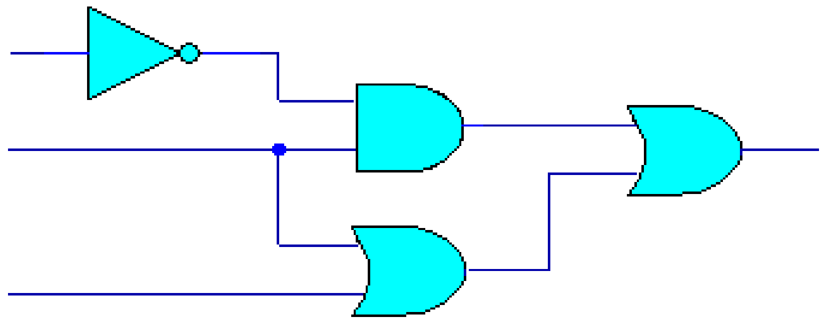
Combinational Logic



Objectives

1. Convert a logic expression into a sum-of-products expression.
2. Use the Karnaugh map as a tool to simplify and design logic circuits.
3. Explain the operation of both exclusive-OR and exclusive-NOR circuits.
4. Design logic circuits with and without the help of truth tables.
5. Identify and understand inhibit circuits.
6. Cite the basic characteristics of digital ICs
7. Understand the inherent operative differences between TTL and CMOS.

Introduction

Digital systems can be broadly classified into:

1. Combinational logic
2. Sequential Logic

With *Combinational Logic*, the output is purely dependant upon the input combinations at any moment in time.

Output = Function of the inputs at the instant

A *Combinational Logic* circuit has no memory characteristics, i.e. what happened previously have no effects on the present states.

Combinational Logic circuit uses only the logic gates seen in the previous chapter, i.e.

AND

OR

NOT

NAND

NOR, etc.

Sum - of - Products expressions

Boolean Expressions are typically expressed in what's called a Sum-of-Products form, that is:

Two or more AND terms (*products*) ‘Ored’ (i.e. logically *summed*) together

Examples are:

1) $Y = ABC + \overline{A}B\overline{C}$

2) $X = AB + \overline{A}B\overline{C} + \overline{C}\overline{D} + D$

3) $Z = A\overline{B} + \overline{C}\overline{D} + EFGK + HL$

Such Boolean expressions are typically obtained from a truth table by summing the input terms (input combinations) that produces a H or 1 at the output.

Another form of expressions that may be obtained from a truth table is a product-of-sums.
E.g.

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

Simplifying Logic Circuits

Typically, a logic expression obtained (e.g. from a truth table) may not be the simplest solution.

Prior to implementation, it is advisable to try and simplify the expression.

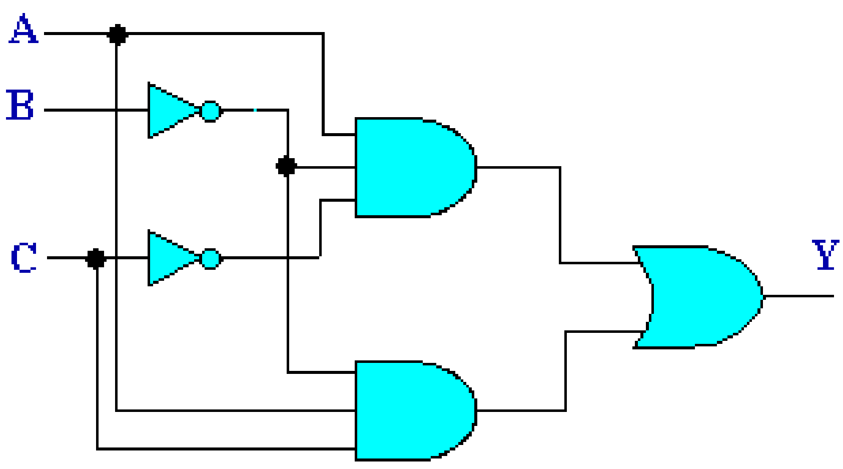
Example

C	B	A	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Using the ‘sum-of-products’ approach, the Boolean expression from the truth table is:

