# PG-TRB (COMPUTER INSTRUCTOR)

## DIGITAL ELECTRONICS

**Fundamental Concepts: NAND, NOR, Exclusive-OR, Boolean Algebra**

- Basic logic circuits with one or more inputs and one output are known as gates
- Gates are used as the building blocks in the design of more complex digital logic circuits

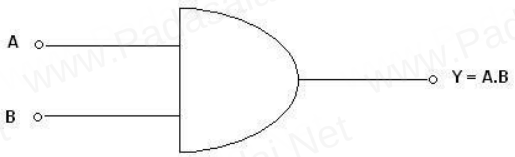
There are several ways of representing logic functions:

– Symbols to represent the gates
– Truth tables
– Boolean algebra

**AND gate**

A gate is simply an electronic circuit which operates a one or more signals to produce an output signal. The output is high only for certain combination of input signals.

An AND gate has a high output only when all inputs are high. The output is low when any one input is low.



Boolean expression for AND gate operation is

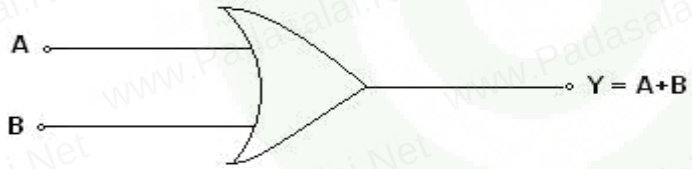
Y=A . B

**Truth table**

| A | B | Y = A . B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**OR gate**

An OR gate produces a high output when any or the entire inputs are high. The output is low only when all the inputs are low.



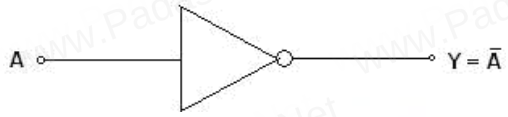

The Boolean expression for an OR gate is Y=A+B

**Truth table:**

| A | B | Y = A + B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**NOT gate:**

A NOT gate (Figure 1.3) is also called an inverter. The circuit has one input and one output. The output is the complement of the input. If the input signal is high, the output is low and vice versa.




The Boolean expression for NOT gate is $Y = \bar{A}$

**Truth table:**

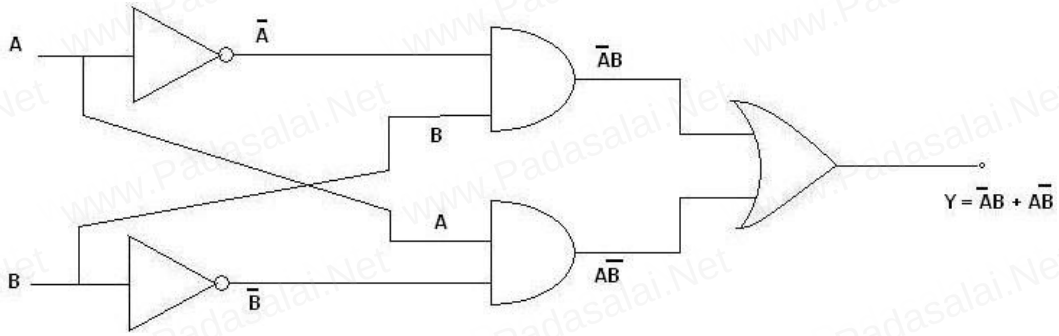
| A | Y = Ā |
|---|---|
| 0 | 1 |
| 1 | 0 |

If two NOT gates are cascaded then the output will be same as the input and the circuit is called buffer circuit.

**XOR gate**

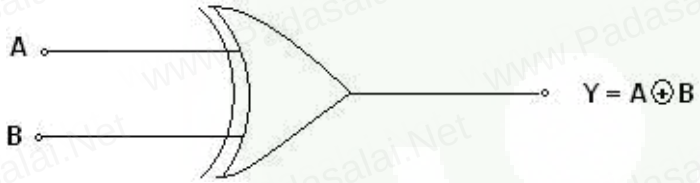
XOR (Figure 1.6) gate is an abbreviation of exclusive OR gate. It has two inputs and one output. For a two input XOR gate, the output is high when the inputs are different and the output is low when the inputs are same. In general, the output of an XOR gate is high when the number of its high inputs is odd. The Boolean expression of the XOR gate is

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




**Truth table**:

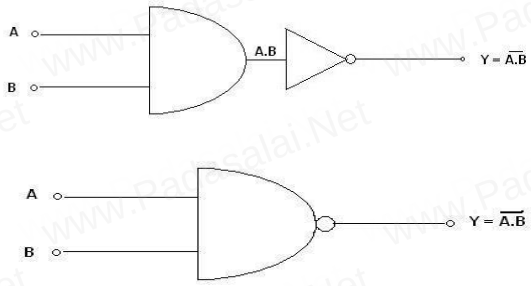
| A | B | Y = A ⊕ B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |




**Universal Gates**

**NAND gate**

A NAND gate has two or more input signals but only one output signal. All input signals must be high to get a low output. When one AND gate is combined with a NOT gate, a NAND gate is obtained.





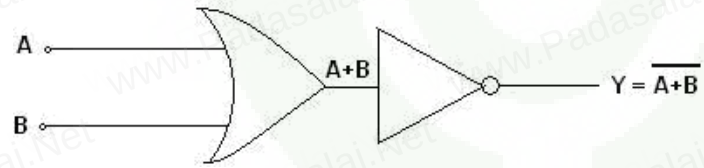


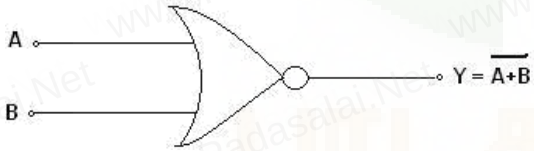
Truth table:

| A | B | Y = A . B |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**NOR gate**:

NOR gate (Fig. 1.5) has two or more input signals and one output signal. It consists of one OR gate followed by an inverter. A NOR gate produces a high output only when all the inputs are low.



| A | B | $Y = \overline{A + B}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

**Basic Laws of Boolean Algebra**

**Commutative law:**

A + B = B + A B + A = A + B

**Associative law:**

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

**Distributive law**

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

**Other laws of Boolean algebra:**

1. A + 0 = A

2. A + 1 = 1

3. A + A = A 4. A + Ā = 1 5. A .0

= 0

6. A .1 = A

7. A .A = A

8. A . Ā = 0

=

9. A = A

10. A + A.B =A

11. A.(A + B) = A

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

13. A + Ā.B =A + B

14. A.(Ā +B) = A.B

15. (A + B).(Ā + C) = A.C + Ā.B

16. (A + C).(Ā + B) = A.B + Ā.C

## De Morgan's Theorems:

**I Theorem statement:**

The complement of a sum is equal to the product of the complements.

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

**II Theorem Statement:**

The complement of a product is equal to the sum of the complements.

$\overline{A \cdot B} = \bar{A} + \bar{B}$

**Proof of first theorem:**

To prove $\overline{A + B} = \bar{A} \cdot \bar{B}$

**Case 1:** A=0, B=0

L.H.S => $\overline{A + B} = \overline{0 + 0} = \overline{0} = 1$

R.H.S => $\bar{A} \cdot \bar{B} = \bar{0} \cdot \bar{0} = 1 \cdot 1 = 1$

**Case 2:** A=0, B=1

L.H.S => $\overline{A + B} = \overline{0 + 1} = \overline{1} = 0$

R.H.S => $\bar{A} \cdot \bar{B} = \bar{0} \cdot \bar{1} = 1 \cdot 0 = 0$

**Case 3:** A=1, B=0

L.H.S => $\overline{A + B} = \overline{1 + 0} = \overline{1} = 0$

R.H.S => $\bar{A} \cdot \bar{B} = \bar{1} \cdot \bar{0} = 0 \cdot 1 = 0$

**Case 4:** A=1, B=1

L.H.S => A + B = 1 + 1 = 1 = 0

R.H.S => $\overline{A} \cdot \overline{B} = 1 \cdot 1 = 0 \cdot 0 = 0$

**Truth table**

| A | B | $\overline{A + B}$ | $\overline{A} \cdot \overline{B}$ |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |

**Proof of second theorem:**

To prove $\overline{A \cdot B} = \overline{A} + \overline{B}$

**Case 1:** A=0, B=0

L.H.S => $\overline{A \cdot B} = \overline{0 \cdot 0} = \overline{0} = 1$

R.H.S => $\overline{A} + \overline{B} = \overline{0} + \overline{0} = \overline{1} + \overline{1} = 1$

**Case 2:** A=0, B=1

L.H.S => $\overline{A \cdot B} = \overline{0 \cdot 1} = \overline{0} = 1$

R.H.S => $\overline{A} + \overline{B} = \overline{0} + \overline{1} = \overline{1} + \overline{0} = 1$

**Case 3:** A=1, B=0

$\overline{A \cdot B} = 1 \cdot 0 = 0 = \overline{1}$

L.H.S =>

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

R.H.S =>

**Case 4:** A=1, B=1

$\overline{A \cdot B} = \overline{1 \cdot 1} = \overline{1} = 0$

L.H.S =>

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

R.H.S =>

**Truth table**

| A | B | $\overline{A \cdot B}$ | $\overline{A} + \overline{B}$ |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |

**Number Systems and Codes: Primary, Octal, Hexadecimal, Signed numbers codes**

A number system relates quantities and symbols. The base or radix of a number system represents the number of digits or basic symbols in that particular number system. In decimal system the base is 10, because of use the numbers 0, 1, 2,3,4,5,6,7,8 and 9.

