1.

The least significant bit of the sum is defined by the 'sum' bit.

The SOP form of the sum and carry are as follows:

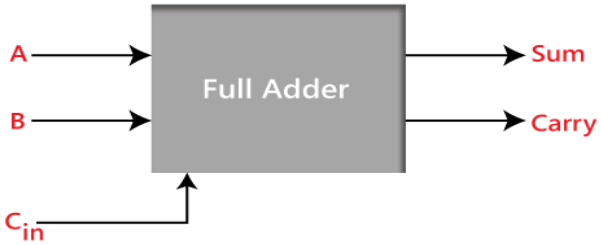
Sum = x'y+xy' (A ex-or B)
Carry = xy



Full Adder

The half adder is used to add only two numbers. To overcome this problem, the full adder was developed. The full adder is used to add three 1-bit binary numbers A, B, and carry C. The full adder has three input states and two output states i.e., sum and carry.

Block Diagram



Truth Table

A	B	C	Sum	Carry
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

Sum = A'B'C + A'BC' + AB'C' + ABC

= C (A'A + B'B) + C' (A'B+ AB')

= C⊕ A⊕B

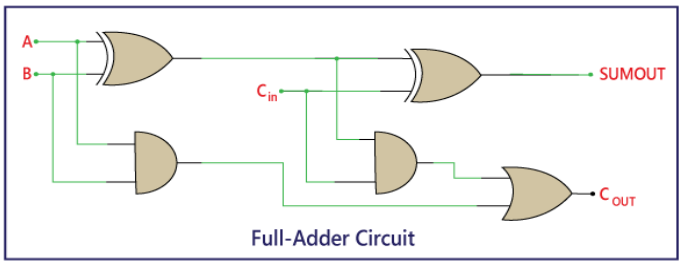
= A ⊕ B ⊕ C

Carry= A'BC+AB'C+ABC'+ABC

= AB + C (A'B + AB')

= AB + C($A \oplus B$)

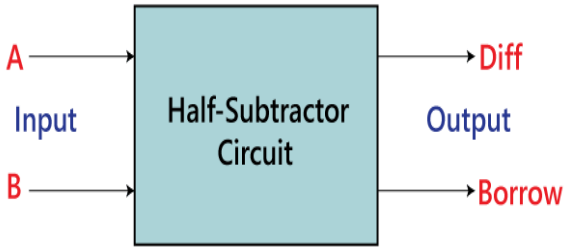
Refer this and this



Half subtractor

The half subtractor is also a building block for subtracting two binary numbers. It has two inputs and two outputs. This circuit is used to subtract two single bit binary numbers A and B. The 'diff' and 'borrow' are two output states of the half subtractor.

Block diagram



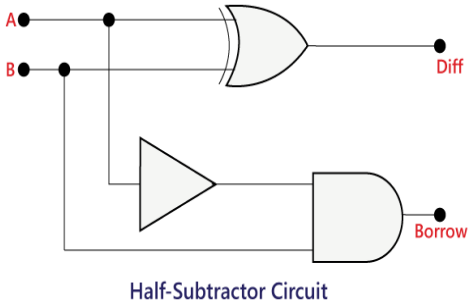
Truth Table

Inputs		Outputs	
A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

The SOP form of the Diff and Borrow is as follows:

Diff= A'B+AB'

Borrow = A'B

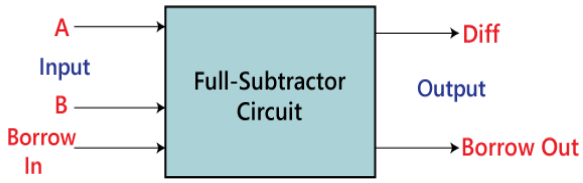


Half-Subtractor Circuit

Full Subtractor

The Half Subtractor is used to subtract only two numbers. To overcome this problem, a full subtractor was designed. The full subtractor is used to subtract three 1-bit numbers A, B, and C, which are minuend, subtrahend, and borrow, respectively. The full subtractor has three input states and two output states i.e., diff and borrow.

Block diagram

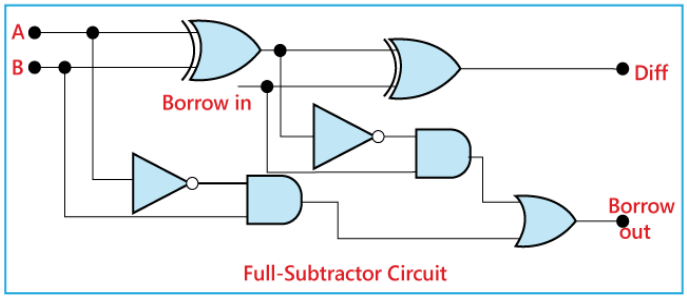


Truth Table

Inputs			Outputs	
A	B	$Borrow_{in}$	Diff	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

Borrow=x' z+x' y+yz

Diff=xy' z'+x' y' z+xyz+x'yz'



Diff:

Perform the XOR operation of input A and B.

Perform the XOR operation of the outcome with 'Borrow'. So, the difference is (A XOR B) XOR 'Borrowin' which is also represented as: (A ⊕ B) ⊕ 'Borrowin'

Borrow:

Perform the 'AND' operation of the inverted input A and B.

Perform the 'XOR' operation of input A and B.

Perform the 'OR' operations of both the outputs that come from the previous two steps. So the 'Borrow' can be represented as: A'.B + (A ⊕ B)'

Unit 2

Boolean Algebra

Refer this

Sum of Product (SOP)

Sum of Product is the abbreviated form of SOP. Sum of product form is a form of expression in Boolean algebra in which different product terms of inputs are being summed together. This product is not arithmetical multiply but it is Boolean logical AND and the Sum is Boolean logical OR.

To understand better about SOP, we need to know about min term.

Min Term

Min term means the term that is true for a minimum number of combination of inputs. That is true for only one combination of inputs.

Since AND gate also gives True only when all of its inputs are true so we can say min terms are AND of input combinations like in the table given below.

A	B	C	Min term
0	0	0	$m_0 = \bar{A}\,\bar{B}\,\bar{C}$
0	0	1	$m_1 = \bar{A}\,\bar{B}\,C$
0	1	0	$m_2 = \bar{A}\,B\,\bar{C}$
0	1	1	$m_3 = \bar{A}\,B\,C$
1	0	0	$m_4 = A\,\bar{B}\,\bar{C}$
1	0	1	$m_5 = A\,\bar{B}\,C$
1	1	0	$m_6 = A\,B\,\bar{C}$
1	1	1	$m_7 = A\,B\,C$

3 inputs have 8 different combinations. Each combination has a Min Terms denoted by small m and its decimal combination number written in subscript. Each of these min terms will be only true for the specific input combination.

Types of Sum Of Product (SOP) Forms

There are few different forms of Sum of Product.

- Canonical SOP Form
- Non-Canonical SOP Form
- Minimal SOP Form

Canonical SOP Form

This is the standard form of Sum of Product. It is formed by O Ring the minterms of the function for which the output is true. This is also known as Sum of Min terms or Canonical disjunctive normal form (CDNF). It is just a fancy name. "canonical" means "standardized" and "disjunctive" means "Logical OR union".

Canonical SOP expression is represented by summation sign ∑ and minterms in the braces for which the output is true.

For example, a functions truth table is given below.

(Note that truth table may differ)

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

For this function the canonical SOP expression is

F = ∑ (m1, m2, m3, m5)

Which means that the function is true for the min terms {1, 2, 3, 5}.

By expanding the Summation we get.

F = m1 + m2 + m3 + m5

Now putting min terms in the expression

F = A'B'C + A'BC' + A'BC + AB'C

Canonical form contains all inputs either complemented or non-complemented in its product terms.

Non-Canonical SOP Form

As the name suggests, this form is the non-standardized form of SOP expressions. The product terms are not the min terms but they are simplified. Let's take the above function in canonical form as an example.

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

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

$$F = \bar{A}\bar{B}C + \bar{A}B(1) + A\bar{B}C$$

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

This expression is still in Sum of Product form but it is non-canonical or non-standardized form.

Minimal SOP Form

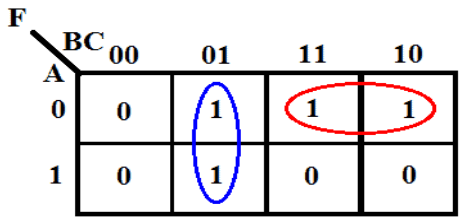
This form is the most simplified SOP expression of a function. It is also a form of non-canonical form. Minimal SOP form can be made using Boolean algebraic theorems but it is very easily made using Karnaugh map (K-map).

Minimal SOP form is preferred because it uses the minimum number of gates and input lines. it is commercially beneficial because of its compact size, fast speed, and low fabrication cost.

Let's take an example of the function given above in canonical form.

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

Its **K-map** is given below.



According to the K-map, the output expression will be

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

This is the most simplified & optimized expression for the said function. This expression requires only two 2-input AND gates & one 2-input OR gate. However, the canonical form needs four 3-input AND gates & one 4-input OR gate, which is relatively more costly than minimal form implementation.

Product of Sum

The short form of the product of the sum is POS, and it is one kind of Boolean algebra expression. In this, it is a form in which products of the dissimilar sum of inputs are taken, which are not arithmetic result & sum although they are logical Boolean AND & OR correspondingly. Before going to understand the concept of the product of the sum, we have to know the concept of the max term.

The maxterm can be defined as a term that is true for the highest number of input combinations otherwise that is false for single input combinations. Because OR gate also provides false for just one input combination. Thus, Max term is OR of any complemented otherwise non-complemented inputs.

X	Y	Z	Max Term (M)
0	0	0	X+Y+Z = M0
0	0	1	X+Y+Z' = M1
0	1	0	X+Y'+ Z = M2
0	1	1	X+Y'+Z' = M3
1	0	0	X'+Y+Z= M4
1	0	1	X'+Y+Z' = M5
1	1	0	X'+Y'+Z = M6
1	1	1	X'+Y'+Z' = M7

In the above table, there are three inputs namely X, Y, Z and the combinations of these inputs are 8. Every combination has a max term that is specified with M.

In max term, every input is complemented as it provides only '0' while the stated combination is applied & complement of minterm is a max term.

M3 =m3'
(X'YZ)' = M3
X+Y'+Z'=M3 (De Morgan's Law)

Types of Products of Sum (POS)

The product of the sum is classified into three types which include the following.

Canonical Product of Sums

Non – Canonical Product of Sums

Minimal Product of Sums

(This method is basically the opposite of Sum Of Product)

(Refer this)

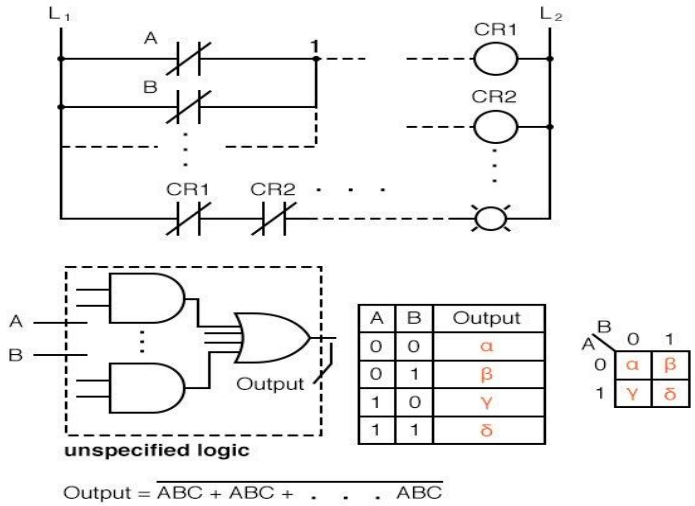
K-MAP

The K-map is a systematic way of simplifying Boolean expressions. With the help of the K-map method, we can find the simplest POS and SOP expression, which is known as the minimum expression. The K-map provides a cookbook for simplification.

Just like the truth table, a K-map contains all the possible values of input variables and their corresponding output values. However, in K-map, the values are stored in cells of the array. In each cell, a binary value of each input variable is stored.

(Refer this)

Truth Table



A	B	Output
0	0	α
0	1	β
1	0	γ
1	1	δ

A \ B	0	1
0	α	β
1	γ	δ

Pairs, Quads and Octet

Pairs, Quads, and Octets

Pairs

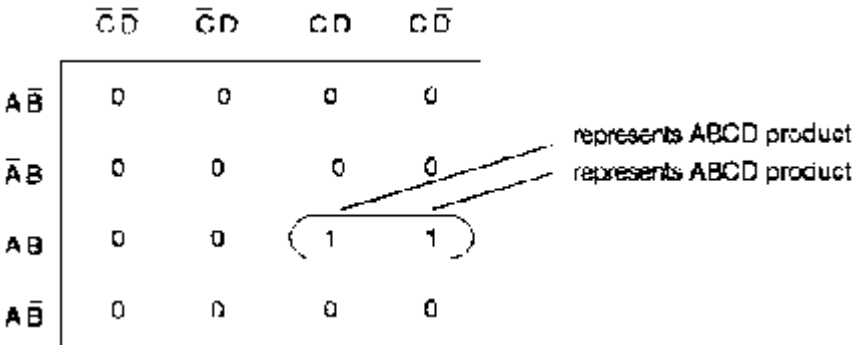


Fig. 4-4: Four variable simplification

As you see in Fig. 4-4, only one variable goes from uncomplement to complement. Whenever this happens, you can eliminate the variable that changes form.