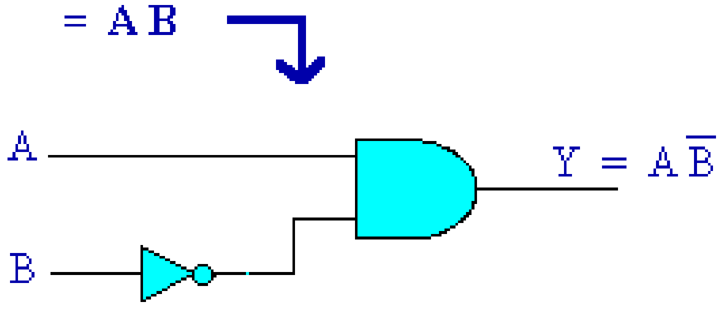
$$Y = \overline{C}\,\overline{B}\,A + C\,\overline{B}\,A$$

Implementing $Y = \overline{C}\,\overline{B}\,A + C\,\overline{B}\,A$:



If however the Boolean expression is first simplified, a simpler and better implementation is obtained:

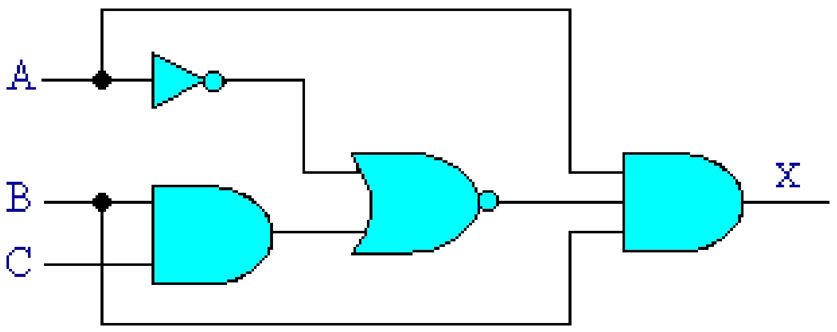
$$
\begin{aligned}
Y &= \overline{C}\,\overline{B}\,A + C\,\overline{B}\,A \\
&= A\overline{B}(\overline{C} + C) \\
&= A\overline{B}(1) \\
&= A\overline{B}
\end{aligned}
$$



Note that *not all* Boolean expressions can be simplified.

Try out this Example:

Given the circuit shown, obtain the Boolean expression, simplify it and then implement the resultant circuit:



Working from inputs to output, the Boolean expression for output X is:

$$X = AB(\overline{\overline{A} + BC})$$

Applying DeMorgan's theorems to the inverted term,

$$X = AB(A.[\overline{B} + \overline{C}])$$

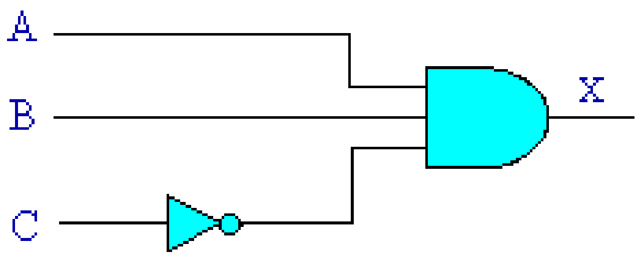
Expanding the terms,

$$X = ABA\overline{B} + ABA\overline{C}$$

Removing the redundant variable, and taking note that $B.\overline{B} = 0$

$$X = AB\overline{C}$$

i.e.



Algebraic Simplification

The general objective for any logic circuit construction is always to *simplify the Boolean expression, if possible*, prior to implementation.

Boolean Theorems can be used to simplify a logic expression.

Unfortunately, it is always not obvious which theorem(s) to use, or if an expression is already in its simplest form.

The general approach is:

1. Break down all large inverter signs by using DeMorgan's theorems.
2. Apply the appropriate theorem and remove any redundant terms.
3. Obtain expression in *sum-of-products* form.
4. Check for common factors and collect terms if required.

Step 4 is not required if a sum-of-products implementation is used.

Designing Combinational Logic Circuits

Generally, the tasks (i.e. steps) required in the design of combinational logic circuits is:

	Design Methodology
1.	Define the requirements
2.	Define Truth-Table
3.	Obtain Boolean Expression
4.	Simplify Boolean expression whenever possible
5.	Look at available logic ICs
6.	Modify circuit (if necessary)
7.	Build circuit
8.	Test and verify truth table

In a paper design, steps 5, 6 & 7 are *not* required.

Example

Design a logic circuit whose output is HIGH only when a majority of inputs A, B and C are HIGH.

1) Define requirements, i.e. 'Understand the problem'
   Output X = 1 when there are two or more 1's in the input combinations.