### Binary Number System

A binary number system is a code that uses only two basic symbols. The digits can be any two distinct characters, but it should be 0 or 1. The binary equivalent for some decimal numbers are given below

| Decimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Binary | 0 | 1 | 10 | 11 | 100 | 101 | 110 | 111 | 1000 | 1001 | 1010 | 1011 |

Each digit in a binary number has a value or weight. The LSB has a value of 1. The second from the right has a value of 2, the next 4 , etc.,

| 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|
| $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |

**Binary to decimal conversion:**

(1001)2 = X10

$1001 = 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0$

$=8+0+0+1$

(1001)2 = (9)10

**Fractions:**

For fractions the weights of the digit positions are written from right of the binary point and weights are given as follows.

| $2^{-1}$ | $2^{-2}$ | $2^{-3}$ | $2^{-4}$ | $2^{-5}$ |
|---|---|---|---|---|

**E.g.:**

(0.0110) 2= X10

$=0x2^{-1} + 1x2^{-2} + 1x2^{-3} + 0x2^{-4}$

=0 x0.5 + 1x0.25 + 1x0.125 + 0x0.0625

= (0.375)10

**E.g.:**

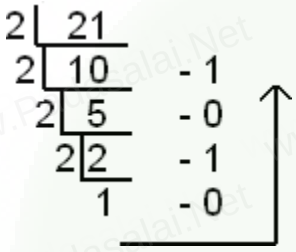
(1011.101) 2=X10

$=1x2^3 + 0x2^2 + 1x2^1 + 1x2^0 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3}$

$=8 + 0 + 2 + 1 + 0.5 + 0 + 0.125$

$= (11.625)10$

**E.g.:(21)2 = X2**




(21)2 = (10101)2

**Fractions:**

The fraction is multiplied by 2 and the carry in the integer position is written after each multiplication. Then they are written in the forward order to get the corresponding binary equivalent.

(0.4375)10 = X2

2 x 0.4375 = 0.8750 => 0

2 x 0.8750 = 1.750 => 1

2 x 0. 750 = 1.5 => 1

2 x 0.5 = 1.0 => 1

E.g.

## Octal Number System

Octal number system has a base of 8 i.e., it has eight basic symbols. First eight decimal digits 0, 1,2,3,4,5,6,7 are used in this system.

**Octal to decimal conversion:**

In the octal number system each digit corresponds to the powers of 8. The weight of digital position in octal number is as follows

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $8^4$ | $8^3$ | $8^2$ | $8^1$ | $8^0$ | $8^{-1}$ | $8^{-2}$ | $8^{-3}$ |

To convert from octal to decimal multiply each octal digit by its weight and add the resulting products.

**E.g.:(48)8 = X10**

$$48 = 4 \times 8^1 + 7 \times 8^0$$

$$= 32 + 7$$

$$= 39$$

(48)8 = (39)10

(0.437 5)10 = (0.011 1)2

**E.g.:**

(22.34)8 = X10

$22.34 = 2 \times 8^{1} + 2 \times 8^{0} + 3 \times 8^{-1} + 4 \times 8^{-2}$

=16 + 2 + 3 x 1/8 +4 x 1/64

= (18.4375)

(22.34)8 = (18.4375)10

**Decimal to octal conversion:**

Here the number is divided by 8 progressively and each time the remainder is written and finally the remainders are written in the reverse order to form the octal number. If the number has a fraction part, that part is multiplied by 8 and carry in the integer part is taken. Finally the carries are taken in the forward order.

**E.g.:**

(19.11)10 = X8

$8\overline{)19}$

2 - 3

0.11 x 8 = 0.88 => 0

0.88 x 8 = 7.04 => 7

0.04 x 8 = 0.32 => 0

0.32 x 8 = 2.56 => 2

0.56 x 8 = 4.48 => 4

(19.11)10 = (23.07024)8

**Octal to binary conversion:**

Since the base of octal number is 8, i.e., the third power of 2, each octal number is converted into its equivalent binary digit of length three.

**E.g.:**

(57.127)8 = X2

5 7 . 1 2 7

101 111 . 001 010 111

(57.127)8 = (101111001010111)2

**Binary to octal:**

The given binary number is grouped into a group of 3 bits, starting at the octal point and each group is converted into its octal equivalent.

**E.g.:**

(1101101.11101)2 = X8

001 101 101.111 010

1 5 5 . 7 2

(1101101.11101)2 = (155.72) 8

## Hexadecimal Number System:

The hexadecimal number system has a base of 16. It has 16 symbols from 0 through 9 and A through F.

| Decimal | Hexadecimal | Binary |
|---|---|---|
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |

| | | |
|---|---|---|
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| 10 | A | 1010 |
| 11 | B | 1011 |
| 12 | C | 1100 |
| 13 | D | 1101 |
| 14 | E | 1110 |
| 15 | F | 1111 |
| | | |

**Binary to hexadecimal:**

The binary number is grouped into bits of 4 from the binary point then the corresponding hexadecimal equivalent is written.

**E.g.:**

(100101110 . 11011) 2 = X16

0001 0010 1110 . 1101 1000

1 2 E . D 8

(100101110 . 11011) 2 = (12E . D8)16

**Hexadecimal to binary:**

Since the base of hexadecimal number is 16, i.e., the fourth power of 2, each hexadecimal number is converted into its equivalent binary digit of length four.

**E.g.:**

(5D. 2A)16 = X2

5 D . 2 A

0101 1101 . 0010 1010

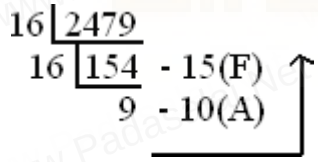
(5D. 2A)16 = (01011101.00101010)2

**Decimal to hexadecimal:**

The decimal number is divided by 16 and carries are taken after each division and then written in the reverse order. The fractional part is multiplied by 16 and carry is taken in the forward order.

**E.g.:**

(2479.859)) 10 = X 16




16 x 0.859 = 13.744 => 13 (D)

16 x 0.744 = 11.904 => 11 (B)

16 x 0.904 = 14.464 => 14 (E)

16 x 0.464 = 7.424 => 7

16 x 0.424 = 6.784 => 6

(2479.859)10 = (9AF.DBE76)16

**Hexadecimal to decimal:**

Each digit of the hexadecimal number is multiplied by its weight and then added.

**E.g.:**

(81.21) 16 = X10

$=8 \times 16^{1} + 1 \times 16^{0} + 2 \times 16^{-1} + 1 \times 16^{-2}$

$=8 \times 16 + 1 \times 1 + 2/16 + 1/16^{2}$

= (129.1289)10

(81.21) 16 = (129.1289)10

## Binary Arithmetic

**Binary Addition:**

To perform the binary addition we have to follow the binary table given below. 0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 0 => plus a carry-over of 1

Carry-overs are performed in the same manner as in decimal arithmetic. Since 1 is the largest digit in the binary system, any sum greater than 1 requires that a digit be considered over.

**E.g.**

| | | |
|---|---|---|
| 111 | 1010 | 11.01 |
| 110 | 1101 | 101.11 |
| 1001 | 10111 | 1001.00 |

**Binary Subtraction:**

To perform the binary subtraction the following binary subtraction table should be followed.

0 – 0 = 0
1 – 0 = 1
1 – 1 = 0
0 – 1 = 1 with a borrow of 1 is equivalent to 10 – 1 = 1

**E.g.:**

111
010
101

**E.g.:**

110.01
100.10
001.11

**1's complement:**

To obtain 1's complement of a binary number each bit of the number is subtracted from 1.

**E.g.:**

| Binary number | 1's Complement |
|---|---|
| 0101 | 1010 |
| 1001 | 0110 |
| 1101 | 0010 |
| 0001 | 1110 |

Thus 1's complement of a binary number is the number that results

when we change each 0 to a 1 and each 1 to a 0.

**1's complement subtraction:**

Instead of subtracting the second number from the first, the 1's complement of the second number is added to the first number. The last carry which is said to be a END AROUND CARRY, is added to get the final result.

**E.g.:**

7 - 111 ----------> 111 +

3 011 1's complement 100

4 1011 +

└→1

100 □ result

If there is no carry in the 1's complement subtraction, it indicates that the result is a negative and number will be in its 1's complement form. So complement it to get the final result.

**E.g.:**

8 - 1000 ----------> 1000 +

10 1010 1's complement 0101

4 1101 1's complement - 0010 □ result

The following points should be noted down when we do 1's complement subtraction.

1. Write the first number (minuend) as such.
2. Write the 1's complement of second number(subtrahend)
3. Add the two numbers.
4. The carry that arises from the addition is said to be "end around carry".
5. End-around carry should be added with the sum to get the result.
6. If there is no end around carry find out the 1's complement of the sum and put a negative sign before the result as the result is negative.

**2's Complement:**

2's complement results when we add '1' to 1's complement of the given number i.e., 2's complement =1's complement + 1

| Binary Number | 1's complement | 2's complement |
|---|---|---|
| 1010 | 010<br>1 | 0110 |
| 0101 | 101<br>0 | 1011 |
| 1001 | 011<br>0 | 0111 |
| 0001 | 111<br>0 | 1111 |

**2's Complement Subtraction: Steps:**

1. Write the first number as such
2. Write down the 2's complement of the second number.
3. Add the two numbers.
4. If there is a carry, discard it and the remaining part (sum) will be the result (positive).
5. If there is no carry, find out the 2's complement of the sum and put negative sign before the result as the result is negative.

**E.g.:**

1) 10 - 1010 ----------> 1010 +
8 1000 2's complement 1000
2 10010

0010 □ result

2) 5 - 0101 ----------> 0101 +
12 1100 2's complement
0100
4 1001 2's complement – 0111□ result

**Binary multiplication:**

The table for binary multiplication is given below 0 x 0 = 0

0 x 1 = 0

1 x 0 = 0

1 x 1 = 1

**E.g.:**

1011 x 110

1011 x

110 0000

1011

1011

1000010

**E.g.:**

101.01 x 11.01

101.01 x

11.01 101 01

00000

10101

10101

10001.0001

**Binary division:**

The table for binary division is as follows. 0 ÷ 1 = 0

1 ÷ 1 = 1

As in the decimal system division by zero is meaning less.