Proof: $X = ABCD + ABC\bar{D}$

$$X = ABC(D + \bar{D})$$

X = A B C

Ex:

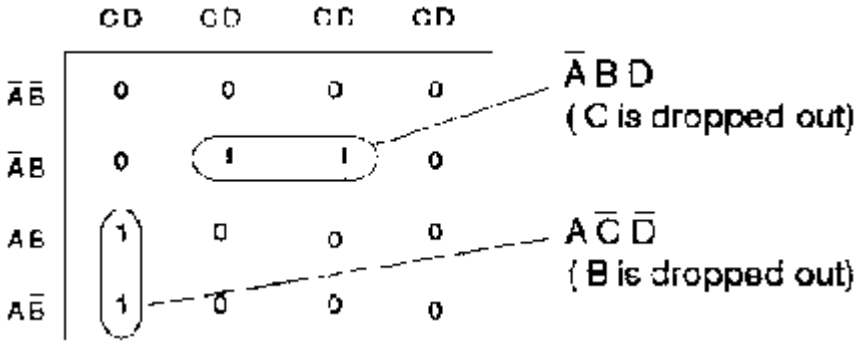


Fig. 4-5: Pairs

Whenever you see a pair first encircle it and then simplify to get the simplified Boolean expression:

$$X = A\bar{C}\bar{D} + \bar{A}BD$$

Quad

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

Fig. 4-6: Quad

Quad: A group of 4 one's that are horizontally or vertically adjacent. End to end or in form of a square.

A quad eliminates two variables and their complements.

Proof: $X = AB\bar{C} + ABC$ (two pairs)

X = A B (C + C)

X = A B

Encircle the quad and step through the different one's in the quad and determine which two variables go from complement to uncomplement (or vs), these are the variables that drop out.

Ex:

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

Fig. 4-7: Quad

The variables B and D can be eliminated. So we get the following equation:

X = A C

Octet

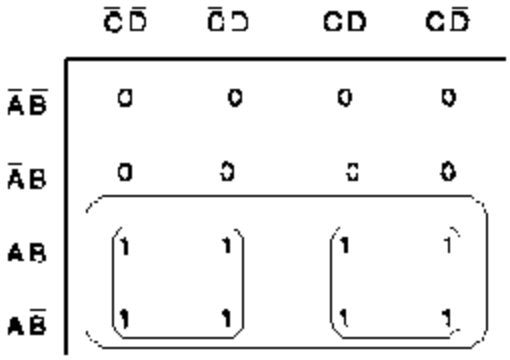


Fig. 4-8: Octet

An octet eliminates three variables and their complements.

Proof: $X = A\bar{C} + AC$ (two quads)

$X = A\ (C + C)$

$X = A$

Karnaugh Simplifications

Process:

1. Draw the Karnaugh map
2. Look for octets and encircle them.
3. Look for quads and encircle them.
4. Look for pairs and encircle them.
5. Simplify and write down the equation.

Ex:

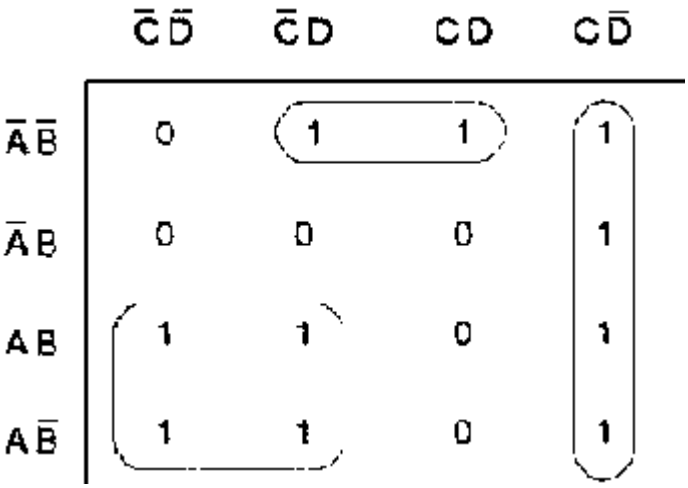


Fig. 4-9: Karnaugh map

$X = A\bar{C} - C\bar{D} + \bar{A}BD$

Overlapping and Rolling

Overlapping groups

Ex:

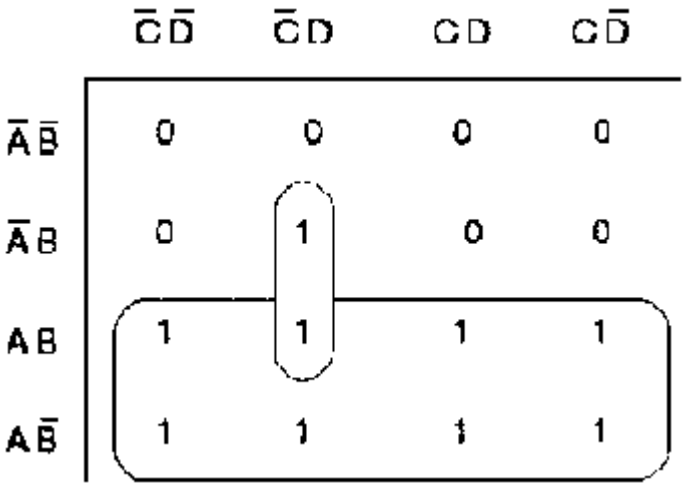


Fig. 4-10: Karnaugh map

Groups can overlap to get a simpler equation:

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

Rolling the map

Ex:

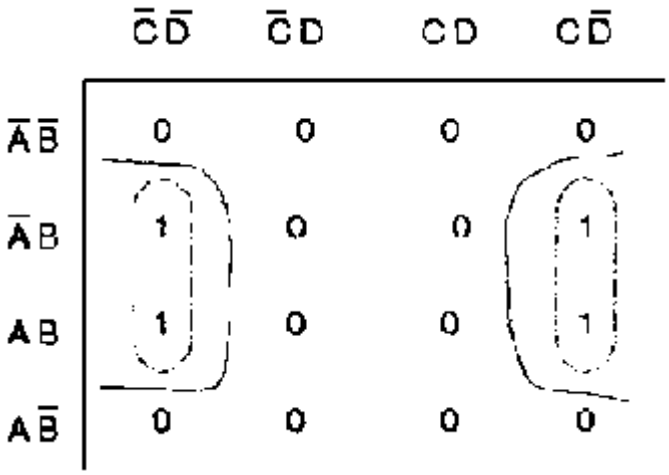


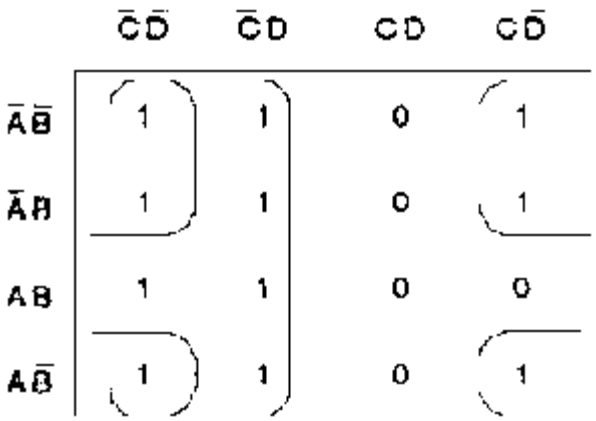
Fig. 4-11: Karnaugh map

Instead of encircling two pairs:

$$X = B\bar{C}D - BC\bar{D}$$

We can roll the map and encircle a quad:

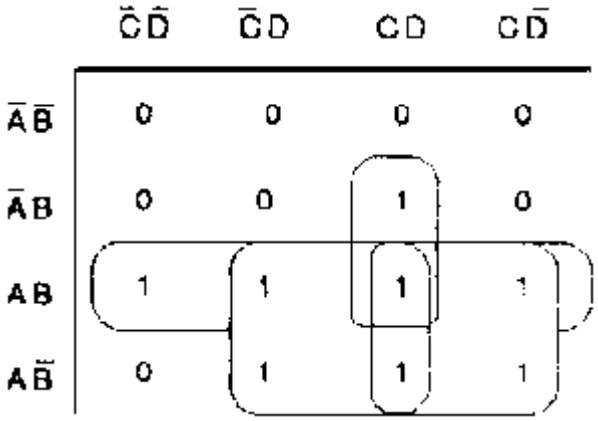
$$X = B\bar{D}$$



HO: Simplify the following map.

Solution:

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



HO: Simplify the following map.

Solution:

$$X = AB + AD + AC + BCD$$

K-Map Simplification

Refer this

Digital logic families and their parameters

In Digital Designs, our primary aim is to create an Integrated Circuit (IC). A Circuit configuration or arrangement of the circuit elements in a special manner will result in a particular Logic Family. Electrical Characteristics of the IC will be identical. In other words, the different parameters like Noise Margin, Fan In, Fan Out etc will be identical.

Different ICs belonging to the same logic families will be compatible with each other.

The basic Classification of the Logic Families are as follows:
A) Bipolar Families
B) MOS Families
C) Hybrid Devices

A) Bipolar Families:
1. Diode Logic (DL)
2. Resistor Transistor Logic (RTL)
3. Diode Transistor Logic (DTL)
4. Transistor- Transistor Logic (TTL)
5. Emitter Coupled Logic (ECL) or Current Mode Logic (CML)
6. Integrated Injection Logic (IIL)

B) MOS Families:
1. P-MOS Family
2. N-MOS Family
3. Complementary-MOS Family
â€¢ Standard C-MOS
â€¢ Clocked C-MOS
â€¢ Bi-CMOS
â€¢ Pseudo N-MOS
â€¢ C-MOS Domino Logic
â€¢ Pass Transistor Logic

C) Hybrid Family:
Bi-CMOS Family

Diode Logic
In DL (diode logic), only Diode and Resistors are used for implementing a particular Logic. Remember that the Diode conducts only when it is Forward Biased.

Resistor Transistor Logic
In RTL (resistor transistor logic), all the logic are implemented using resistors and transistors. One basic thing about the transistor (NPN), is that HIGH at input causes output to be LOW (i.e. like a inverter). In the case of PNP transistor, the LOW at input causes output to be HIGH.

Diode Transistor Logic
In DTL (Diode transistor logic), all the logic is implemented using diodes and transistors.

Transistor Transistor Logic

In Transistor Transistor logic or just TTL, logic gates are built only around transistors.

Emitter Coupled Logic
The main specialty of ECL is that it is operating in Active Region than the Saturation Region. That is the reason for its high-speed operation. As you can see in the figure, the Emitters of the Transistors Q1 and Q2 are coupled together.

Some Characteristics we consider for the selection of a particular Logic Family are:
â€¢ Supply voltage range
â€¢ Speed of response
â€¢ Power dissipation
â€¢ Input and output logic levels
â€¢ Current sourcing and sinking capability
â€¢ Fan in
â€¢ Fan-out
â€¢ Noise margin

Introduction of Digital logic families

1. Small scale integration SSI <12 no of gates

2. Medium scale integration MSI 12 to 99 no of gates

3. Large scale integration LSI 100 to 9999 no of gates

4. Very large-scale integration VLSI 10,000 or more

Characteristics/Parameters of Digital ICs
Input /Output voltage level:
The following currents and voltages are specified which are very useful in the design of digital systems.

High-level input voltage, VIH: This is the minimum input voltage which is recognized by the gate as logic 1.

Low-level input voltage, VIL: This is the maximum input voltage which is recognized by the gate as logic 0.
High-level output voltage, VOH: This is the minimum voltage available at the output corresponding to logic 1.

Low-level output voltage, VOL: This is the maximum voltage available at the output corresponding to logic 0.

High-level input current, IIH: This is the minimum current which must be supplied by a driving source corresponding to 1 level voltage.

Low-level input current, IIL: This is the minimum current which must be supplied by a driving source corresponding to 0 level voltage.

High-level output current, IOH: This is the maximum current which the gate can sink in 1 level.

Low-level output current, IOL: This is the maximum current which the gate can sink in 0 level.

High-level supply current, ICC (1): This is the supply current when the output of the gate is at logic 1.

Low-level supply current, ICC (0): This is the supply current when the output of the gate is at logic (0).

Propagation Delay:

Definition: The time required for the output of a digital circuit to change states after a change at one or more of its inputs. The speed of a digital circuit is specified in terms of the propagation delay time. The delay times are measured between the 50 percent voltage levels of input and output waveforms. There are two delay times, tpHL: when the output goes from the HIGH state to the LOW state and tpLH, corresponding to the output making a transition from the LOW state to the HIGH state. The propagation delay time of the logic gate is taken as the average of these two delay times.

Fan-in

Definition: Fan-in (input load factor is the number of input signals that can be connected to a gate without causing it to operate outside its intended operating range. expressed in terms of standard inputs or units loads (ULs).

Fan-out

Definition: Fan-out (output load factor) is the maximum number of inputs that can be driven by a logic gate. A fanout of 10 means that 10 unit loads can be driven by the gate while still maintaining the output voltage within specifications for logic levels 0 and 1.