2) Create the truth table:

C	B	A	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

3) From the truth table, obtain the Boolean expression in a sum-of-products form

$$X = \overline{C}BA + C\overline{B}A + CB\overline{A} + CBA$$

Each product is an input term that produces a 1 in output X.

E.g. combination 011 produces X=1. This gives the product $\overline{C}BA$.

4. Simplify (if possible) the Boolean expression.

For the equation:

$$X = AB\overline{C} + A\overline{B}C + \overline{A}BC + ABC$$

Expanding a single ABC term to 3 terms,

$$X = AB\overline{C} + A\overline{B}C + \overline{A}BC + ABC + ABC + ABC$$

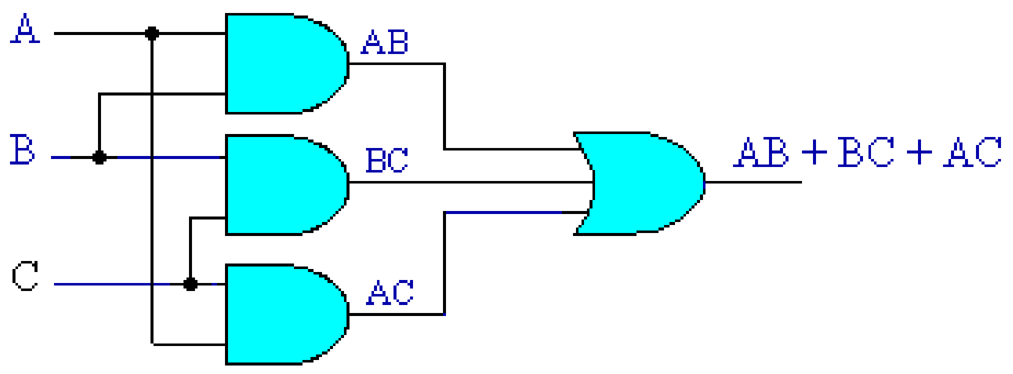
By collecting common terms,

$$X = AB(\overline{C}+C) + AC(\overline{B}+B) + BC(\overline{A}+A)$$

Removing redundant terms,

$$X = AB + AC + BC$$

5. Implement using basic gates:



Alternatively, the simplified equation could also be expressed as:

$X = AB + C.(A + B)$ *or*

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

Karnaugh Map Method

A Karnaugh Map (or K-Map) is a graphical representation of a truth table.

It expresses the relationship between the inputs and outputs of a logic system.

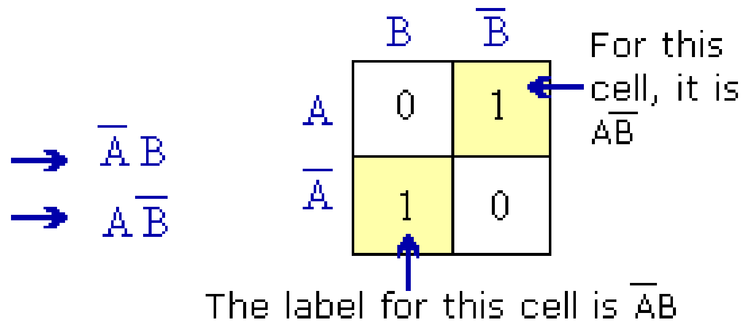
Can be used for simplification of Boolean expressions of up to 5 variables.

6 or more variables are best left to computers.

Example

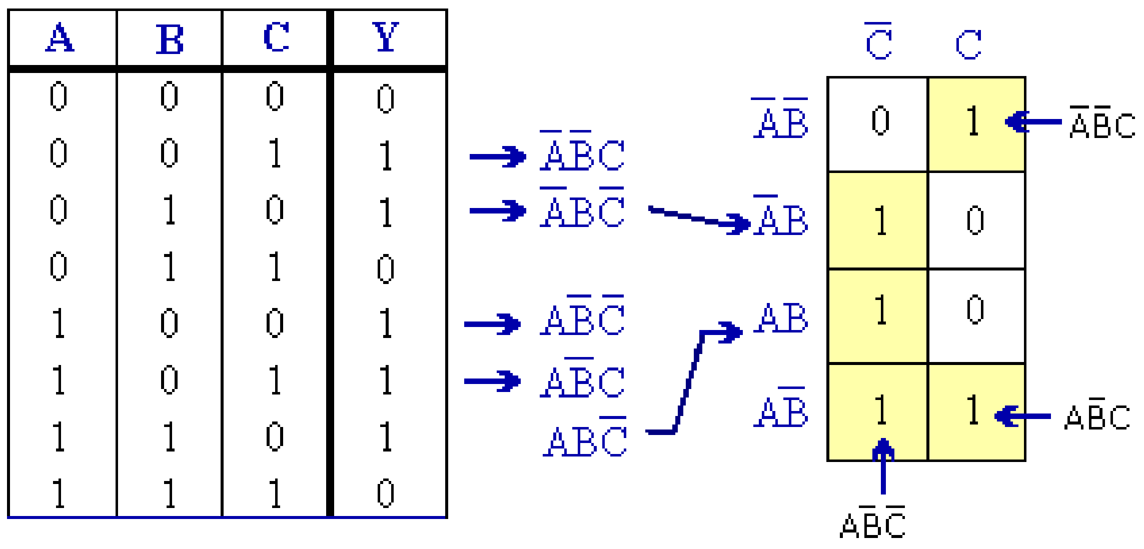
A 2 variable Truth table translates to a 2 variable K-map.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0