**E.g.:**

1) 1100 ÷ 11

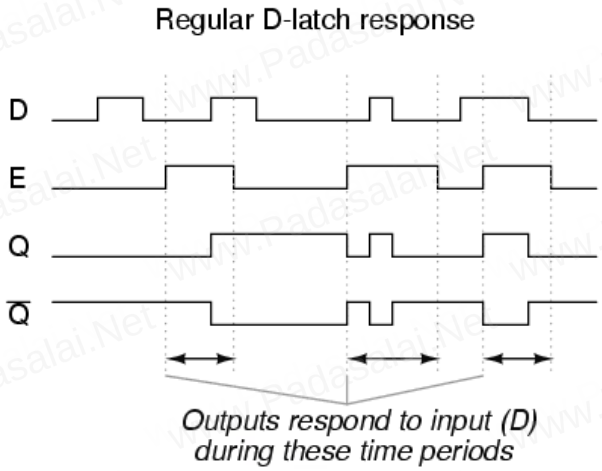
```
        100
11 | 1100
   | 11
   0
```

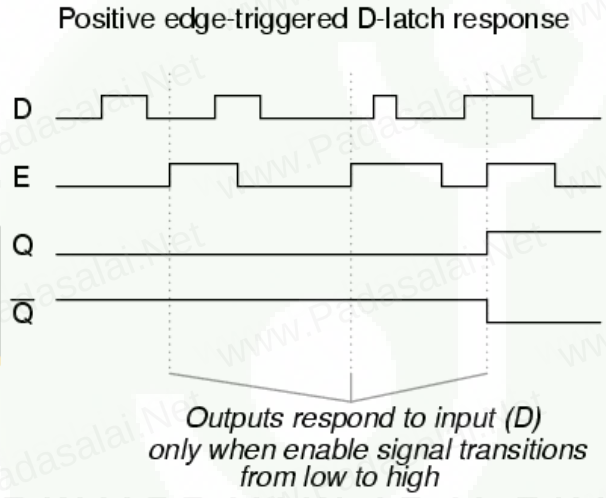
2) 1001 ÷ 10

```
        100.1
10 | 1001
   | 10
   0010
     10
      0
```

# Edge triggered latches Flip Flops

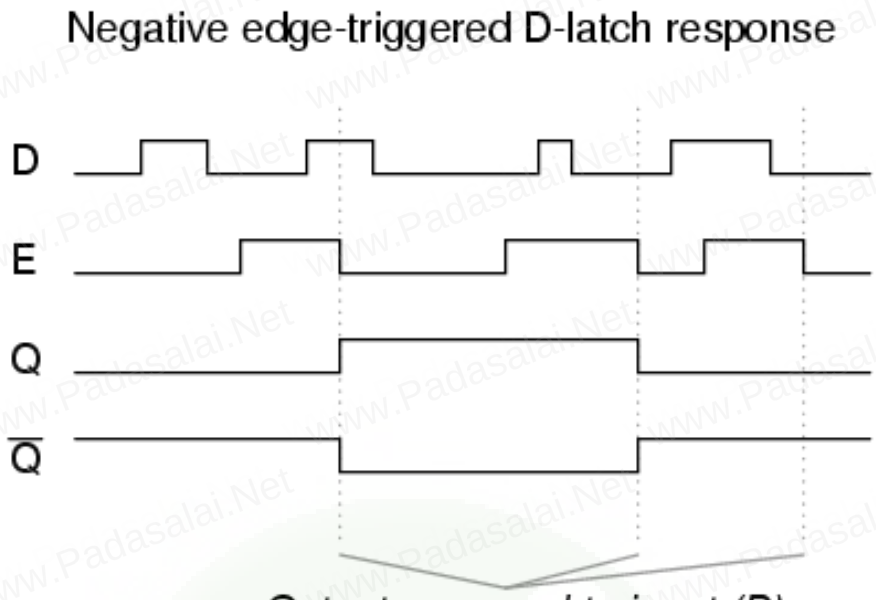
So far, we've studied both S-R and D latch circuits with an enable inputs. The latch responds to the data inputs (S-R or D) only when the enable input is activated. In many digital applications, however, it is desirable to limit the responsiveness of a latch circuit to a very short period of time instead of the entire duration that the enabling input is activated. One method of enabling a multivibrator circuit is called *edge triggering*, where the circuit's data inputs have control only during the time that the enable input is *transitioning* from one state to another. Let's compare timing diagrams for a normal D latch versus one that is edge-triggered:




In the first timing diagram, the outputs respond to input D whenever the enable (E) input is high, for however long it remains high. When the enable signal falls back to a low state, the circuit remains latched. In the second timing diagram, we note a distinctly different response in the circuit output(s): it only responds to the D input during that brief moment of time when the enable signal *changes*, or *transitions*, from low to high. This is known as *positive* edge-triggering.

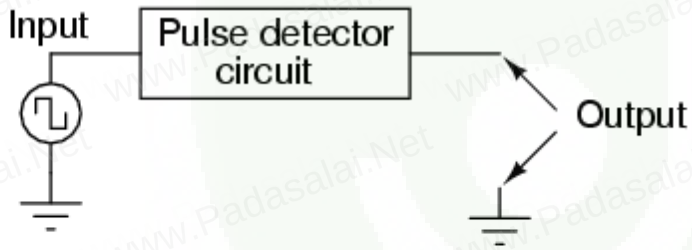
There is such a thing as *negative* edge triggering as well, and it produces the following response to the same input signals:

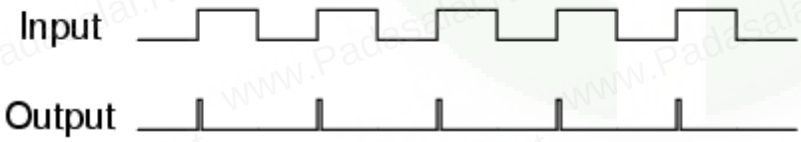



Negative edge-triggered D-latch response
D
E
Q
Q
Outputs respond to input (D)
only when enable signal transitions
from high to low

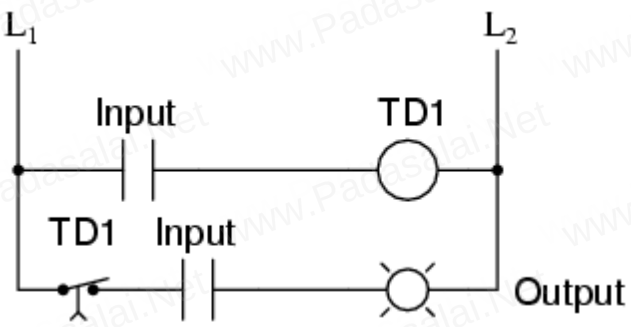
Whenever we enable a multivibrator circuit on the transitional edge of a square-wave enable signal, we call it a *flip-flop* instead of a *latch*. Consequently, and edge-triggered S-R circuit is more properly known as an S-R flip-flop, and an edge-triggered D circuit as a D flip-flop. The enable signal is renamed to be the *clock* signal. Also, we refer to the data inputs (S, R, and D, respectively) of these flip-flops as *synchronous* inputs, because they have effect only at the time of the clock pulse edge (transition), thereby synchronizing any output changes with that clock pulse, rather than at the whim of the data inputs.

But, how do we actually accomplish this edge-triggering? To create a "gated" S-R latch from a regular S-R latch is easy enough with a couple of AND gates, but how do we implement logic that only pays attention to the *rising or falling edge* of a changing digital signal? What we need is a digital circuit that outputs a brief pulse whenever the input is activated for an arbitrary period of time, and we can use the output of this circuit to briefly enable the latch. We're getting a little ahead of ourselves here, but this is actually a kind of monostable multivibrator, which for now we'll call a *pulse detector*.

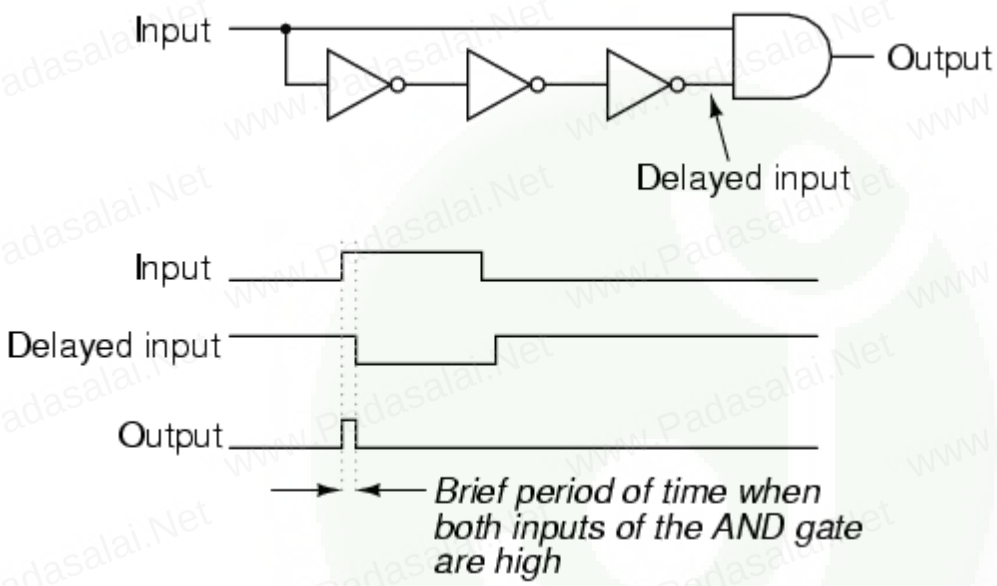


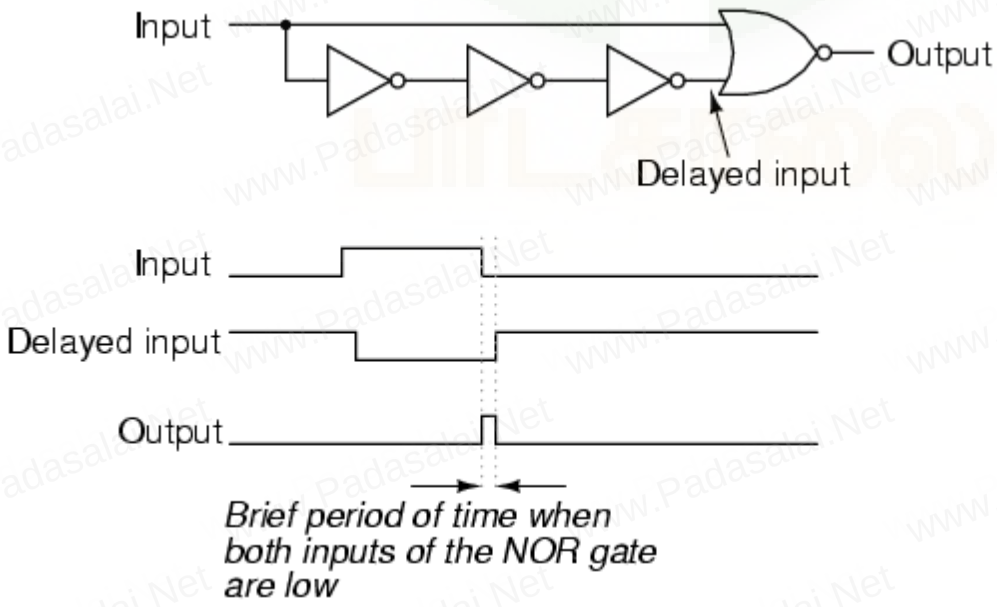

The duration of each output pulse is set by components in the pulse circuit itself. In ladder logic, this can be accomplished quite easily through the use of a time-delay relay with a very short delay time:



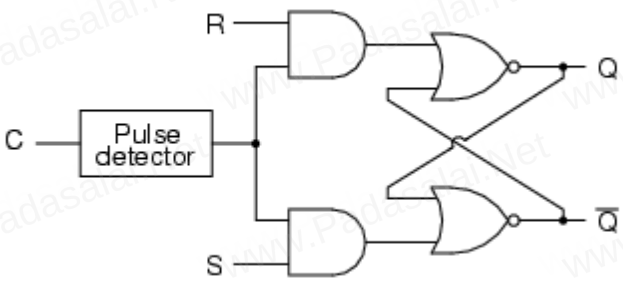
Implementing this timing function with semiconductor components is actually quite easy, as it exploits the inherent time delay within every logic gate (known as*propagation delay*). What we do is take an input signal and split it up two ways, then place a gate or a series of gates in one of those signal paths just to delay it a bit, then have both the original signal and its delayed counterpart enter into a two-input gate that outputs a high signal for the brief moment of time that the delayed signal has not yet caught up to the low-to-high change in the non-delayed signal. An example circuit for producing a clock pulse on a low-to-high input signal transition is shown here:




This circuit may be converted into a negative-edge pulse detector circuit with only a change of the final gate from AND to NOR:
Now



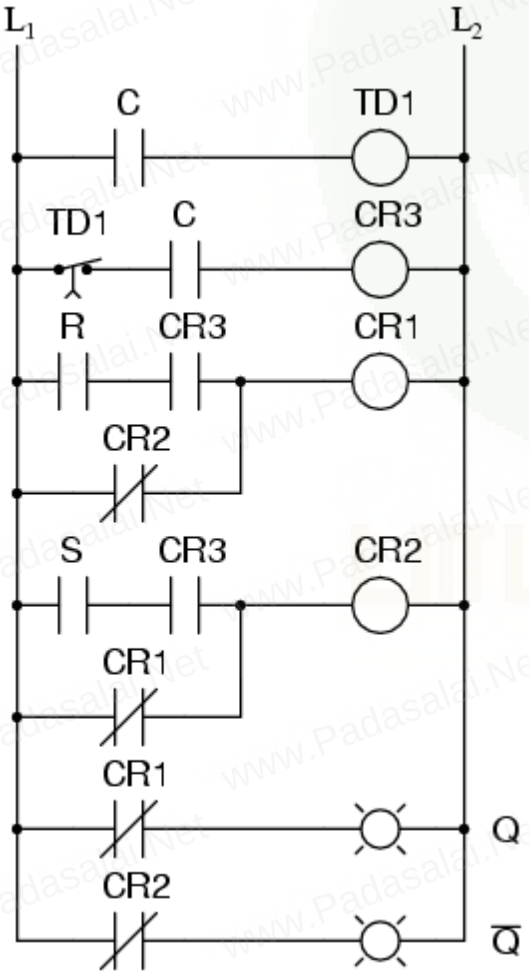
Now that we know how a pulse detector can be made, we can show it attached to the enable input of a latch to turn it into a flip-flop. In this case, the circuit is a S-R flip-flop:




| C | S | R | Q | $\overline{Q}$ |
|---|---|---|---|---|
| ⎍ | 0 | 0 | latch | latch |
| ⎍ | 0 | 1 | 0 | 1 |
| ⎍ | 1 | 0 | 1 | 0 |
| ⎍ | 1 | 1 | 0 | 0 |
| x | 0 | 0 | latch | latch |
| x | 0 | 1 | latch | latch |
| x | 1 | 0 | latch | latch |
| x | 1 | 1 | latch | latch |

Only when the clock signal (C) is transitioning from low to high is the circuit responsive to the S and R inputs. For any other condition of the clock signal ("x") the circuit will be latched.

A ladder logic version of the S-R flip-flop is shown here:

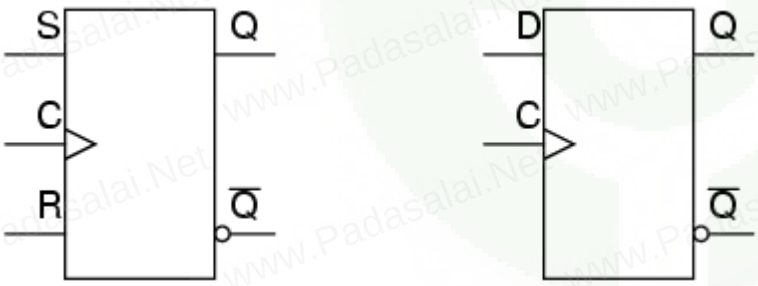



| C | S | R | Q | $\overline{Q}$ |
|---|---|---|---|---|
| ⎍ | 0 | 0 | latch | latch |
| ⎍ | 0 | 1 | 0 | 1 |
| ⎍ | 1 | 0 | 1 | 0 |
| ⎍ | 1 | 1 | 0 | 0 |
| x | 0 | 0 | latch | latch |
| x | 0 | 1 | latch | latch |
| x | 1 | 0 | latch | latch |
| x | 1 | 1 | latch | latch |

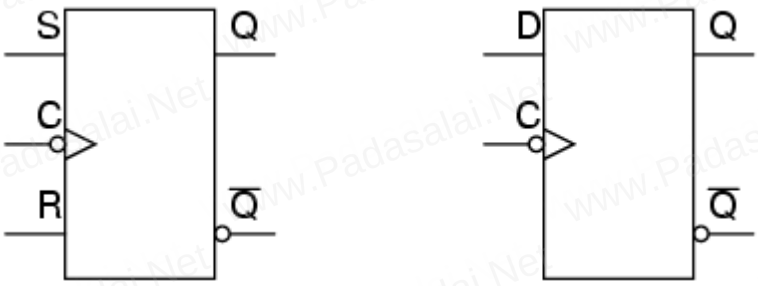
Relay contact CR3 in the ladder diagram takes the place of the old E contact in the S-R latch circuit, and is closed only during the short time that both C is closed and time-delay contact TR1 is closed. In either case (gate or ladder circuit), we see that the inputs S and R have no effect unless C is transitioning from a low (0) to a high (1) state. Otherwise, the flip-flop's outputs latch in their previous states.

It is important to note that the invalid state for the S-R flip-flop is maintained only for the short period of time that the pulse detector circuit allows the latch to be enabled. After that brief time period has elapsed, the outputs will latch into either the set or the reset state. Once again, the problem of a *race condition* manifests itself. With no enable signal, an invalid output state cannot be maintained. However, the valid "latched" states of the multivibrator -- set and reset -- are mutually exclusive to one another. Therefore, the two gates of the multivibrator circuit will "race" each other for supremacy, and whichever one attains a high output state first will "win."
The block symbols for flip-flops are slightly different

The block symbols for flip-flops are slightly different from that of their respective latch counterparts:




The triangle symbol next to the clock inputs tells us that these are edge-triggered devices, and consequently that these are flip-flops rather than latches. The symbols above are positive edge-triggered: that is, they "clock" on the rising edge (low-to-high transition) of the clock signal. Negative edge-triggered devices are symbolized with a bubble on the clock input line:



Both of the above flip-flops will "clock" on the falling edge (high-to-low transition) of the clock signal.

**REVIEW:**

☐ A *flip-flop* is a latch circuit with a "pulse detector" circuit connected to the enable (E) input, so that it is enabled only for a brief moment on either the rising or falling edge of a clock pulse.

☐ Pulse detector circuits may be made from time-delay relays for ladder logic applications, or from semiconductor gates (exploiting the phenomenon of *propagation delay*).

# Decoders and Encoders

- Decoders
- Expansion of decoders
- Combinational circuit implementation with decoders
- Some examples of decoders
- Encoders
- Major limitations of encoders
- Priority encoders
- Some examples of ecnoders

## Decoders

As its name indicates, a decoder is a circuit component that decodes an input code. Given a binary code of n-bits, a decoder will tell which code is this out of the $2_n$ possible codes

(See Figure 1(a)).

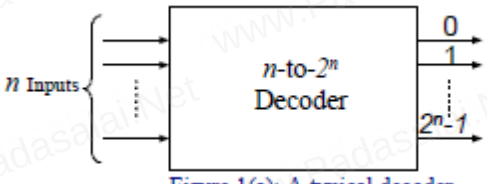



Figure 1(a): A typical decoder

Thus, a decoder has n- inputs and 2n outputs. Each of the 2n outputs corresponds to one of the possible 2n input combinations

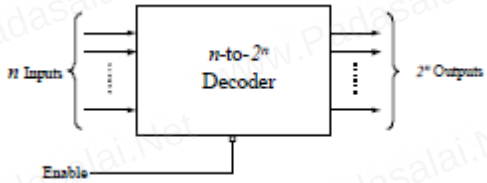



Figure 1(b): A typical decoder

Figure 1(b) shows the block diagram of a typical decoder, which has n input lines, and m output lines, where m is equal to $2^n$. The decoder is called n-to-m decoder. Apart from this, there is also a single line connected to the decoder called enable line. The operations of the enable line will be discussed in the flowing text.

In general, output i equals 1 if and only if the input binary code has a value of i.

- Thus, each output line equals 1 at only one input combination but is equal to 0 at all other combinations.
- In other words, each decoder output corresponds to a minterm of the n input variables.
- Thus, the decoder generates all of the $2^n$ minterms of n input variables.

## Example: 2-to-4 decoders

Let us discuss the operation and combinational circuit design of a decoder by taking the specific example of a 2-to-4 decoder. It contains two inputs denoted by $A_1$ and $A_0$ and four outputs denoted by $D_0$, $D_1$, $D_2$, and $D_3$ as shown in figure 2. Also note that $A_1$ is the MSB while $A_0$ is the LSB.