Digital IC gates are classified not only by their logic operation, but also by the specific logic circuit family to which it belongs. Each logic family has its own basic electronic circuit upon which more complex digital circuits and functions are developed.
Different types of logic gate families :
RTL : Resistor Transistor Logic gate family
DCTL : Direct Coupled Transistor Logic gate family
RCTL : Resistor Capacitor Transistor Logic gate family

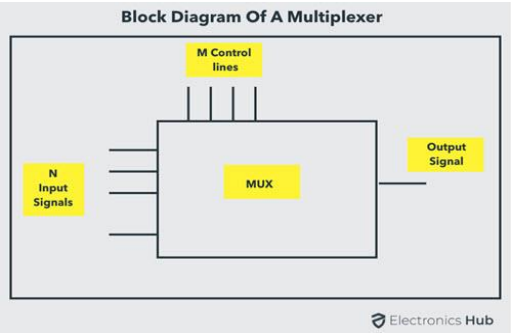
DTL : Diode Transistor Logic gate family
TTL : Transistor Transistor logic gate family
IIL : Integraeted Injection gate family

Multiplexer

Multiplexer means many into one. A multiplexer is a circuit used to select and route any one of the several input signals to a single output. A simple example of a non-electronic circuit of a multiplexer is a single pole multi-position switch.

Multiplexers can handle two types of data i.e., analog and digital. For analog application, multiplexers are built using relays and transistor switches. For digital application, they are built from standard logic gates.

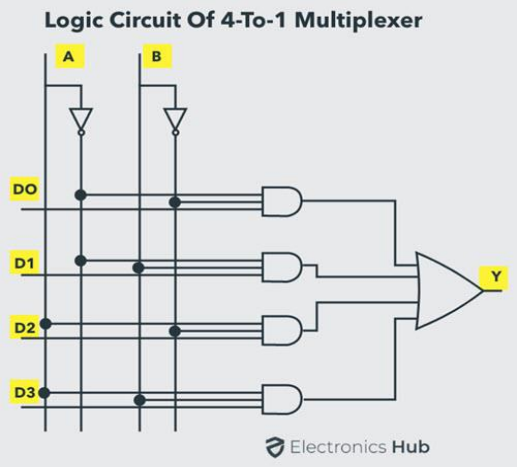
The multiplexer used for digital applications, also called digital multiplexer, is a circuit with many inputs but only one output. By applying control signals (also known as Select Signals), we can steer any input to the output. Some of the common types of multiplexers are 2-to-1, 4-to-1, 8-to-1, 16-to-1 multiplexer.



Understanding 4-to-1 Multiplexer

The 4-to-1 multiplexer has 4 input bits, 2 control or select bits, and 1 output bit. The four input bits are D0, D1, D2 and D3. Only one of this is transmitted to the output Y. The output depends on the values of A and B, which are the control inputs. The control input determines which of the input data bit is transmitted to the output.

For instance, as shown in figure, when A B = 0 0, the upper AND gate is enabled, while all other AND gates are disabled. Therefore, data bit D0 is transmitted to the output, giving Y = Do.



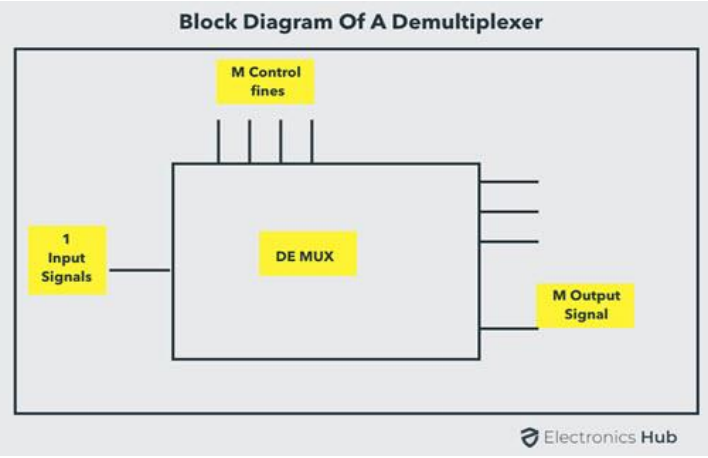
If the control input is changed to A B = 1 1, all gates are disabled except the bottom AND gate. In this case, D3 is transmitted to the output and Y = D3.

(Refer this for rest)

Demultiplexer

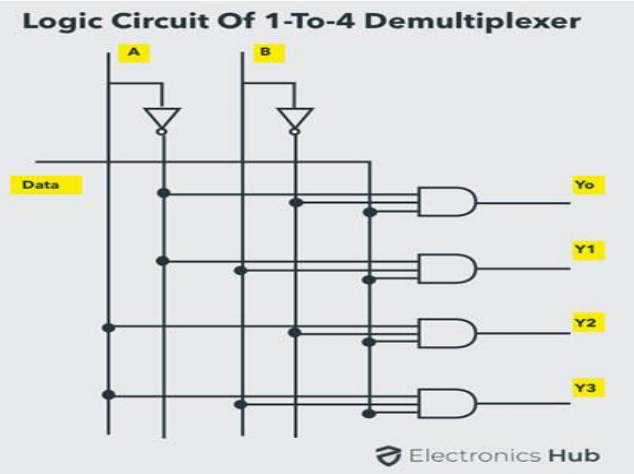
Demultiplexer means one to many. A demultiplexer is a circuit with one input and many outputs. By applying control signal, we can steer any input to the output. Few types of demultiplexer are 1-to 2, 1-to-4, 1-to-8 and 1-to 16 demultiplexer.

Following figure illustrate the general idea of a demultiplexer with 1 input signal, m control signals, and n output signals.



Understanding 1-to-4 Demultiplexer

The 1-to-4 demultiplexer has 1 input bit, 2 control or select bits, and 4 output bits. An example of 1-to-4 demultiplexer is IC 74155. The 1-to-4 demultiplexer is shown in figure below-



The input bit is labelled as Data D. This data bit is transmitted to the selected output lines, which depends on the values of A and B, the control or Select Inputs.

When A B = 0 1, the second AND gate from the top is enabled while other AND gates are disabled. Therefore, data bit D is transmitted to the output Y1, giving Y1 = Data.

If D is LOW, Y1 is LOW. If D is HIGH, Y1 is HIGH. The value of Y1 depends upon the value of D. All other outputs are in low state.

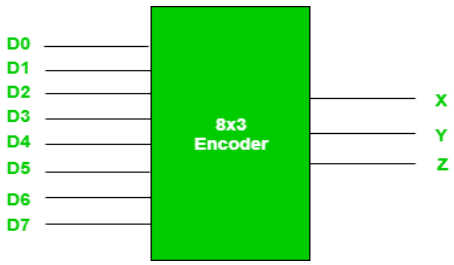
If the control input is changed to A B = 1 0, all the gates are disabled except the third AND gate from the top. Then, D is transmitted only to the Y2 output, and Y2 = Data.

(Refer this for the rest)

Decoder and Encoders

1. Encoders –
An encoder is a combinational circuit that converts binary information in the form of a 2N input lines into N output lines, which represent N bit code for the input. For simple encoders, it is assumed that only one input line is active at a time.

As an example, let's consider Octal to Binary encoder. As shown in the following figure, an octal-to-binary encoder takes 8 input lines and generates 3 output lines.



Truth Table –

D7	D6	D5	D4	D3	D2	D1	D0	X	Y	Z
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

As seen from the truth table, the output is 000 when D0 is active; 001 when D1 is active; 010 when D2 is active and so on.

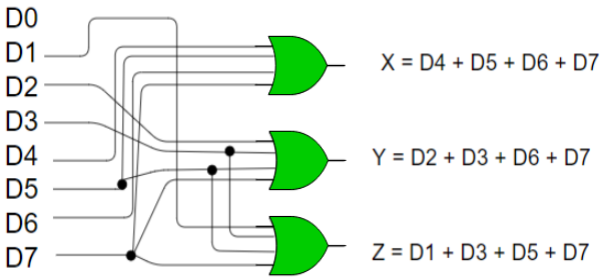
Implementation –
From the truth table, the output line Z is active when the input octal digit is 1, 3, 5 or 7. Similarly, Y is 1 when input octal digit is 2, 3, 6 or 7 and X is 1 for input octal digits 4, 5, 6 or 7. Hence, the Boolean functions would be:

X = D4 + D5 + D6 + D7

Y = D2 +D3 + D6 + D7

Z = D1 + D3 + D5 + D7

Hence, the encoder can be realised with OR gates as follows:



One limitation of this encoder is that only one input can be active at any given time. If more than one inputs are active, then the output is undefined. For example, if D6 and D3 are both active, then, our output would be 111 which is the output for D7. To overcome this, we use Priority Encoders.

Another ambiguity arises when all inputs are 0. In this case, encoder outputs 000 which actually is the output for D0 active. In order to avoid this, an extra bit can be added to the output, called the valid bit which is 0 when all inputs are 0 and 1 otherwise.

2. Decoders –

A decoder does the opposite job of an encoder. It is a combinational circuit that converts n lines of input into 2n lines of output.

Let's take an example of 3-to-8 lines decoder.

Truth Table –

X	Y	Z	D0	D1	D2	D3	D4	D5	D6	D7
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

X	Y	Z	D0	D1	D2	D3	D4	D5	D6	D7
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Implementation –

D0 is high when X = 0, Y = 0 and Z = 0. Hence,

D0 = X' Y' Z'

Similarly,

D1 = X' Y' Z

D2 = X' Y Z'

D3 = X' Y Z

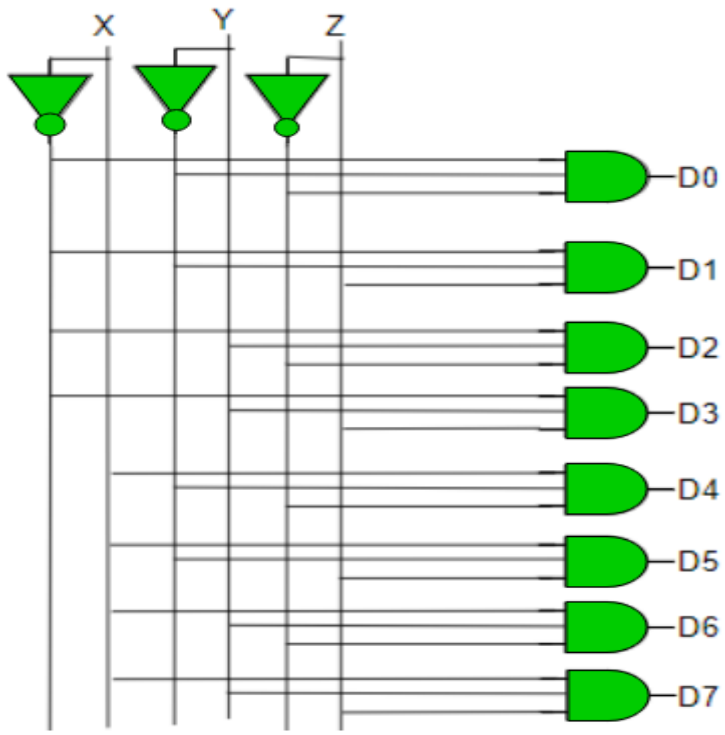
D4 = X Y' Z'

D5 = X Y' Z

D6 = X Y Z'

D7 = X Y Z

Hence,



BCD to Decimal Decoder

(?)

Parity Checkers

Definition: The parity bit or check bit are the bits added to the binary code to check whether the particular code is in parity or not, for example, whether the code is in even parity or odd parity is checked by this check bit or parity bit. The parity is nothing but number of 1's and there are two types of parity bits they are even bit and odd bit.

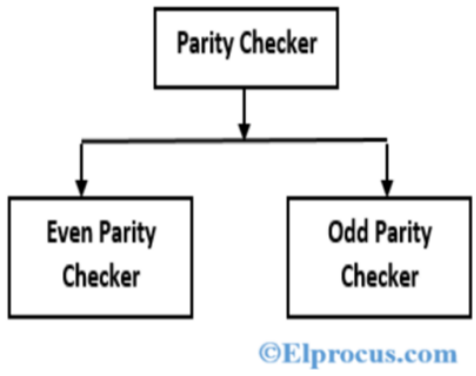
In odd parity bit, the code must be in an odd number of 1's, for example, we are taking 5-bit code 100011, this code is said to be odd parity because there is three number of 1's in the code which we have taken. In even parity bit the code must be in even number of 1's, for example, we are taking 6-bit code 101101, this code is said to be even parity because there are four number of 1's in the code which we have taken.

What is parity checker?

Definition: The combinational circuit at the receiver is the parity checker. This checker takes the received message including the parity bit as input. It gives output '1' if there is some error found and gives output '0' if no error is found in the message including the parity bit.

Types of Parity Checker

The classification of the parity checker is shown in the below figure



Types-of-parity-checker

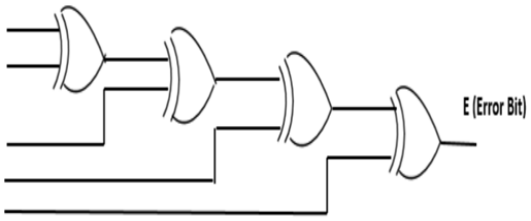
Even Parity Checker

In even parity checker if the error bit (E) is equal to '1', then we have an error. If error bit E=0 then indicates there is no error.

Error Bit (E) =1, error occurs

Error Bit (E) =0, no error

The parity checker circuit is shown in the below figure



©Elprocus.com

Odd Parity Checker

In odd parity checker if an error bit (E) is equal to '1', then it indicates there is no error. If an error bit E=0 then indicates there is an error.