$\bar{A}B$
$A\bar{B}$
B
$\bar{B}$
A
$\bar{A}$
0
1
1
0
For this cell, it is $A\bar{B}$
The label for this cell is $\bar{A}B$

Each cell in the K-map corresponds to one combination in the truth table.

A 3 variable truth table (with 8 combinations) converts to an 8-cell K-map, as illustrated below:



A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

$$\begin{array}{c|cc} & \overline{C} & C \\ \hline \overline{A}\,\overline{B} & 0 & 1 \\ \overline{A}B & 1 & 0 \\ AB & 1 & 0 \\ A\overline{B} & 1 & 1 \end{array}$$

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

It is important to note that the labelling for the cells in a K-map must be as specified.

This sequence in terms of 0's and 1's is basically:

00 01 11 10

or in terms or variables,

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

Similarly, a <u>4 variable</u> truth table with 16 combinations translates to a 16-cell K-map.

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

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

Note that both row and column labelling sequence follow the order:

00 01 11 10

With this sequence, the label between adjacent cells on a row/column basis differs by only one variable.

When using the K-map, the main objective is always to:

- Loop (i.e. group) as many 1's as possible.
- The number of 1's in a loop must however be equal to 2^N (i.e. 2, 4, 8, 16 etc.)

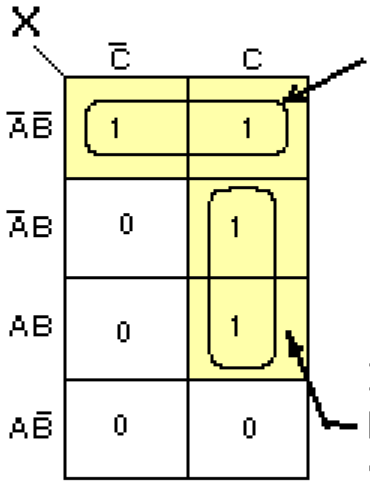
The expression derived from the K-map is in the form of a Sum-of-products that is suited for implementing a NAND-only circuit.

The K-map is labelled such that the variables corresponding to each cell differs by one from its adjacent neighbours.

Hence,

1.	looping 2 adjacent 1's eliminates one variable.
2.	looping 4 adjacent 1's eliminates two variables.
3.	looping 8 adjacent 1's eliminating three variables, and so on...

Looping a pair of adjacent 1s (i.e. **two 1's**) in a K-map eliminates the variable that appears in complemented and un-complemented form, i.e. with the ‘bar’ and without the ‘bar’.



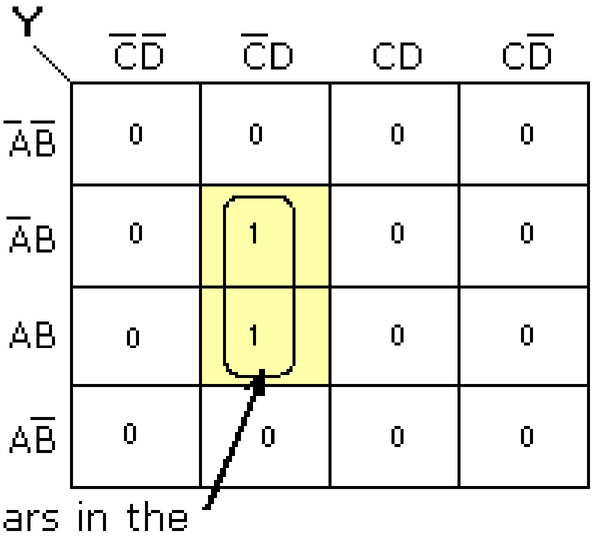
X	$\overline{C}$	C
$\overline{A}\,\overline{B}$	1	1
$\overline{A}B$	0	1
AB	0	1
$A\overline{B}$	0	0

For this pair, the C variable appears in both complemented & uncomplemented form & is eliminated. The term is thus $\overline{A}\,\overline{B}$

In this loop, the term is BC as A is the only variable with & without the bar.