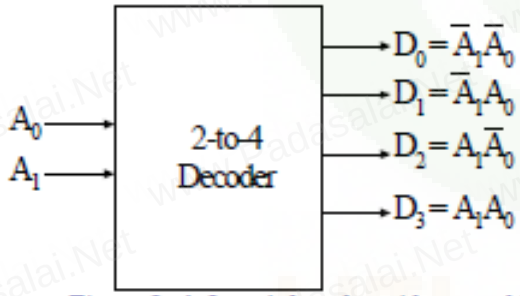



Figure 2: A 2-to-4 decoder without enable

| Decimal # | Input | | Output | | | |
|---|---|---|---|---|---|---|
| | $A_1$ | $A_0$ | $D_0$ | $D_1$ | $D_2$ | $D_3$ |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 1 | 0 |
| 3 | 1 | 1 | 0 | 0 | 0 | 1 |

Table 1: Truth table for 2-to-4 decoder

As we see in the truth table (table 1), for each input combination, one output line is activated, that is, the output line corresponding to the input combination becomes 1, while other lines remain inactive. For example, an input of 00 at the input will activate line $D_0$. 01 at the input will activate line $D_1$, and so on.

Notice that, each output of the decoder is actually a minterm resulting from a certain combination of the inputs, that is

- $D_0 = A_1 A_0$, ( minterm $m_0$) which corresponds to input 00
- $D_1 = A_1 A_0$, ( minterm $m_1$) which corresponds to input 01
- $D_2 = A_1 A_0$, ( minterm $m_2$) which corresponds to input 10
- $D_3 = A_1 A_0$, ( minterm $m_3$) which corresponds to input 11

This is depicted in Figures 2 where we see that each input combination will inovke the corresponding output, where each output is minterm corresponding to the input combination.

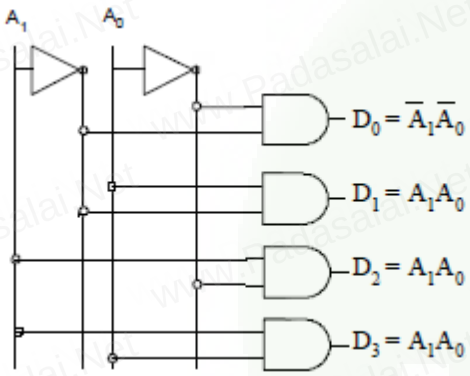



Figure 3: Implementation 2-to-4 decoder

The circuit is implemented with AND gates, as shown in figure 3. In this circuit we see that the logic equation for $D_0$ is $A_1$

/ $A_0$

/. $D_0$ is $A_1$

/ $A_0$, and so on. These are in fact the minterms being implemented. Thus, each output of the decoder generates a minterm corresponding to the input combination.

## The “enable” input in decoders

Generally, decoders have the “enable” input .The enable input perroms no logical operation, but is only esponsible for making the decoder ACTIVE or INACTIVE.

- If the enable “E”
- is zero, then all outputs are zero regardless of the input values.
- is one, then the decoder performs its normal operation.

For example, consider the 2-to-4 decoder with the enable input (Figure 4). The enable input is only responsible for making the decoder active or inactive. If Enable E is zero, then all outputs of the decoder will be zeros, regardless of the values of $A_1$ and $A_0$.

However, if E is 1, then the decoder will perform its normal operation, as is shown in the truth table (table 2). In this table we see that as long as E is zero, the outputs $D_0$ to $D_3$ will remain zero, no matter whatever value you provide at the inputs $A_1$ $A_0$, depicted by two don't cares. When E becomes 1, then we see the same behavior as we saw in the case of 2-to-4 decoder discussed earlier.

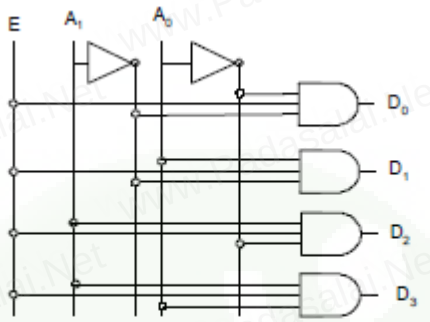



Figure 4: Implementation 2-to-4 decoder with enable

| Decimal value | Enable | Inputs | | Outputs | | | |
|---|---|---|---|---|---|---|---|
| | E | $A_1$ | $A_0$ | $D_0$ | $D_1$ | $D_2$ | $D_3$ |
| | 0 | X | X | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 3 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |

Table 2: Truth table of 2-to-4 decoder with enable

**Example: 3-to-8 decoders**

In a three to eight decoder, there are three inputs and eight outputs, as shown in figure 5.

A0 is the least significant variable, while A2 is the most significant variable.

The three inputs are decoded into eight outputs. That is, binary values at the input form a combination, and based on this combination, the corresponding output line is activated.

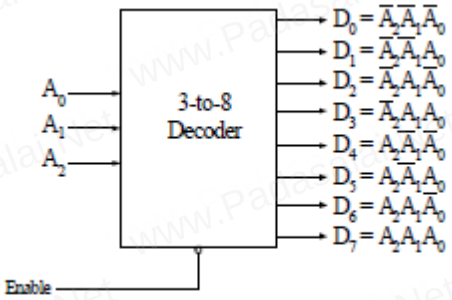


Figure 5: A 3-to-8 decoder with enable

Each output represents one minterm.

- For example, for input combination A2A1A0 = 001, output line D1 equals 1 while all other output lines equal 0's
- It should be noted that at any given instance of time, one and only one output line can be activated. It is also obvious from the fact that only one combination is possible at the input at a time, so the corresponding output line is activated.

| Dec. Code | Inputs | | | Outputs | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | $A_2$ | $A_1$ | $A_0$ | $D_0$ | $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 4 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 6 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 7 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

## Table 3: Truth table of 3-to-8 decoder

Since each input combination represents one minterm, the truth table (table 3) contains eight output functions, from D0 to D7 seven, where each function represents one and only one minterm. Thus function D0 is A2
/ A1
/ A0
/. Similarly function D7 is A2A1A0. The corresponding circuit is given in Figure 6. In this figure, the three inverters provide complement of the inputs, and each one of the AND gates generates one of the minterms.
It is also possible to add an Enable input to this decoder.

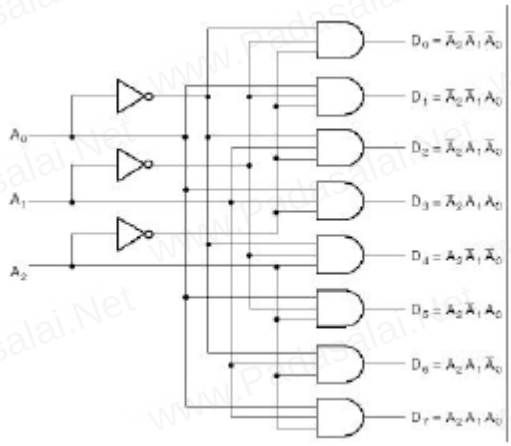


Figure 6: Implementation of a 3-to-8 decoder without enable

## Decoder Expansion

- It is possible to build larger decoders using two or more smaller ones.
- For example, a 6-to-64 decoder can be designed with four 4-to-16 decoders and one 2-to-4 line decoder.

Example: Construct a 3-to-8 decoder using two 2-to-4 deocders with enable inputs.

Figure 7 shows how decoders with enable inputs can be connected to form a larger decoder. Two 2-to-4 line decoders are combined to build a 3-to-8 line decoder.

- The two least signifncat bits (i.e. $A_1$ and $A_0$) are connected to both decoders
- Most signifcant bit ($A_2$) is connected to the enable input of one decoder.
- The complement of most significant bit ($A_2$) is connected to the enable of the other decoder.
- When $A_2 = 0$, upper decoder is enabled, while the lower is disabled. Thus, the outputs of the upper decoder correspond to minterms $D_0$ through $D_3$.
- When $A_2 = 1$, upper decoder is disabled, while the lower is enabled. Thus, the outputs of the lower decoder correspond to minterms $D_4$ through $D_7$.

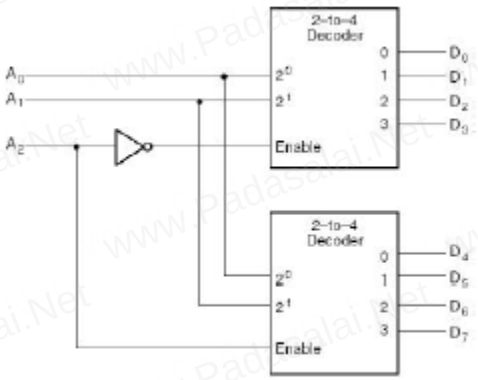



Figure 7: Implementing a 3-to-8 decoder with two 2-to-4 decoders

## Decoder design with NAND gates

- Some decoders are constructed with NAND rather than AND gates.
- In this case, all decoder outputs will be 1's except the one corresponding to the input code which will be 0.

| Decimal # | Input | | Output | | | |
|---|---|---|---|---|---|---|
| | $A_1$ | $A_0$ | $D_0'$ | $D_1'$ | $D_2'$ | $D_3'$ |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 | 0 | 1 |
| 3 | 1 | 1 | 1 | 1 | 1 | 0 |

$$\overline{D}_0 = \overline{\overline{A}_1\overline{A}_0} \qquad \overline{D}_1 = \overline{\overline{A}_1 A_0}$$

$$\overline{D}_2 = \overline{A_1\overline{A}_0} \qquad \overline{D}_3 = \overline{A_1 A_0}$$

Table 4: Truth table of 2-to-4 decoder with NAND gates

Table 4: Truth table of 2-to-4 decoder with NAND gates

This decoder can be constructed without enable, similar to what we have seen in the design of decoder with AND gates, without enable. The truth table and corresponding minters are given in table 4. Notice that the minters are in the complemented form.

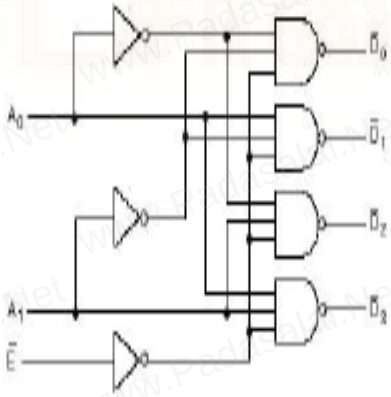



Figure 8: A 2-to-4 decoder with Enable constructed with NAND gates.

| Decimal value | Enable | Inputs | | Outputs | | | |
|---|---|---|---|---|---|---|---|
| | E' | $A_1$ | $A_0$ | $D_0$' | $D_1$' | $D_2$' | $D_3$' |
| | 1 | X | X | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 2 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 3 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |

$$\overline{D}_0 = E\overline{A}_1\overline{A}_0 \qquad \overline{D}_1 = E\overline{A}_1A_0$$

$$\overline{D}_2 = EA_1\overline{A}_0 \qquad \overline{D}_3 = EA_1A_0$$

Table 5: Truth table of 2-to-4 decoder with Enable using NAND gates

A 2-to-4 line decoder with an enable input constructed with NAND gates is shown in figure 8.