Error Bit (E) =1, no error

Error Bit (E) =0, error occurs

The parity checker won't be able to detect if there are errors in more than '1' bit and the correct of data is also not possible, these are the main disadvantages of the parity checker.

For more, refer this

Unit 3

(Video Guide here)

Types of Flip-Flops:

There are basically four different types of flip flops and these are:

1. Set-Reset (SR) flip-flop or Latch

2. JK flip-flop

3. D (Data or Delay) flip-flop

4. T (Toggle) flip-flop

5. Master Slave Flip-Flop

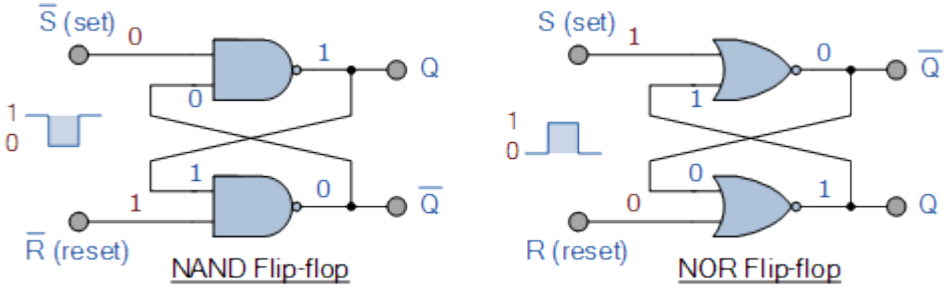
Set Reset:

The SR flip flop is a 1-bit memory bistable device having two inputs, i.e., SET and RESET. The SET input 'S' set the device or produce the output 1, and the RESET input 'R' reset the device or produce the output 0. The SET and RESET inputs are labelled as S and R, respectively.

The SR flip flop stands for "Set-Reset" flip flop. The reset input is used to get back the flip flop to its original state from the current state with an output 'Q'. This output depends on the set and reset conditions, which is either at the logic level "0" or "1".

The NAND gate SR flip flop is a basic flip flop which provides feedback from both of its outputs back to its opposing input. This circuit is used to store the single data bit in the memory circuit. So, the SR flip flop has a total of three inputs, i.e., 'S' and 'R', and current output 'Q'. This output 'Q' is related to the current history or state. The term "flip-flop" relates to the actual operation of the device, as it can be "flipped" to a logic set state or "flopped" back to the opposing logic reset state.

Basic NAND and NOR SR Flip-flops



Above are the two basic configurations for the asynchronous SR bistable flip-flop using either a negative input NAND gate, or a positive input NOR gate. For the SR bistable latch using two cross-coupled NAND gates operates with both inputs normally HIGH at logic level "1".

The application of a LOW at logic level "0" to the S input with R held HIGH causes output Q to go HIGH, setting the latch. Likewise, a logic level "0" on the R input with input S held HIGH causes the Q output to go LOW, resetting the latch. For the SR NAND gate latch, the condition of S = R = 0 is forbidden.

For the conversion of flip-flops using two cross-coupled NOR gates, when the output Q = 1 and Q = 0, the bistable latch is said to be in the Set state. When Q = 0 and Q = 1, the NOR gate latch is said to be in its Reset state. Then we can see that the operation of the NOR and NAND gate flip-flops are basically just the complements of each other.

The implementation of an SR flip-flop using two cross-coupled NAND gates requires LOW inputs. However, we can convert the operation of a NAND SR flip-flop to operate in the same manner as the NOR gate implementation with active HIGH (positive logic) inputs by using inverters, (NOT Gates) within the basic bistable design.

(Refer this to understand the Truth Table, functioning, and the logic circuit)

(This)

D Flip-Flop

(Understand "Edge Triggering")

D type Edge Triggered flip flop

D edge triggered flip-flop is the flip-flop in which the output can change only with the edge of the clock pulse, regardless of the change in the input. That means the output of the flip-flop changes with the transition of the clock pulse, either from high to low to high.

D type Edge Triggered flip flop type

Edge triggered D type flip flop can be of 2- types:

- Negative edge trigger D type flip flop.
- Positive Edge trigger D type flip flop.

The edge triggered flip Flop is also called dynamic triggering flip flop.

Edge Triggered D flip flop Circuit Diagram

The circuit diagram of the edge triggered D type flip flop explained here. First, the D flip-flop is connected to an edge detector circuit, which will detect the negative edge or positive edge of the clock pulse. Then, according to the output of the edge detector circuit, the D flip flop will operate accordingly.

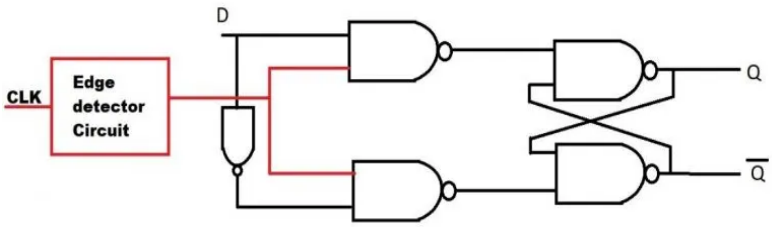
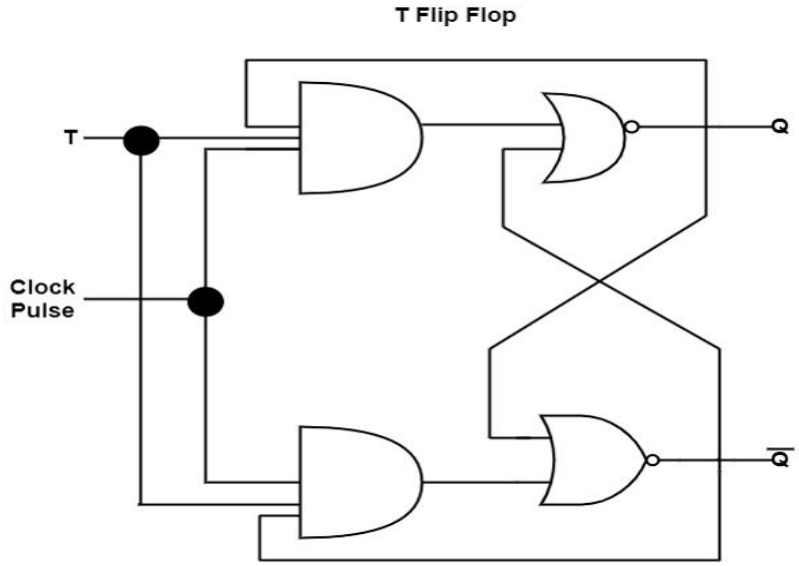


Fig. Circuit diagram of edge triggered d type flip flop

Edge Triggered T Flip-Flop

The Toggle Flip-flop is another type of bistable sequential logic circuit based around the previous clocked JK flip-flop circuit. The toggle flip-flop can be used as a basic digital element for storing one bit of information, as a divide-by-two divider or as a counter. Toggle flip-flops have a single input and one or two complementary outputs of Q and Q which change state on the

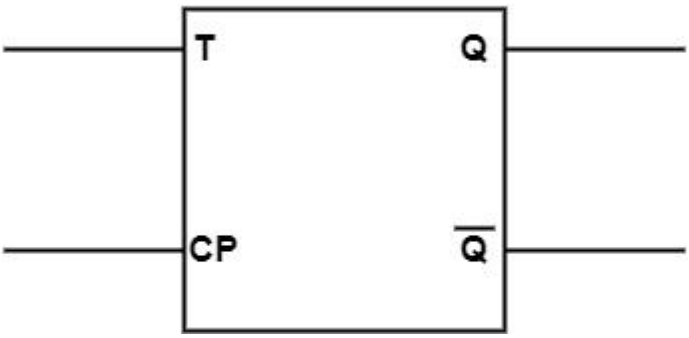
positive edge (rising edge) or negative edge (falling edge) of an input clock signal or pulse.



The truth table of T flip-flop is displayed in the table.

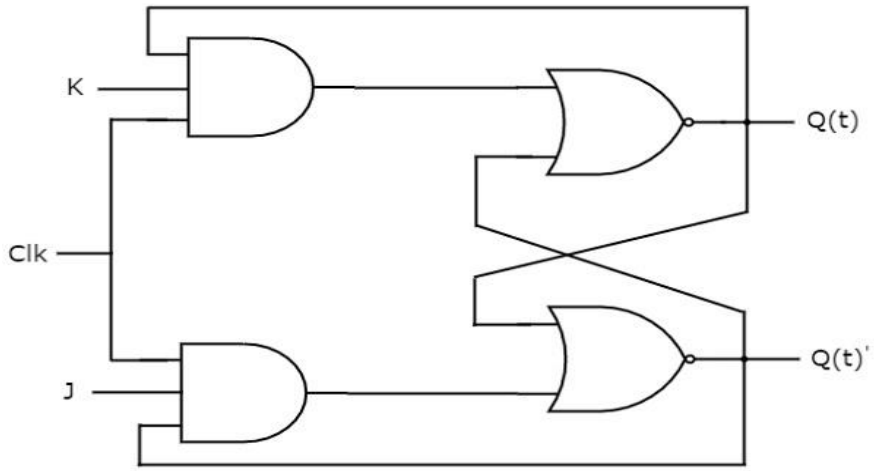
QN	T	QN+1
0	0	0
0	1	1
1	0	1
1	1	0

The logic symbol of the T flip-flop is shown in the figure.



JK Flip-Flop *(refer website table)*

JK flip-flop is the modified version of SR flip-flop. It operates with only positive clock transitions or negative clock transitions. The circuit diagram of JK flip-flop is shown in the following figure.



This circuit has two inputs J & K and two outputs Qtt & Qtt'. The operation of JK flip-flop is similar to SR flip-flop. Here, we considered the inputs of SR flip-flop as S = J Qtt' and R = KQtt in order to utilize the modified SR flip-flop for 4 combinations of inputs.

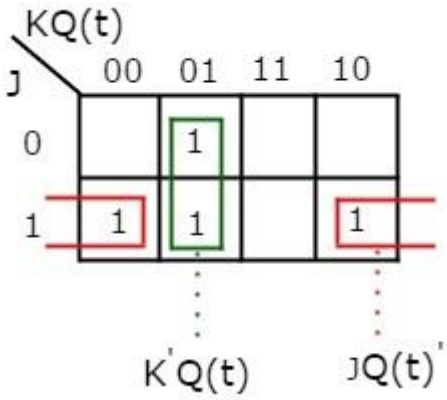
The following table shows the state table of JK flip-flop.

J	K	Qt+1t+1
0	0	Qtt
0	1	0
1	0	1
1	1	Qtt'

Here, Qtt & Qt+1t+1 are present state & next state respectively. So, JK flip-flop can be used for one of these four functions such as Hold, Reset, Set & Complement of present state based on the input conditions, when positive transition of clock signal is applied. The following table shows the characteristic table of JK flip-flop.

Present Inputs		Present State	Next State
J	K	Qtt	Qt+1t+1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

By using three variable K-Map, we can get the simplified expression for next state, Qt+1t+1. Three variable K-Map for next state, Qt+1t+1 is shown in the following figure.



The maximum possible groupings of adjacent ones are already shown in the figure. Therefore, the simplified expression for next state Qt+1t+1 is

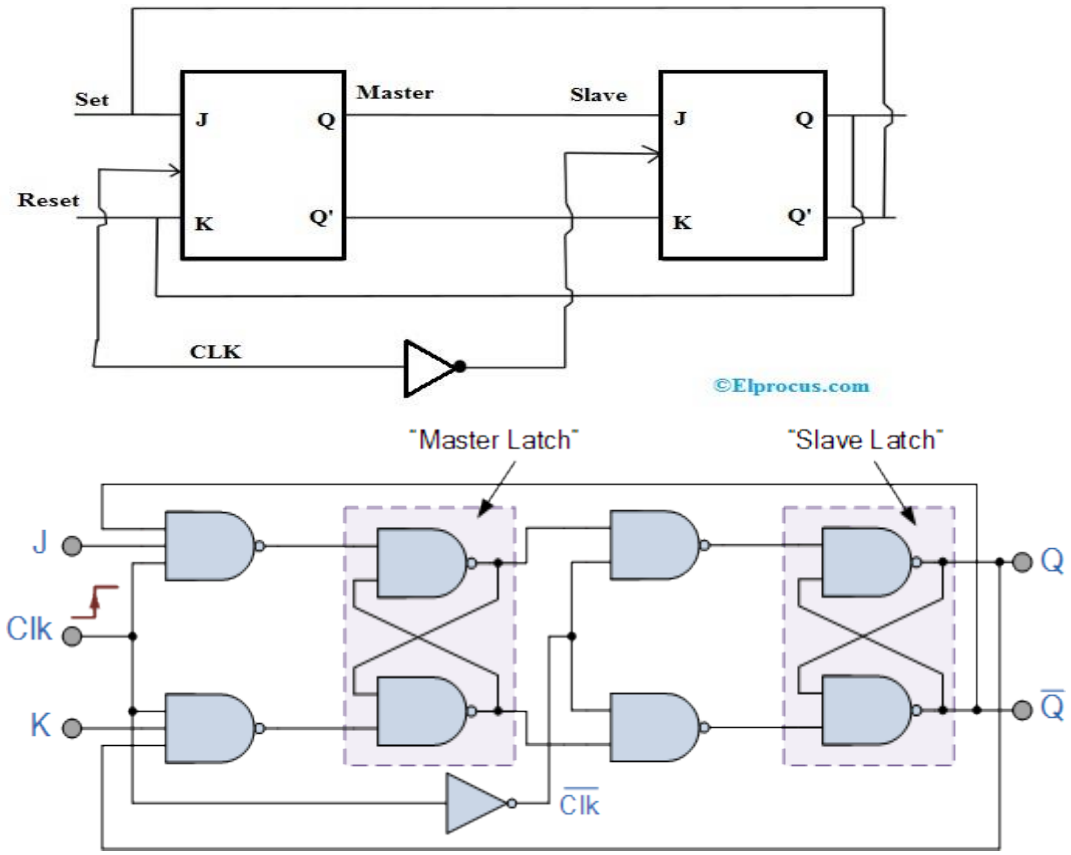
Q(t+1)=JQ(t)'+K'Q(t)

Master-Slave Flip-Flop

What is a Master-Slave Flip Flop?

Basically, this type of flip flop can be designed with two JK FFs by connecting in series. One of these FFs, one FF works as the master as well as other FF works as a slave. The connection of these FFs can be done like this, the master FF output can be connected to the inputs of the slave FF. Here slave FF's outputs can be connected to the inputs of the master FF.

In this type of FF, an inverter is also used addition to two FFs. The inverter connection can be done in such a way that where the inverted CLK pulse can be connected to the slave FF. In other terms, if CLK pulse is 0 for a master FF, then CLK pulse will be 1 for a slave FF. Similarly, when CLK pulse is 1 for master FF, then CLK pulse will be 0 for slave FF.



(Working)

Triggering

The state of a flip-flop is changed by a momentary change in the input signal. This change is called a trigger and the transition it causes is said to trigger the flip-flop.

Propagation Delay

The amount of time it takes for the output of the first Flip-Flop to travel to the input of the second Flip-Flop is the Propagation Delay. The further apart those two Flip-Flops are or the more combinational logic in the middle, the longer the propagation delay between the two of them.

Setup and Hold time

Setup time is the amount of time required for the input to a Flip-Flop to be stable before a clock edge. Hold time is similar to setup time, but it deals with events after a clock edge occurs. Hold time is the minimum amount of time required for the input to a Flip-Flop to be stable after a clock edge.

RAM and ROM

RAM, which stands for random access memory, and ROM, which stands for read-only memory, are both present in your computer.

RAM is volatile memory that temporarily stores the files you are working on. ROM is non-volatile memory that permanently stores instructions for your computer.

There are two main types of RAM: Dynamic RAM (DRAM) and Static RAM (SRAM).

- DRAM (pronounced DEE-RAM), is widely used as a computer's main memory. Each DRAM memory cell is made up of a transistor and a capacitor within an integrated circuit, and a data bit is stored in the capacitor. Since transistors always leak a small amount, the capacitors will slowly discharge, causing information stored in it to drain; hence, DRAM has to be refreshed (given a new electronic charge) every few milliseconds to retain data.
- SRAM (pronounced ES-RAM) is made up of four to six transistors. It keeps data in the memory as long as power is supplied to the system unlike DRAM, which has to be refreshed periodically. As such, SRAM is faster but also more expensive, making DRAM the more prevalent memory in computer systems.

(Falls under types of ROM)

EPROM: Erasable Programmable ROM chips allow you to write and rewrite them many times. These chips feature a quartz window through which a specialized EPROM programmer emits a specific frequency of ultraviolet light. This light burns out all the tiny charges in the EPROM to reopen its circuits. This exposure effectively renders the chip blank again, after which you can reprogram it according to the same process as a PROM. EPROM chips will eventually wear out, but they frequently have lifetimes of over 1000 erasures.

EEPROM: To modify an Electrically Erasable Programmable ROM chip, apply localized electrical fields to erase and rewrite the data. EEPROMs have several advantages over other types of ROM. Unlike the earlier forms, you can rewrite EEPROM without dedicated equipment, without removing it from the hardware, and in specifically designated increments. You don't have to erase and rewrite everything to make a single edit.

Volatile and non-volatile memory

RAM is volatile memory, which means that the information temporarily stored in the module is erased when you restart or shut down your computer. Because the information is stored electrically on transistors, when there is no electric current, the data disappears. Each time you request a file or information, it is retrieved either from the computer's storage disk or the internet. The data is stored in RAM, so each time you switch from one program or page to another, the information is instantly available. When the computer is shut down, the memory is cleared until the process begins again. Volatile memory can be changed, upgraded, or expanded easily by users.

ROM is non-volatile memory, which means the information is permanently stored on the chip. The memory does not depend on an electric current to save data, instead, data is written to individual cells using binary code. Non-volatile memory is used for parts of the computer that do not change, such as the initial boot-up portion of the software, or the firmware instructions that make your printer run. Turning off the computer does not have any effect on ROM. Non-volatile memory cannot be changed by users.

Unit 4

Registers

Introduction:

To increase the storage capacity, we have to use a group of Flip-Flop. This group of Flip-flop is known as Registers.

A processor register (CPU register) is one of a small set of data holding places that are part of the computer processor.

A register may hold an instruction, a storage address, or any kind of data (such as a bit sequence or individual characters). Some instructions specify registers as part of the instruction. For example, an instruction may specify

that the contents of two defined registers be added together and then placed in a specified register.

A register must be large enough to hold an instruction - for example, in a 64-bit computer, a register must be 64 bits in length.

Modes of Operation

(Note: Serial = One bit at a time, Parallel = All bits at a time)

SISO

There are four basic modes of operation based on the movement of data in the Registers and they are:

- Serial In – Serial Out (SISO) Mode
- Serial In – Parallel Out (SIPO) Mode
- Parallel In – Serial Out (PISO) Mode
- Parallel In – Parallel Out (PIPO) Mode
- Serial in – Serial Out (SISO) Mode

SISO mode accepts data serially under clock control i.e. the data transmitted is one bit at a time in either left or right direction. The stored information is produced as its output. Fig. 2 below shows a SISO mode of Shift Register consisting of 4 D-Type Flip-Flops (FF0, FF1, FF2 and FF3). They are connected serially with the same clock (CLK) signal applied to each Flip-Flop.

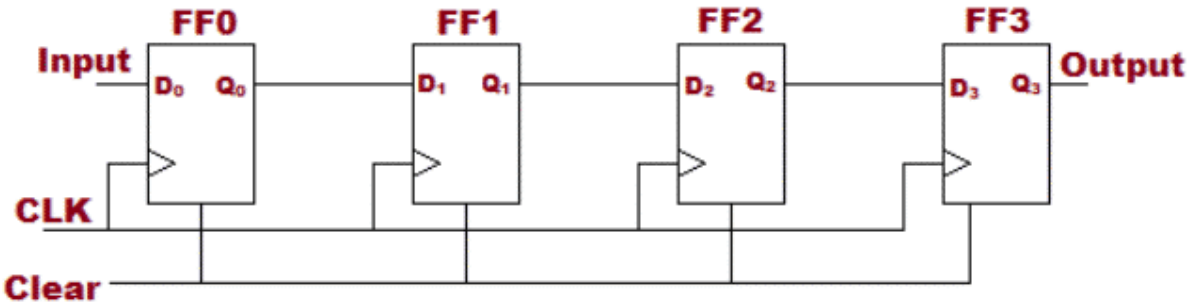


Fig. 2 – Schematic of Serial In – Serial Out (SISO) Mode

Serial In – Parallel Out (SIPO) Mode

The SIPO mode of Shift Registers accepts data serially i.e. one bit at a time through a single data line and produces a parallel output. Fig.3 shows a SIPO mode consisting of 4 D-Type Flip-Flop's (FF0, FF1, FF2 and FF3). In addition to the CLK Signal, Clear (CLR) signal is also connected to all the Flip-Flops to 'RESET' them.

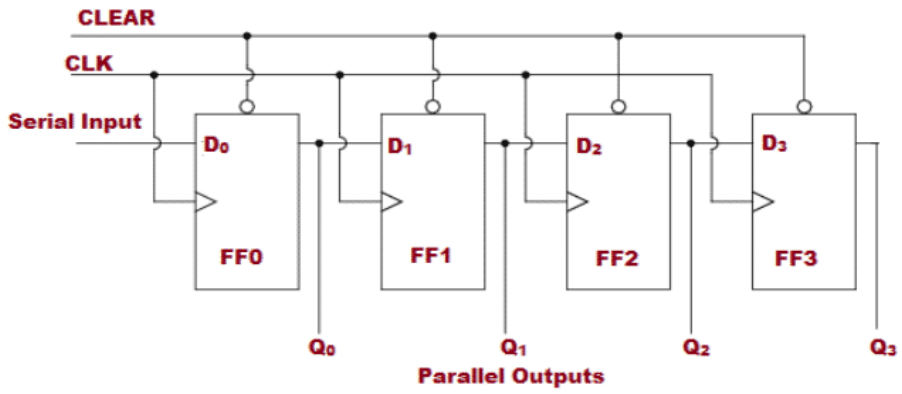


Fig. 3 – Schematic of Serial In – Parallel Out (SIPO) Mode

Parallel In – Serial Out (PISO) Mode

PISO mode of Shift Registers allows parallel data input i.e. data is fed separately to each input of Flip Flop and produces a required serial output. A Multiplexer is connected at the input of each Flip-Flop. The previous output and the parallel data input are connected to the input of the Multiplexer and output of the Mux is connected to the next Flip-Flop. Since the same CLK signal is applied, all the Flip-Flops are synchronous with each other.

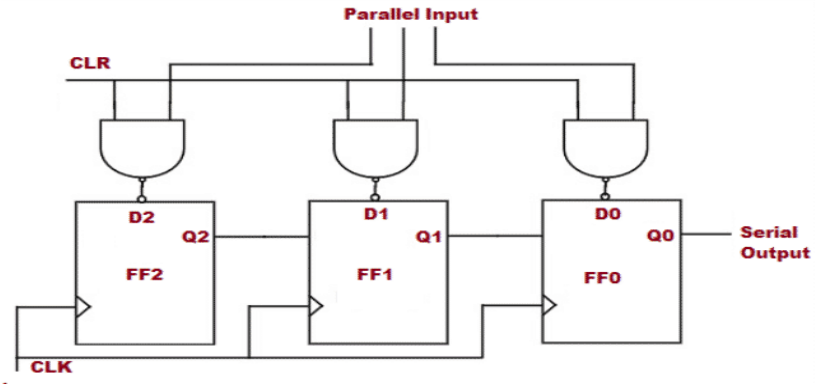


Fig. 4 – Schematic of Parallel In – Serial Out (PISO) Mode

Parallel In – Parallel Out (PIPO) Mode

In PIPO mode of Shift Registers, there is no serial shifting of the data and hence the Flip-Flops are not interconnected. The input and output to each Flip-Flop is separate. All the 4 Flip-Flops are connected to the same Clock (CLK) and Clear (CLR) signal.

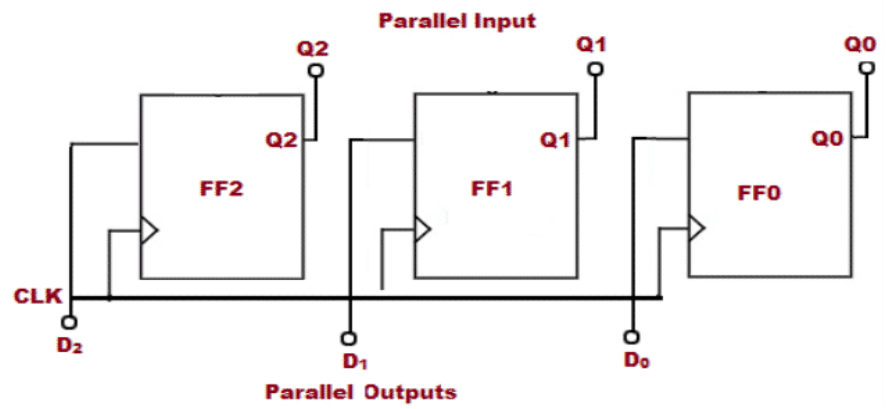


Fig. 5 – Schematic of Parallel In – Parallel Out (PIPO) Mode

Counters

Introduction

According to Wikipedia, in digital logic and computing, a Counter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal. Counters are used in digital electronics for counting purpose, they can count specific event happening in the circuit. For example, in UP counter a counter increases count for every rising edge of clock.

Asynchronous Counter

(here)

Synchronous Counter

The synchronous counter is a type of counter in which the clock signal is simultaneously provided to each flip-flop present in the counter circuit. More specifically, we can say that each flip-flop is triggered in synchronism with the clock input.

Unlike asynchronous counter where separate clock pulses are used to trigger the flip-flop, all the flip-flops in synchronous counters are triggered using a single clock pulse.