The circuit operates with complemented outputs and enable input E' is also complemented to match the outputs of the NAND gate decoder. The decoder is enabled when E' is equal to zero. As indicated by the truth table, only one output can be equal to zero at any given time, all other outputs being equal to one. The output with the value of zero represents the minterm selected by inputs $A_1$ and $A_0$. The circuit is disabled when E' is equal to one, regardless of the values of the other two inputs. When the circuit is disabled, none of the outputs are equal to zero, and none of the minterms are selected.

The corresponding logic equations are also given in table 5.

## Combinational circuit implementation using decoder

- As known, a decoder provides the $2^n$ minterms of $n$ input variables
- Since any boolean functions can be expressed as a sum of minterms, one can use a decoder to implement any function of $n$ variables.
- In this case, the decoder is used to generate the $2^n$ minterms and an additional OR gate is used to generate the sum of the required minterms.
- In this way, any combinational circuit with $n$ inputs and $m$ outputs can be implemented using an $n$-to-$2^n$ decoder in addition to $m$ OR gates.

Remember, that

- The function need not be simplified since the decoder implements a function using the minterms, not product terms.
- Any number of output functions can be implemented using a single decoder, provided that all those outputs are functions of the same input variables.

## Example: Decoder Implementation of a Full Adder

Let us look at the truth table (table 6) for the given problem. We have two outputs, called S, which stands for sum, and C, which stands for carry. Both sum and carry are functions of X, Y, and Z.

| Decimal value | Input | | | Output | |
|---|---|---|---|---|---|
| | X | Y | Z | S | C |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 2 | 0 | 1 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 | 0 | 1 |
| 4 | 1 | 0 | 0 | 1 | 0 |
| 5 | 1 | 0 | 1 | 0 | 1 |
| 6 | 1 | 1 | 0 | 0 | 1 |
| 7 | 1 | 1 | 1 | 1 | 1 |

Table 6: Truth table of the Full Adder

- The output functions S & C can be expressed in sum-of-minterms forms as follows:
- S (X,Y,Z) = m (1,2,4,7)
- C (X,Y,Z) = m (3,5,6,7)

Looking at the truth table and the functions in sum of minterms form, we observe that there are three nputs, X, Y, and Z that correspond to eight minterms. This implies that a 3-to-8 decoder is needed to implement this function. This implementation is given in Figure 9, where the sum S is implemented by taking minterms 1, 2, 4, and 7 and the OR gates forms the logical sum of minterm for S. Similarly, carry C is implemented by taking logical sum of minterms 3, 5, 6, and 7 from the same decoder.

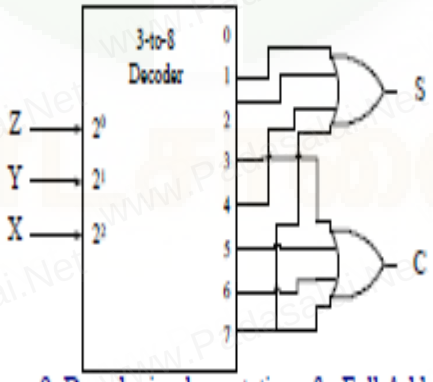



Figure 9: Decoder implementation of a Full Adder

## Encoders

- An encoder performs the inverse operation of a decoder, as shown in Figure 10.

- It has $2_n$ inputs, and $n$ output lines.
- Only one input can be logic 1 at any given time (active input). All other inputs must be 0's.
- Output lines generate the binary code corresponding to the active input.

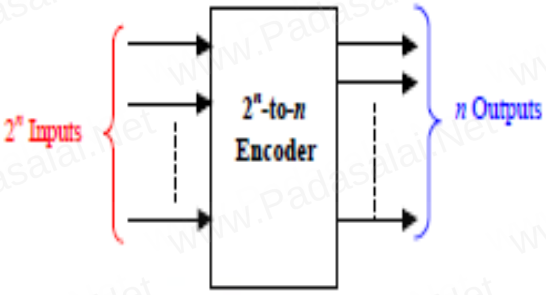



Figure 10: A typical Encoder

## Example: Octal-to-binary encoder

We will use 8-to-3 encoder (Figure 11) for this problem, since we have eight inputs, one for each of the octal digits, and three outputs that generate the corresponding binary number. Thus, in the truth table, we see eight input variables on the left side of the vertical lines, and three variables on the right side of the vertical line (table 7).

| Inputs | | | | | | | | Outputs | | | Decimal Code |
|---|---|---|---|---|---|---|---|---|---|---|---|
| E7 | E6 | E5 | E4 | E3 | E2 | E1 | E0 | A2 | A1 | A0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 3 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 4 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 5 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 6 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 7 |

Table 7: Truth table of Octal-to-binary encoder

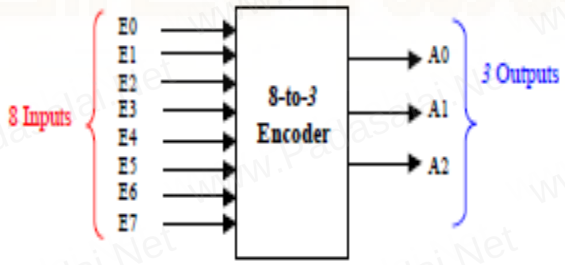



Figure 11: Octal-to-binary encoder

Figure 11: Octal-to-binary encoder

- Note that not all input combinations are valid.

- Valid combinations are those which have exactly one input equal to logic 1 while all other inputs are logic 0's.
- Since, the number of inputs = 8, K-maps cannot be used to derive the output Boolean expressions.
- The encoder implementation, however, can be directly derived from the truth table
- Since $A_0 = 1$ if the input octal digit is 1 or 3 or 5 or 7, then we can write:

$A_0 = E_1 + E_3 + E_5 + E_7$

- Likewise, $A_1 = E_2 + E_3 + E_6 + E_7$, and similarly
- $A_2 = E_4 + E_5 + E_6 + E_7$
- Thus, the encoder can be implemented using three 4- input OR gates.

## Major Limitation of Encoders

- Exactly one input must be active at any given time.
- If the number of active inputs is less than one or more than one, the output will be incorrect.
- For example, if $E_3 = E_6 = 1$, the output of the encoder $A_2A_1A_0 = 111$, which implies incorrect output.

**Two Problems to Resolve.**

1. If two or more inputs are active at the same time, what should the output be?
2. An output of all 0's is generated in 2 cases:
   - when all inputs are 0
   - when $E_0$ is equal to 1.

How can this ambiguity be resolved?

***Solution To Problem 1*:**

- Use a *Priority Encoder* which produces the output corresponding to the input with higher priority.
- Inputs are assigned priorities according to their subscript value; e.g. higher subscript inputs are assigned higher priority.
- In the previous example, if $E_3 = E_6 = 1$, the output corresponding to $E_6$ will be produced ($A_2A_1A_0 = 110$) since $E_6$ has higher priority than $E_3$.

***Solution To Problem 2*:**

- Provide one more output signal *V* to indicate *validity* of input data.
- *V* = 0 if none of the inputs equals 1, otherwise it is 1

## Example: 4-to-2 Priority Encoders

- Sixteen input combinations
- Three output variables A1, A0, and V
- V is needed to take care of situation when all inputs are equal to zero.

| Inputs | | | | Outputs | | |
|---|---|---|---|---|---|---|
| E3 | E2 | E1 | E0 | A1 | A0 | V |
| 0 | 0 | 0 | 0 | X | X | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Invalid Input

Table 8: Truth table of 4-to-2 Priority Encoder

In the truth table (table 8), we have sixteen input combinations. In the output, we have three variables. The variable V is needed to take care of the situation where all inputs are zero. In that case V is kept at zero, regardless of the values of A1 and A0. This combination is highlighted green. In all other cases, V is kept at 1, because at least one of the inputs is one.

When E0 is 1, the output combination of A1 and A0 is 00. This combination is highlighted blue.

Then we have two combinations highlighted yellow. In both these combinations, A1 and A0 are 01. This is because in both these combinations E1 is 1, regardless of the value of E0, and since E1 has higher subscript, the corresponding output value is 01.

This is followed by four input combinations in pink. In these four combinations, the output A1A0 is 10, since E2 is 1 in all these combinations, and E2 has the highest precedence compared to E0 and E1. Although E0 and E1 are also having a value of one in this set of four combinations, but they do not have the priority.

Finally we have the last eight input combinations, whose output is 11. This is because E3 is the highest priority input, and it is equal to 1. Though the other inputs with smaller subscripts, namely, E2, E1, and E0 are also having values of one in some combinations, but they do not have the priority.

The truth table can be rewritten in a more compact form using don't care conditions for inputs as shown below in table 9.

| | Inputs | | | | Outputs | | |
|---|---|---|---|---|---|---|---|
| | E3 | E2 | E1 | E0 | A1 | A0 | V |
| 1 | 0 | 0 | 0 | 0 | X | X | 0 |
| 2 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 3 | 0 | 0 | 1 | X | 0 | 1 | 1 |
| 4 | 0 | 1 | X | X | 1 | 0 | 1 |
| 5 | 1 | X | X | X | 1 | 1 | 1 |

Table 9: Truth table of 4-to-2 priority encoder (compact form)

With 4 Input variables, the truth table must have 16 rows, with each row representing an input combination.

- With don't care input conditions, the number of rows can be reduced since rows with don't care inputs will actually represent more than one input combination.
- Thus, for example, row # 3 represents 2 combinations since it represents the input conditions $E_3E_2E_1E_0$=0010 and 0011.
- Likewise, row # 4 represents 4 combinations since it represents the input conditions $E_3E_2E_1E_0$=0100, 0101, 0110 and 0111.
- Similarly, row # 5 represents 8 combinations.
- Thus, the total number of input combinations represented by the 5-row truth table = 1+ 1+ 2+ 4 + 8= 16 input combinations.