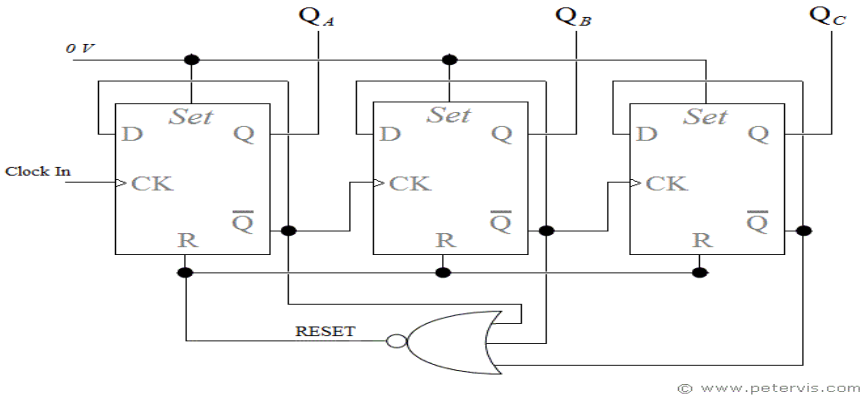
(Here)

Ripple Counter

Ripple counter is a special type of Asynchronous counter in which the clock pulse ripples through the circuit. The n-MOD ripple counter forms by combining n number of flip-flops. The n-MOD ripple counter can count 2n states, and then the counter resets to its initial value.

MOD-7 Ripple Counter

A modulo 7 (MOD-7) counter circuit, known as divide-by-7 counter, can be made using three D-type flip-flops. The circuit design is such that the counter counts from 0 to 6, and on the count of seven, it automatically resets to begin the count again. On the seventh count, all the Q outputs will be 1's, and their complementary outputs (/Q) will be 0's. Hence, we feed the inputs of the NOR gate with complimentary outputs.



The trick is to start with a MOD-8 counter and then look for the binary sequence 000, which is when the output from the NOR gate is HIGH. This output is then used to control the RESET function on all three flip-flops.

Decade Counter

A binary coded decimal (BCD) is a serial digital counter that counts ten digits. It resets for every new clock input. As it can go through 10 unique combinations of output, it is also called as “Decade counter”. A BCD counter can count 0000, 0001, 0010, 1000, 1001, 1010, 1011, 1110, 1111, 0000, and 0001 and so on.

Refer this

4-Bit Down Counter

A 4-bit down counter is a digital counter circuit, which provides a binary countdown from binary 1111 to 0000. This circuit uses four D-type flip-flops, which are positive edge triggered. At each stage, the flip-flop feeds its inverted output (/Q) back into its own data input (D). However, it feeds its non-inverted output (Q) to the clock input (CK) of the following stage. This type of circuit operates in an asynchronous (ripple) manner because the flip-flop stages do not rely on a common clock pulse for timing. The operational speed of the counter depends upon the signal propagation through successive stages, rather than a common clock pulse as in synchronous circuits.

When RESET is applied on all the flip-flops simultaneously, their Q outputs become logic 0 state, and /Q outputs become logic 1 state. The /Q outputs are fed back into their own data inputs (D) and therefore all that they require is the rising edge of a clock pulse to transfer the data at input (D) to output Q.

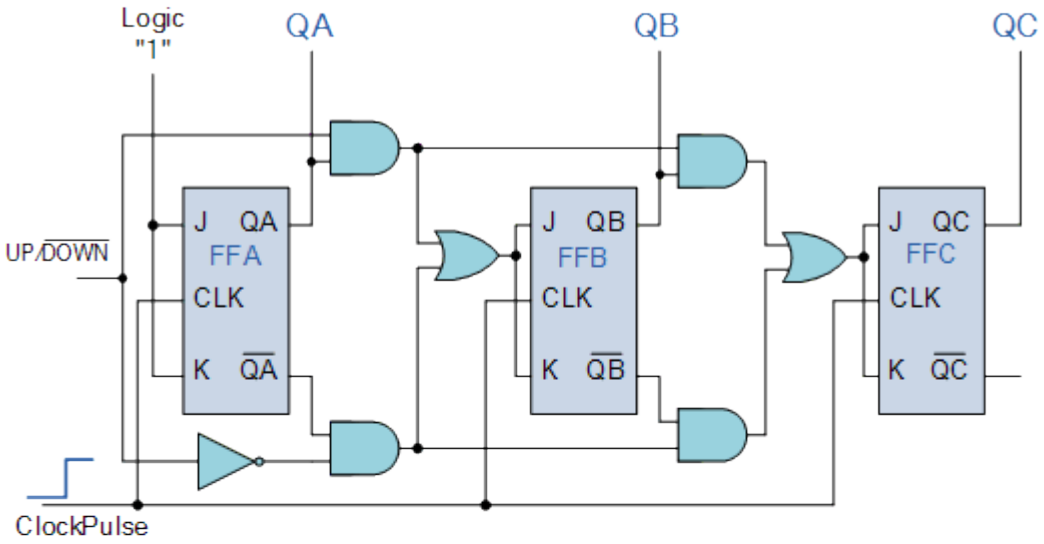
Truth Table

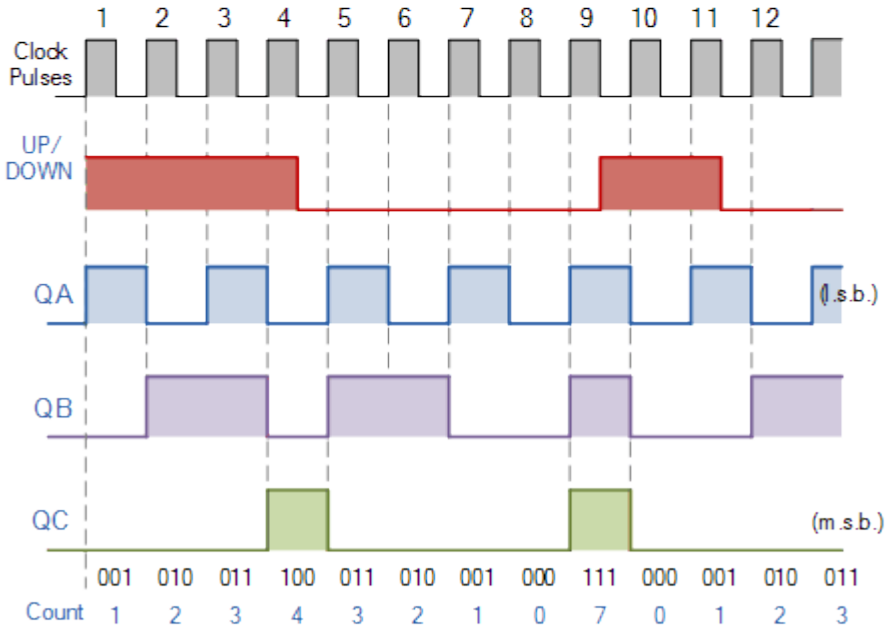
Decimal	QD	QC	QB	QA
15	1	1	1	1
14	1	1	1	0
13	1	1	0	1
12	1	1	0	0
11	1	0	1	1
10	1	0	1	0
09	1	0	0	1
08	1	0	0	0
07	0	1	1	1
06	0	1	1	0
05	0	1	0	1
04	0	1	0	0
03	0	0	1	1
02	0	0	1	0
01	0	0	0	1
00	0	0	0	0

This type of circuit is useful in timer applications. A count down from decimal 15 to decimal 11 becomes simply a matter of looking for the 1-0-1-1 output, because at the decimal count of 11, the logic state of QA=1, QB=1, QC=0, and QD=1. In TTL digital circuits, logic 1 state, usually represents +5 V electrically, and we can use these outputs in many circuits for timer applications.

Up/down counter

Synchronous 3-bit Up/Down Counter





The circuit above is of a simple 3-bit Up/Down synchronous counter using JK flip-flops configured to operate as toggle or T-type flip-flops giving a maximum count of zero (000) to seven (111) and back to zero again. Then the 3-Bit counter advances upward in sequence (0,1,2,3,4,5,6,7) or downwards in reverse sequence (7,6,5,4,3,2,1,0).

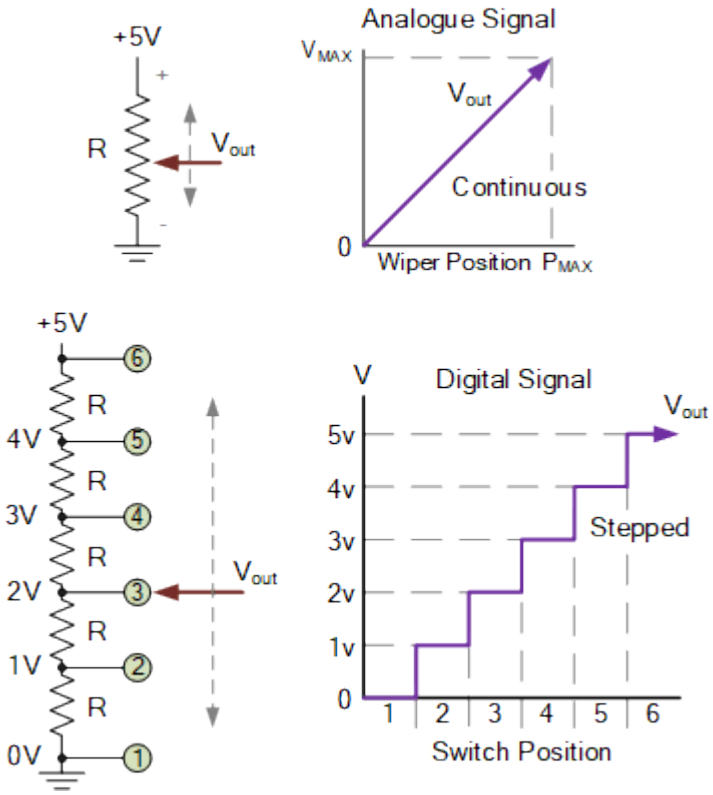
Generally, most bidirectional counter chips can be made to change their count direction either up or down at any point within their counting sequence. This is achieved by using an additional input pin which determines the direction of the count, either Up or Down and the timing diagram gives an example of the counter operation as this Up/Down input changes state.

Analog to Digital Converter

Analogue-to-Digital Converters, (ADCs) allow micro-processor controlled circuits, Arduinos, Raspberry Pi, and other such digital logic circuits to communicate with the real world. In the real world, analogue signals have continuously changing values which come from various sources and sensors which can measure sound, light, temperature or movement, and many digital systems interact with their environment by measuring the analogue signals from such transducers.

While analogue signals can be continuous and provide an infinite number different voltage values, digital circuits on the other hand work with binary signal which have only two discrete states, a logic "1" (HIGH) or a logic "0" (LOW). So, it is necessary to have an electronic circuit which can convert between the two different domains of continuously changing analogue signals and discrete digital signals, and this is where Analogue-to-Digital Converters (A/D) come in.

Analogue and Digital Signals

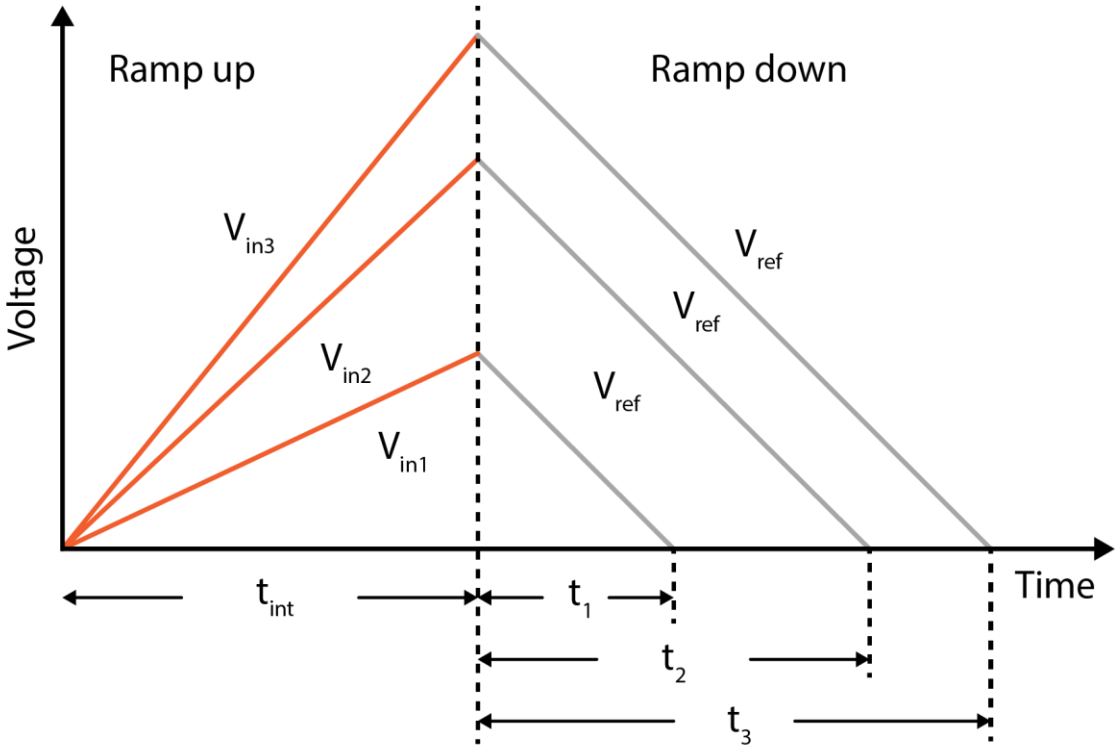


Dual Slope A/D Converters

Dual slope ADCs are accurate but not terribly fast. The principle way they convert analog to digital values is by using an integrator. The voltage is input and allowed to "run up" for a period of time. Then a known voltage of the

opposite polarity is applied and allowed to run back down to zero. When it reaches zero, the system calculates what the input voltage had been by comparing the run-up time with the run-down time, and by knowing what the reference had been. The run-up and run-down times are the two slopes for which this technique has been named.