## Boolean Expressions for V, $A_1$ and $A_0$ and the circuit:

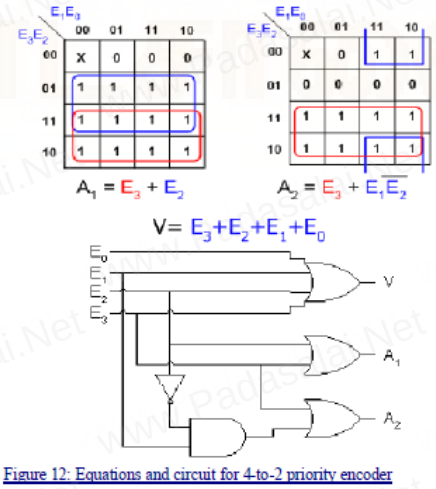



Figure 12: Equations and circuit for 4-to-2 priority encoder

# Multiplexer and Demultiplexer

A multiplexer is a circuit that accept many input but give only one output. A demultiplexer function exactly in the reverse of a multiplexer, that is a demultiplexer accepts only one input and gives many outputs. Generally multiplexer and demultiplexer are used together, because of the communication systems are bi directional.

**Mutliplexer:**

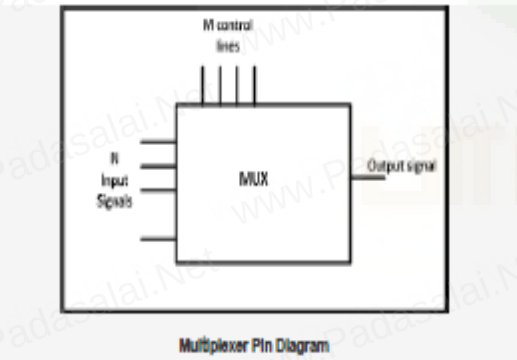
Multiplexer means many into one. A multiplexer is a circuit used to select and route any one of the several input signals to a signal output. An simple example of an non electronic circuit of a multiplexer is a single pole multiposition switch.

Multiposition switches are widely used in many electronics circuits. However circuits that operate at high speed require the multiplexer to be automatically selected. A mechanical switch cannot perform this task satisfactorily. Therefore, multiplexer used to perform high speed switching are constructed of electronic components.

Multiplexer handle two type of data that is analog and digital. For analog application, multiplexer are built of 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 input but only one output. By applying control signals, we can steer any input to the output. Few types of multiplexer are 2-to-1, 4-to-1, 8-to-1, 16-to-1 multiplexer.

Following figure shows the general idea of a multiplexer with n input signal, m control signals and one output signal.



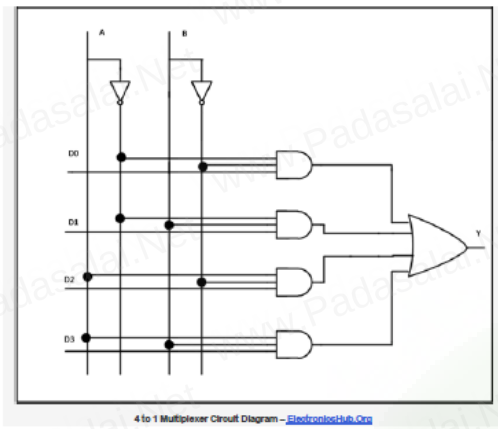
Multiplexer Pin Diagram

## Understanding 4-to-1 Multiplexer:

The 4-to-1 multiplexer has 4 input bit, 2 control 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

value of AB which is the control input. The control input determines which of the input data bit is transmitted to the output.

For instance, as shown in fig. when AB = 00, 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.




4 to 1 Multiplexer Circuit Diagram – ElectronicsHub.Org

If the control input is changed to AB =11, all gates are disabled except the bottom AND gate. In this case, D3 is transmitted to the output and Y = D3.

- An example of 4-to-1 multiplexer is IC 74153 in which the output is same as the input.
- Another example of 4-to-1 multiplexer is 45352 in which the output is the compliment of the input.
- Example of 16-to-1 line multiplexer is IC74150.

## *Applications of Multiplexer:*

Multiplexer are used in various fields where multiple data need to be transmitted using a single line. Following are some of the applications of multiplexers -

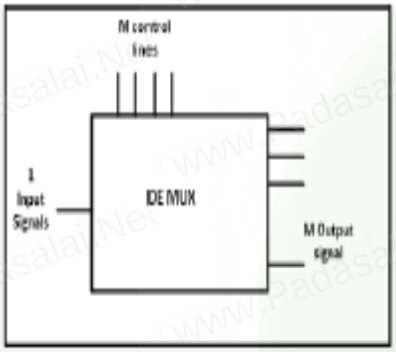
1. **Communication system** – Communication system is a set of system that enable communication like transmission system, relay and tributary station, and communication network. The efficiency of communication system can be increased considerably using multiplexer. Multiplexer allow the process of transmitting different type of data such as audio, video at the same time using a single transmission line.
2. **Telephone network –** In telephone network, multiple audio signals are integrated on a single line for transmission with the help of multiplexers. In this way, multiple audio signals can be isolated and eventually, the desire audio signals reach the intended recipients.
3. **Computer memory** - Multiplexers are used to implement huge amount of memory into the computer, at the same time reduces the number of copper lines required to connect the memory to other parts of the computer circuit.

4. **Transmission from the computer system of a satellite** – Multiplexer can be used for the transmission of data signals from the computer system of a satellite or spacecraft to the ground system using the GPS (Global Positioning System) satellites.

## Demultiplexer:

Demultiplexer means one to many. A demultiplexer is a circuit with one input and many output. 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.

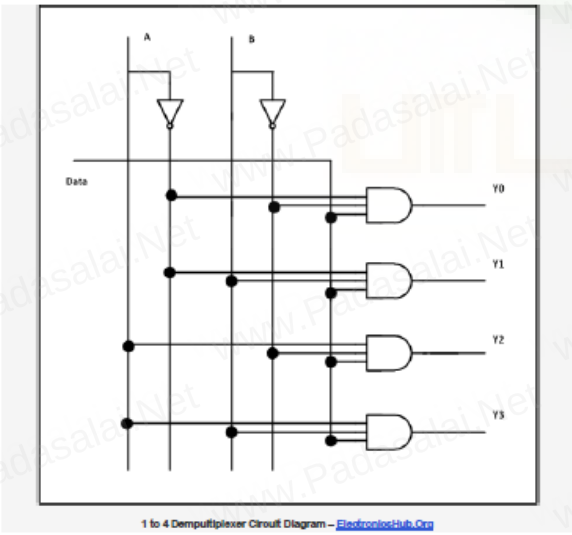
Following figure illustrate the general idea of a demultiplexer with 1 input signal, m control signals, and n output signals.



Demultiplexer Pin Diagram

## Understanding 1- to-4 Demultiplexer:

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



1 to 4 Demultiplexer Circuit Diagram – ElectronicsHub.Org

The input bit is labelled as Data D. This data bit is transmitted to the data bit of the output lines. This depends on the value of AB, the control input.

When AB = 01, the upper second AND gate is enabled while other AND gates are disabled. Therefore, only data bit D is transmitted to the output, 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 AB = 10, 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.

Example of 1-to-16 demultiplexer is IC 74154 it has 1 input bit, 4 control bits and 16 output bit.

## *Applications of Demultiplexer:*

1. Demultiplexer is used to connect a single source to multiple destinations. The main application area of demultiplexer is communication system where multiplexer are used. Most of the communication system are bidirectional i.e. they function in both ways (transmitting and receiving signals). Hence, for most of the applications, the multiplexer and demultiplexer work in sync. Demultiplexer are also used for reconstruction of parallel data and ALU circuits.

2. **Communication System** - Communication system use multiplexer to carry multiple data like audio, video and other form of data using a single line for transmission. This process make the transmission easier. The demultiplexer receive the output signals of the multiplexer and converts them back to the original form of the data at the receiving end. The multiplexer and demultiplexer work together to carry out the process of transmission and reception of data in communication system.

3. **ALU (Arithmetic Logic Unit)** – In an ALU circuit, the output of ALU can be stored in multiple registers or storage units with the help of demultiplexer. The output of ALU is fed as the data input to the demultiplexer. Each output of demultiplexer is connected to multiple register which can be stored in the registers.

4. **Serial to parallel converter** - A serial to parallel converter is used for reconstructing parallel data from incoming serial data stream. In this technique, serial data from the incoming serial data stream is given as data input to the demultiplexer at the regular intervals. A counter is attach to the control input of the demultiplexer. This counter directs the data signal to the output of the demultiplexer where these data signals are stored. When all data signals have been stored, the output of the demultiplexer can be retrieved and read out in parallel.

# K-maps

## Minimization ofBoolean expression

- *The minimization will result in reduction of the number of gates (resulting from less number of terms) and the number of inputs per gate (resulting from less number of variablesperterm)*
- *Theminimizationwillreducecost,efficiencyandpower consumption.*
- **y(x+x`)=y.1=y**
- **y+xx`=y+0=y**
- **(x`y+xy`)=x⊕y**
- **(x`y`+xy)=(x⊕y)`**

# Minimum SOP and POS

- The*minimumsumofproducts(MSOP)*ofafunction,*f*,is a SOP representation of *f* that contains the fewest number of product terms and fewest number of literals of any SOP representation of *f*.

## MinimumSOPandPOS

- *f= (xyz +x`yz+ xy`z+.....)*

*Is called sum of products.*

*The + is sum operator which is an OR gate.*

*Theproduct suchasxyisanANDgateforthetwoinputsxandy.*

## *Example*

- *Minimizethefollowing Booleanfunctionusing*

***sum of products(SOP):***

- ***f(a,b,c,d) =∑m(3,7,11,12,13,14,15)***

abcd

3 0011

7 0111

11 1011

12 1100

13 1101

14 1110

15 1111

**a`b`cd a`bcd ab`cd abc`d` abc`d abcd` abcd**

*Example*

***f(a,b,c,d) =∑m(3,7,11,12,13,14,15)***

**=a`b`cd + a`bcd + ab`cd + abc`d`+ abc`d + abcd` + abcd**

**=cd(a`b` + a`b + ab`) + ab(c`d` + c`d + cd` + cd )**

**=cd(a`[b` + b] + ab`) + ab(c`[d` + d] + c[d` + d])**

**=cd(a`[1] + ab`) + ab(c`[1] + c[1])**

**=ab+ab`cd + a`cd**

**=ab+cd(ab` + a`)**

**=ab+ cd(a +a`)(a`+b`)**

= ab + a`cd + b`cd

= ab +cd(a` +b`)

# Minimum product of sums (MPOS)

- The *minimum product of sums (MPOS)* of a function, *f*, is a POS representation of *f* that contains the fewest number of sum terms and the fewest number of literals of any POS representation of *f*.
- The zeros are considered exactly the same as ones in the case of sum of product (SOP)

## *Example*

$f(a,b,c,d)=\prod M(0,1,2,4,5,6,8,9,10)$

$=\sum m(3,7,11,12,13,14,15)$

=[(a+b+c+d)(a+b+c+d`)(a+b`+c`+d`)

(a`+b+c`+d`)(a`+b`+c+d)(a`+b`+c+d`) (a`+b`+c`+d)(a`+b`+c`+d`)]

**Karnaugh Maps (K-maps)**

- Karnaugh maps -- A tool for representing Boolean functions of up to six variables.