This iterative process is reliable, but it takes time, and there is always a trade-off between resolution and speed because unlike SAR or delta-sigma ADCs, they cannot achieve both. As a result, Dual Slope aka "integrating ADCs" are used in applications like handheld multimeters and are not found in DAQ applications.



Typical Integrating Amplifier, showing the comparator, timer, and controller

Pros

Very precise and accurate measurements

Cons

Slow conversion time due to the ramp-up and ramp-down iteration

Applications

Applications for dual slope ADCs include handheld and benchtop multi-meters.

Written by Grant Maloy Smith, the data acquisition expert

In this article we will review the major types of A/D converters (ADCs) in use today, describing each with enough detail that you will:

- **See** the basic technology of each type of ADC
- **Learn** about the key ADC features and capabilities
- **Understand** which ADC types work best for today's applications
- **Find** out which two major ADC types Dewesoft has selected, and why

Are you ready to get started? Let's go!

Jump to a section

Introduction

The Analog-to-Digital Converter (ADC) is one of the fundamental building blocks of modern data acquisition systems (aka DAQ or DAS systems). The main purpose of the A/D converters within a data acquisition system is to convert conditioned analog signals

into a stream of digital data so that the data acquisition system can process them for display, storage, and analysis.

Main Types of ADC Converters

There are really five major types of ADCs in use today:

- Successive Approximation (SAR) ADC
- Delta-sigma (ΔΣ) ADC
- Dual Slope ADC
- Pipelined ADC
- Flash ADC

Skip down to the Main ADC technologies to see details about each of the types.

Key ADC Features and Capabilities

Every technology has features and capabilities that drive its use in the market. With ADCs these include:

- **Sample rate** - how fast can an ADC convert analog to digital?
- **Bit resolution** - with how much precision can an ADC convert analog to digital?

Let's look at each of these fundamental specifications in more detail:

What is the Sample Rate?

The rate at which the signals are converted from the analog domain to a stream of digital data is called the sample rate or sampling frequency. There is no right or wrong here, it simply depends on the application. For example, barometric pressure changes very slowly over a period of minutes or hours, so you really don't need to

sample it more than once per second. On the other hand, if you're trying to measure a RADAR signature, you need to sample hundreds of millions of times per second, or perhaps even into the billions of samples per second.

In the world of data acquisition, we measure AC voltages and currents, shock and vibration, temperature, strain, pressure, and the like. These signals and sensors require sample rates in the range of DC to 200,000 samples per second (200 kS/s) on average, while a few applications require sampling up to 1,000,000 samples per second (1 MS/s).

The sample rate is usually referred to as the T (time) or X-axis of measurement.

Why Is The Sample Rate Important?

Understanding your signals and their highest possible frequencies is an important part of getting accurate measurements. For example, let's say that we want to measure the output of an accelerometer.

If we expect it to experience vibrations with a maximum frequency of 100 Hz, we must set the sample rate to at least double that (the Nyquist frequency), but in practice ten times oversampling is better in order to get a good quality representation of the signal shape. So in this example, we set the sample rate to 1000 Hz and do the measurement.

Theoretically, everything should be fine, but how do we know that the signal didn't really go much higher in frequency at a considerable amplitude? If it did, then our system would not accurately measure or convert the signal. And, in fact, if this is taken to an extreme, the measured values could even be completely wrong.

To understand aliasing, watch an old movie where a camera was filming at 24 frames per second as a wagon rolled by - at various speeds it can look like the wheels are spinning backward, or even not moving at all.

This is a kind of stroboscopic visual effect caused by the harmonic relationship between the rotational frequency of the wheel versus the picture-taking rate of the camera. Perhaps you've seen videos where a camera's shutter speed was synchronized with a helicopter's blades, where it appears that the helicopter is hanging in the air, it's blades not moving at all.

In the case of a movie or an entertaining video it does not matter, but when making a scientific measurement, if we really believe that the wheels of a car are spinning backward, or that a helicopter's blades are not moving, when in fact they are going quite fast, we have a real-world measurement problem.

In terms of digitizing voltage signals with our ADC, it is important that the sample rate is set appropriately. If we set it too high, we waste processing power and end up with data files that are unnecessarily large and hard to analyze. But if we set it too low, we could have two problems:

1. Missing vital dynamic signal components
2. Ending up with false ("alias") signals (if the system lacks anti-aliasing filtering)

Demonstration of a false signal (alias) in black, caused by sampling too infrequently compared to the original signal.
[Graphic is in the public domain]

Sample Rate Best Practices

At this point, you might think to simply sample much faster than the signal could possibly reach, even orders of magnitude faster. Wouldn't that solve the undersampling problem? Yes, but it would create a new problem: drastically increasing the amount of data

recorded creates a data handling, storage, and analysis problem. And it may not even be possible to sample that fast with your system.

Luckily, there is a better way to avoid aliasing without overloading ourselves with vast amounts of mostly redundant data: anti-aliasing filtering.

Anti-Aliasing Filtering (AAF)

If we filter in the analog domain before the ADC, we can prevent the aliasing problem from ever occurring. Note that it is still important to set a high enough sample rate to capture the frequency range of interest, but at least with Anti-Aliasing Filters (AAF), we will avoid false signals from destroying the integrity of our measurements. The ideal AAF would have a very flat passband AND very sharp cutoff at the Nyquist frequency (essentially half of the sample rate).

Anti-aliasing filter roll-off diagram

Typical AAF configuration: a steep low-pass analog filter before the ADC prevents signals more than half of the maximum bandwidth of the ADC from passing. This is what Dewesoft does with its 16-bit SAR ADCs as found in SIRIUS-HS modules.

However, with their 24-bit Delta-sigma ADCs, Dewesoft systems have an additional DSP filter on the ADC itself that automatically adjusts based on the sample rate that the user has selected. This multi-stage approach provides the most robust anti-aliasing filtering available in DAQ systems today.

What is Bit Resolution and Why Does It Matter?

While the sample rate as discussed in the previous section involves the time (T or X) axis of our digital data stream, bit resolution, or a number of bits involves the amplitude (Y) axis.

In the early days of data acquisition, 8-bit ADCs were common. As of this writing, in the world of DAQ systems, 24-bit ADCs are standard among most data acquisition systems designed to make dynamic measurements, and 16-bit ADCs are commonly considered the minimum resolution for signals in general. There are some low-end systems utilizing 12-bit ADCs.

Because each bit of resolution effectively doubles the possible resolution, systems with 24-bit ADCs provide 2^24 = 16,777,216. Thus, an incoming one-volt signal can be divided into more than 16 million steps on the Y-axis.

16,777,216 steps for a 24-bit ADC is dramatically better than the maximum theoretical 65,656 steps of a 16-bit ADC. Thus the appearance of waveshapes is accordingly more accurate and has a lot more precision, the more resolution you have. This applies to the time axis, too.

24-bit resolution (orange) vs. 16-bit resolution (gray)

DualCoreADC® Technology and Why It Matters

On the amplitude axis, one challenge that engineers have faced for years is the dynamic range. For example: what if we have a signal that is usually less than 5 volts, but at times can range upward dramatically? If we set the resolution of the ADC to accommodate the 0-5V data, the system will be totally overloaded when the signal rises past that.

One solution would be to use two channels set to different gains and refer to one of them for the 0-5V data, and to the other one for the higher amplitude data. But this is very inefficient - we can't possibly use two channels for every input signal - we would need twice as many DAQ systems in order to do the same work. In addition, it would make data analysis after each test much more complex and time-consuming.

Dewesoft's **DualCoreADC®** technology solves this problem by using two separate 24-bit ADCs per channel, and automatically switching between them in real-time and creating a single, seamless channel. These two ADCs always measure the high and low gain of the input signal. This results in the full possible measuring range of the sensor and prevents the signal from being clipped.
With DualCoreADC® technology SIRIUS achieves more than 130 dB signal to noise ratio and more than 160 dB in dynamic range. This is 20 times better than typical 24-bit systems with 20 times less noise.

Multiplexed vs. Single ADC per Channel

Very often in lower-end DAQ systems, such as data loggers or industrial control systems, multiplexed A/D cards are used, because they are less expensive than A/D cards which have a separate ADC chip per input channel.
In a multiplexed ADC system, a single analog-to-digital converter is used to convert multiple signals from analog to digital domain. This is done by multiplexing the analog signals one at a time into the ADC.

This is a lower-cost approach, but it is not possible to precisely align the signals on the time axis, because only one signal can ever be converted at a time. Therefore, there is always a time skew between channels. If a small-time skew error is irrelevant in a given application, then it is not necessarily a bad thing. The same goes for the analog devices used within the system - choosing the best

fit for the application in terms of form, fit, function, and avoiding obsolescence are driving factors.

In addition, since the maximum sample rate is always divided by the number of channels being sampled, the top sample rate per channel is usually lower in multiplexed systems, except in cases where only one or a few channels are being sampled.

In today's data acquisition systems, multiplexed ADC systems are employed primarily by low-end systems, where cost is more important than precision or speed.

Five Main ADC Technologies

There are five major types of ADCs in use today. Each has its place, based on its essential characters of bit resolution and sample rate. Let's look at each of these types, see how they work, and how they are used in the world today.

Comparison of major ADC Types

ADC Type	Pros	Cons	Max Resolution	Max Sample Rate	
Successive Approximation (SAR)	Good speed/resolution ratio	No inherent anti-aliasing protection	18 bits	10 MHz	
Delta-sigma (ΔΣ)	High dynamic performance, inherent anti-aliasing protection	Hysteresis on unnatural signals	32 bits	1 MHz	
Dual Slope	Accurate, inexpensive	Low speed	20 bits	100 Hz	
Pipelined	Very fast	Limited resolution	16 bits	1 GHz	

ADC Type	Pros	Cons	Max Resolution	Max Sample Rate	
Flash	Fastest	Low bit resolution	12 bits	10 GHz	

Each has its own advantages and disadvantages and thus suitability for certain applications. Let's look at each of them:

Successive Approximation ADCs (SAR)

The "bread and butter" ADC of the DAQ world is the SAR analog-to-digital converter (Successive Approximation Register). It offers an excellent balance of speed and resolution and handles a wide variety of signals with excellent fidelity.
It's been around for a long time, therefore SAR designs are stable and reliable, and the chips are relatively inexpensive. They can be configured for both low-end A/D cards, where a single ADC chip is "shared" by multiple input channels (multiplexed A/D boards), or in configurations where each input channel has its own ADC for true simultaneous sampling.

Typical SAR block diagram

The analog input of most ADCs is 5V, which is why nearly all signal conditioning front-ends provide a conditioned output that is the same. The typical SAR ADC uses a sample-and-hold circuit that takes in the conditioned analog voltage from the signal conditioning front-end.

An on-board DAC creates an analog reference voltage equal to the digital code output of the sample and holds a circuit. Both of these are fed into a comparator which sends the result of the comparison to the SAR. This process continues for "n" successive times, with "n" being the bit resolution of the ADC itself, until the closest value to the actual signal is found.

SAR ADCs do not have any inherent anti-aliasing filtering (AAF), so unless this is added before the ADC by the DAQ system, if the engineer selects too low of a sample rate, false signals (aka “aliases”) will be digitized by the SAR ADC. Aliasing is particularly problematic because it is impossible to correct it after digitization. There is no way to fix it with software. It must be prevented either by always sampling faster than the Nyquist frequency of all input signals or by filtering the signals before and within the ADC.

Pros

- Simple circuit with only one comparator needed
- Higher sample rates possible compared to delta-sigma ADCs
- Handles natural and unnatural waveshapes well

Cons

- Anti-aliasing filtering must be added externally
- Bit resolution and dynamic range limited compared to delta-sigma ADCs

Applications

Applications for SAR ADCs include DAQ systems, from low-end multiplexed ADC systems to higher speed single ADC per channel systems, industrial control and measurement, CMOS imaging.

Delta-sigma ADCs (ΔΣ)

A newer ADC design is the delta-sigma ADC (or delta converter), which takes advantage of DSP technology in order to improve amplitude axis resolution and reduce the high-frequency quantization noise inherent in SAR designs.
The complex and powerful design of delta-sigma ADCs makes them ideal for dynamic applications that require as much amplitude axis resolution as possible. This is why they are commonly found in audio, sound and vibration, and a wide range of high-end data acquisition applications. They are also used extensively in precision industrial measurement applications.

Typical Delta-Sigma ADC block diagram

A low-pass filter implemented in a DSP eliminates virtually quantization noise, resulting in excellent signal-to-noise performance.

Delta-sigma ADCs work by over-sampling the signals far higher than the selected sample rate. The DSP then creates a high-resolution data stream from this over-sampled data at the rate that the user has selected. This over-sampling can be up to hundreds of times higher than the selected sample rate. This approach creates a very high-resolution data stream (24-bits is common) and has the advantage of allowing multistage anti-aliasing filtering (AAF), making it virtually impossible to digitize false signals. However, it does impose a kind of speed limit, so delta-sigma ADCs are typically not as fast as SAR ADCs, for example.