K-maps are tables of rows and columns with entries represent 1`s or 0`s of SOP and POS representations

**Karnaugh Maps (K-maps)**

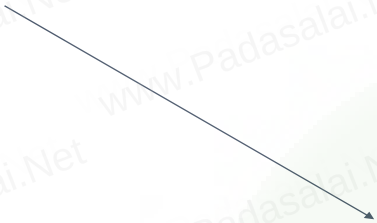
- *An n-variable K-map has $2^n$ cells with each cell corresponding to an n-variable truth table value.*
- K-map cells are labeled with the corresponding truth-table row.
- K-map cells are arranged such that adjacent cells correspond to truth rows that differ in only one bit position (*logical adjacency*).

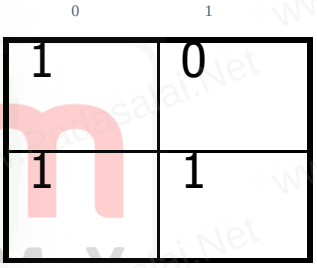
# Karnaugh Maps(K-maps)

- If $m_i$ is a minterm of $f$, then place a 1 in cell $i$ of the K-map.
- If $M_i$ is a maxterm of $f$, then place a 0 in cell $i$.
- If $d_i$ is a don't care of $f$, then place a $d$ or $x$ in cell $i$.

## *Examples*

- *Two variable K-map f(A,B)=∑m(0,1,3)=A`B`+A`B+AB*

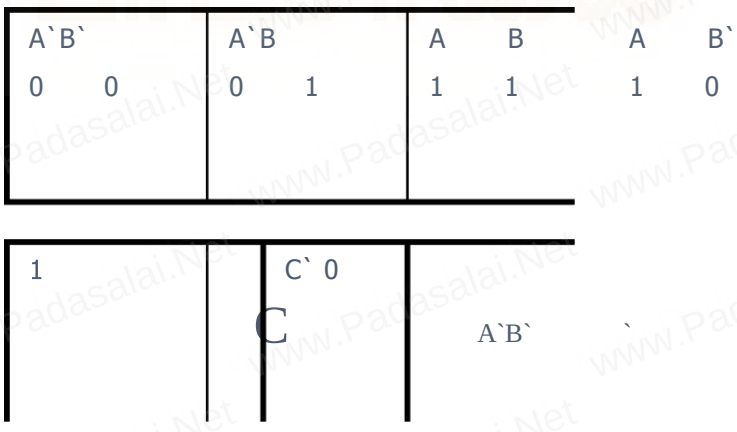





Three variable map

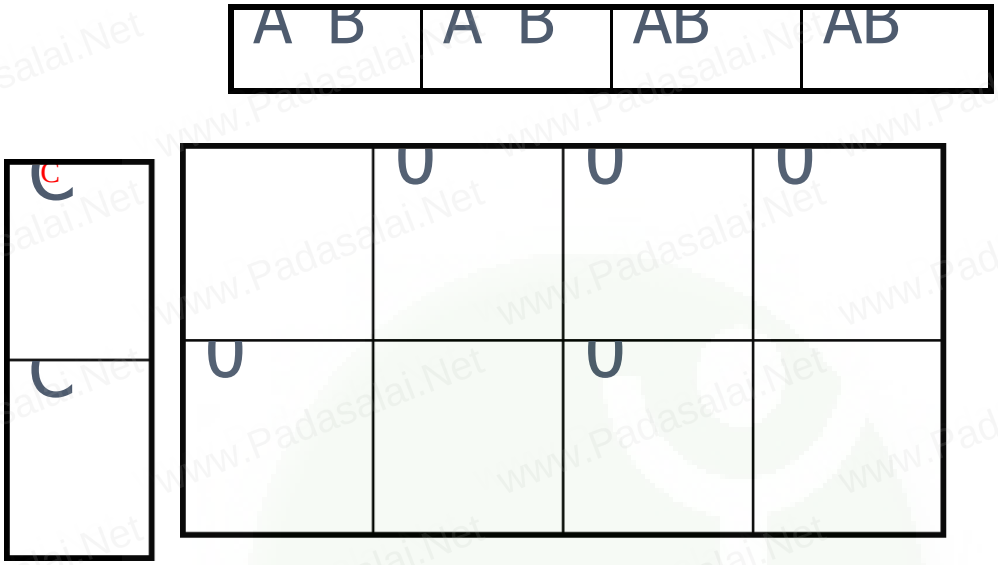
- *f(A,B,C) =∑m(0,3,5)=*

*A`B`C`+A`BC+AB`C*



## Maxtermexample

(A+B) (A+B`) (A`+B`) (A`+B)




*f(A,B,C)* =∏*M*(1,2,4,6,7)

=(A+B+C`)(A+B`+C)(A`+B+C))(A`+B`+C)(A`+B`+C`)

Notethatthecomplementsare(0,3,5)whicharethemintermsof the previousexample

Four variableexample

(a) Mintermform. (b)Maxtermform.

SimplificationofBooleanFunctions UsingK-maps

- K-mapcellsthatarephysicallyadjacentarealsologically adjacent. Also, cellsonanedgeofaK-maparelogically adjacenttocellsontheopposite edgeofthemap.
- If two logically adjacent cells both contain logical 1s, the two cells can be combinedtoeliminatethevariablethat hasvalue1inonecell'slabeland value0intheother.

# Simplification of Boolean Functions Using K-maps

- This is equivalent to the algebraic operation, $aP + a'P = P$ where P is a product term not containing $a$ or $a'$.
- A group of cells can be combined only if all cells in the group have the same value for some set of variables.

## Simplification Guidelines for K-maps

- Always combine as many cells in a group as possible. This will result in the fewest number of literals in the term that represents the group.
- Make as few groupings as possible to cover all minterms. This will result in the fewest product terms.
- Always begin with the largest group, which means if you can find eight members group is better than two four groups and one four group is better than pair of two-group.

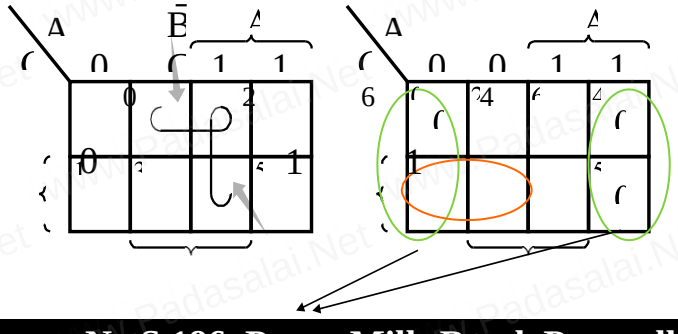
Example

Simplify f=A`BC`+ABC`+ABC **using;**

(a) Sum of minterms. (b) Maxterms.

- Each cell of an $n$-variable K-map has $n$ logically adjacent cells.

F`= B`+A`C F =

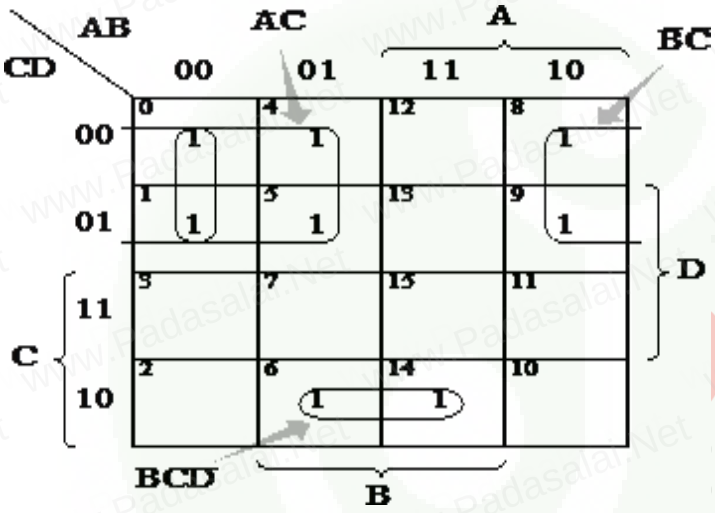


B(A+C`)

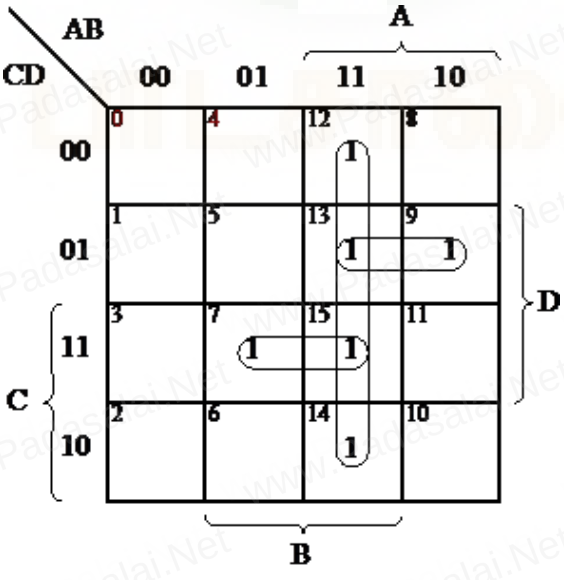
*a-* $f(A,B,C) = AB + BC'$

*b-* $f(A,B,C) = B(A + C)$

## Example




Example




(a)

# Don’t-care condition

- Minterms that may produce either 0 or 1 for the function.
- They are marked with an ´ in the K-map.
- This happens, for example, when we don’t input certain minterms to the Boolean function.
- These don’t-care conditions can be used to provide further simplification of the algebraic expression.

**(Example) F = A`B`C`+A`BC` +ABC`**

**d=A`B`C +A`BC + AB`C**

**F = A` + BC`**