Pros

- High-resolution output (24-bits)
- Over-sampling reduces quantization noise
- Inherent Anti-aliasing filtering

Cons

- Limited to around 200 kS/s sample rate
- Do not handle unnatural shape waveforms as well as SAR

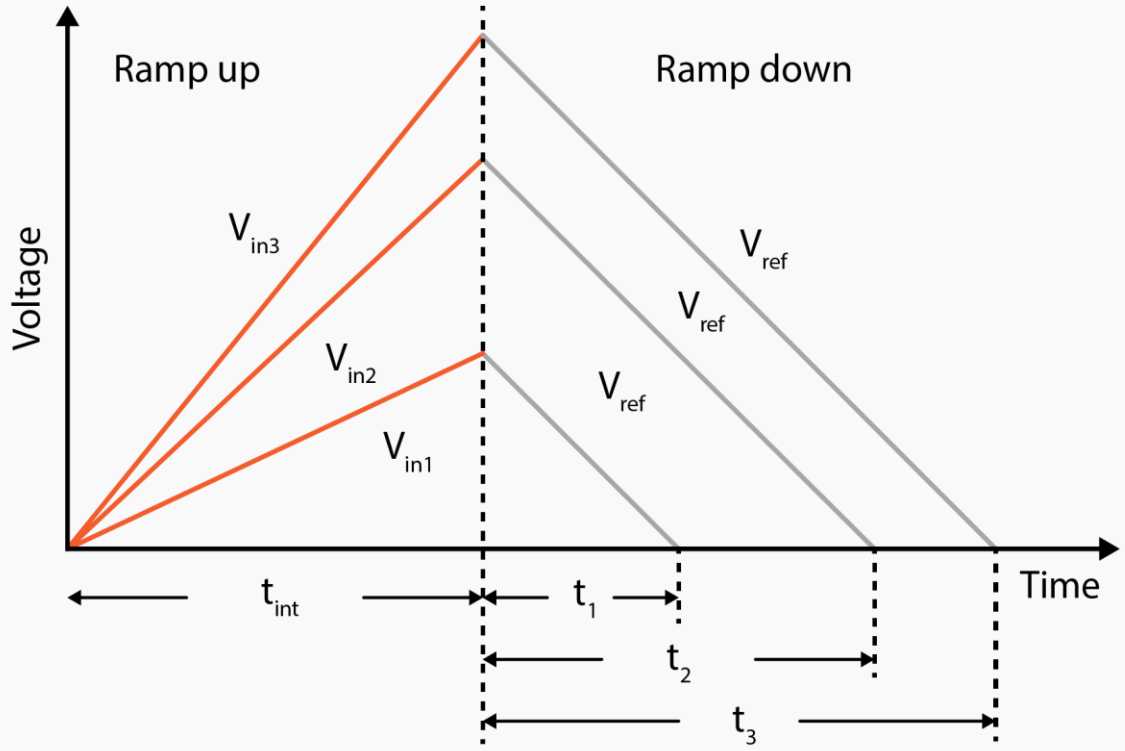
Applications

Applications for Delta-sigma ADCs include data acquisition, especially noise and vibration, industrial balancing, torsional and rotational vibration, power quality monitoring, precision industrial measurements, audio and voiceband, communications.

Dual Slope A/D Converters

Dual slope ADCs are accurate but not terribly fast. The principle way they convert analog to digital values is by using an integrator. The voltage is input and allowed to “run up” for a period of time. Then a known voltage of the opposite polarity is applied and

allowed to run back down to zero. When it reaches zero, the system calculates what the input voltage had been by comparing the run-up time with the run-down time, and by knowing what the reference had been. The run-up and run-down times are the two slopes for which this technique has been named.
This iterative process is reliable, but it takes time, and there is always a trade-off between resolution and speed because unlike SAR or delta-sigma ADCs, they cannot achieve both. As a result, Dual Slope aka “integrating ADCs” are used in applications like handheld multimeters and are not found in DAQ applications.



Typical Integrating Amplifier, showing the comparator, timer, and controller

Pros

- Very precise and accurate measurements

Cons

- Slow conversion time due to the ramp-up and ramp-down iteration

Applications

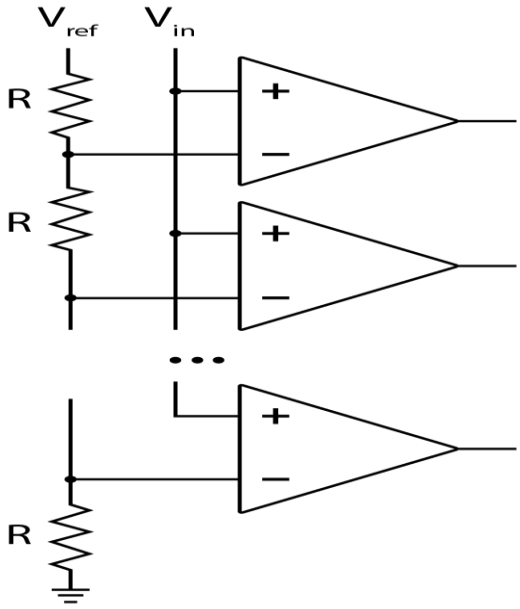
Applications for dual slope ADCs include handheld and benchtop multimeters.

Flash A/D Converters/Parallel Comparator

Flash ADCs are fast and operate virtually without latency, which is why they are the architecture of choice when the highest possible sample rates are needed. They convert analog to a digital signal by comparing it with known reference values. The more known references that are used in the conversion process, the more accuracy can be achieved. For example, if we want a Flash ADC with a 10-bit resolution, we would need to compare the incoming analog signal against 1024 known values. The 8-bit resolution would require 256 known values, and so on.

The more resolution we want, the bigger and more power-hungry the Flash ADC becomes - and the sample rate has to be reduced.

For that reason, the 8-bit resolution is generally the "sweet spot" for these ADCs. Flash ADCs can operate into the low GS/s and still provide an 8-bit resolution.



Flash ADC diagram

Pros

Fastest ADC type

Instant conversion without latency

Cons

Circuit gets bigger and more power-consuming with each bit

The resolution effectively limited to 8-bit

Applications

Applications for Flash ADCs include the fastest digital oscilloscopes, microwave measurements, fiber optics, RADAR detection, and wideband radio.

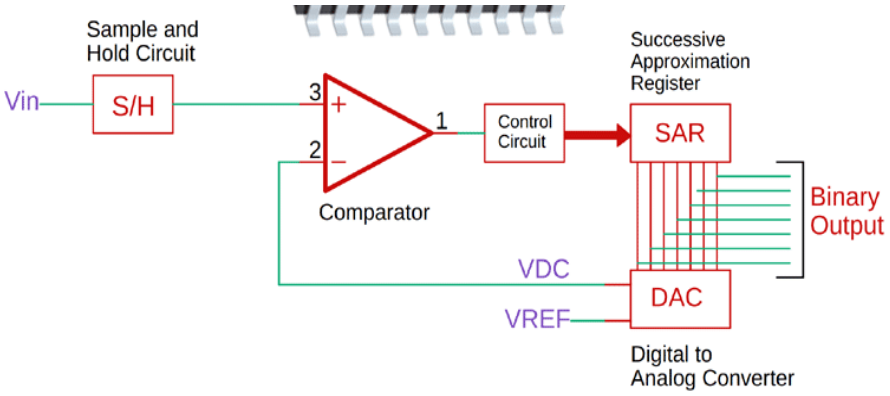
Successive Approximation method

Successive Approximation ADCs (SAR)

The "bread and butter" ADC of the DAQ world is the SAR analog-to-digital converter (Successive Approximation Register). It offers an excellent balance of speed and resolution and handles a wide variety of signals with excellent fidelity.

It's been around for a long time. Therefore, SAR designs are stable and reliable, and the chips are relatively inexpensive. They can be configured for both low-end A/D cards, where a single ADC chip is "shared" by multiple input channels (multiplexed A/D boards), or in configurations where each input channel has its own ADC for true simultaneous sampling.

Typical SAR block diagram



The analog input of most ADCs is 5V, which is why nearly all signal conditioning front-ends provide a conditioned output that is the same. The typical SAR ADC uses a sample-and-hold circuit that takes in the conditioned analog voltage from the signal conditioning front-end.

An on-board DAC creates an analog reference voltage equal to the digital code output of the sample and holds a circuit. Both of these are fed into a comparator which sends the result of the comparison to the SAR. This process continues for "n" successive times, with "n" being the bit resolution of the ADC itself, until the closest value to the actual signal is found.

SAR ADCs do not have any inherent anti-aliasing filtering (AAF), so unless this is added before the ADC by the DAQ system, if the engineer selects too low of a sample rate, false signals (aka "aliases") will be digitized by the SAR ADC. Aliasing is particularly problematic because it is impossible to correct it after digitization.

There is no way to fix it with software. It must be prevented either by always sampling faster than the Nyquist frequency of all input signals or by filtering the signals before and within the ADC.

Pros

Simple circuit with only one comparator needed

Higher sample rates possible compared to delta-sigma ADCs

Handles natural and unnatural waveshapes well

Cons

Anti-aliasing filtering must be added externally

Bit resolution and dynamic range limited compared to delta-sigma ADCs

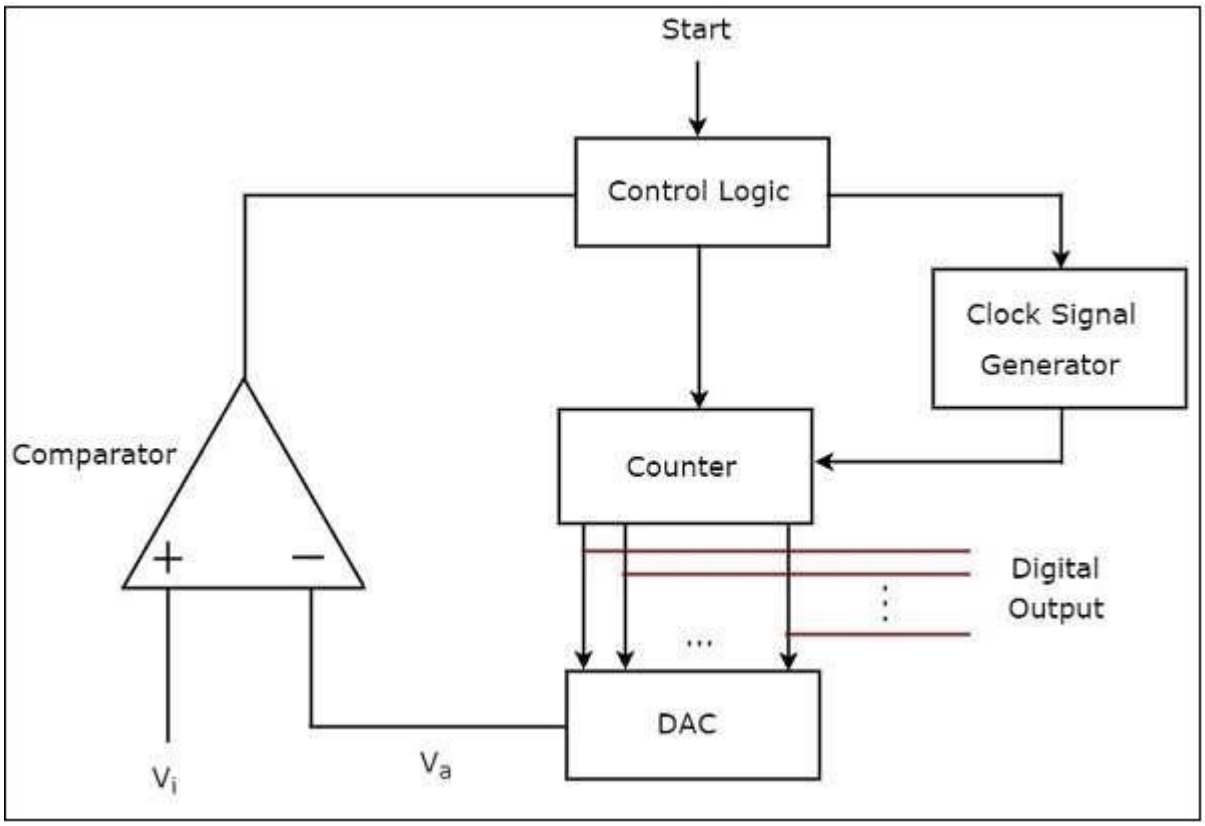
Applications

Applications for SAR ADCs include DAQ systems, from low-end multiplexed ADC systems to higher speed single ADC per channel systems, industrial control and measurement, CMOS imaging.

Counter Type ADC

A counter type ADC produces a digital output, which is approximately equal to the analog input by using counter operation internally.

The block diagram of a counter type ADC is shown in the following figure −



The counter type ADC mainly consists of 5 blocks: Clock signal generator, Counter, DAC, Comparator and Control logic.

The working of a counter type ADC is as follows −

The control logic resets the counter and enables the clock signal generator in order to send the clock pulses to the counter, when it received the start commanding signal.

The counter gets incremented by one for every clock pulse and its value will be in binary (digital) format. This output of the counter is applied as an input of DAC.

DAC converts the received binary (digital) input, which is the output of counter, into an analog output. Comparator compares this analog value,VaVa with the external analog input value Vi.

The output of comparator will be '1' as long as Vi is greater than. The operations mentioned in above two steps will be continued as long as the control logic receives '1' from the output of comparator.

The output of comparator will be '0' when Vi is less than or equal to Va. So, the control logic receives '0' from the output of comparator. Then, the control logic disables the clock signal generator so that it doesn't send any clock pulse to the counter.

At this instant, the output of the counter will be displayed as the digital output. It is almost equivalent to the corresponding external analog input value Vi.

Refer this