Therefore

$$X = \overline{A}\,\overline{B} + BC$$

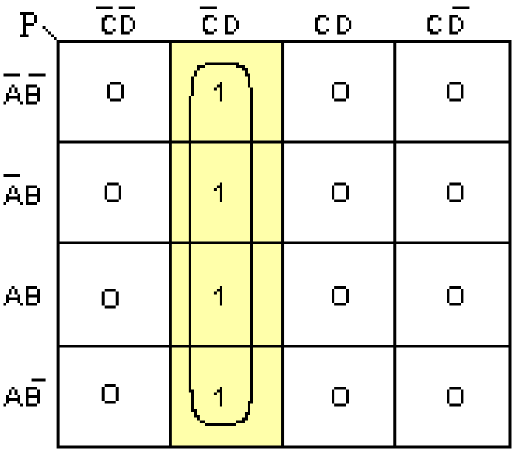


Y	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	0	0	0	0
$\overline{A}B$	0	1	0	0
AB	0	1	0	0
$A\overline{B}$	0	0	0	0

Only the A variable appears in the complemented & uncomplemented form & is thus eliminated.
Term is therefore $B\overline{C}D$

When **looping four 1's** (Quads) they must be:

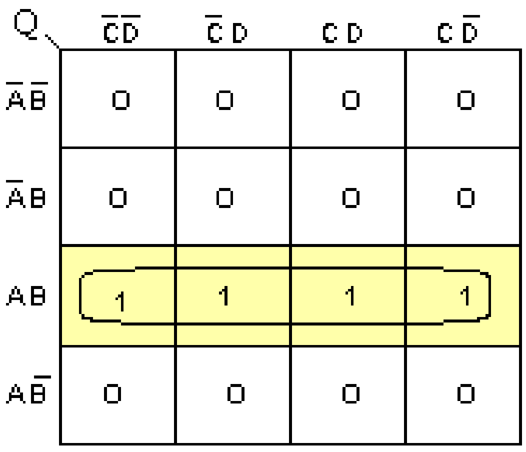
Vertically adjacent



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

$$P = \bar{C}D$$

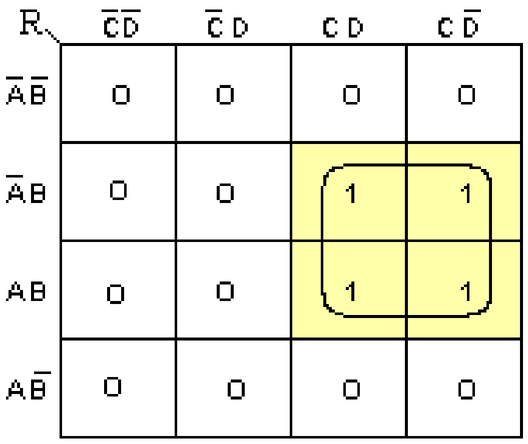
Horizontally adjacent



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

$$Q = AB$$

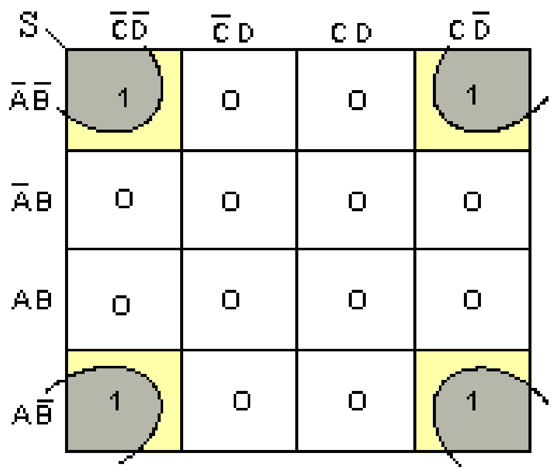
Square



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

What is the term for the square?

Four corners

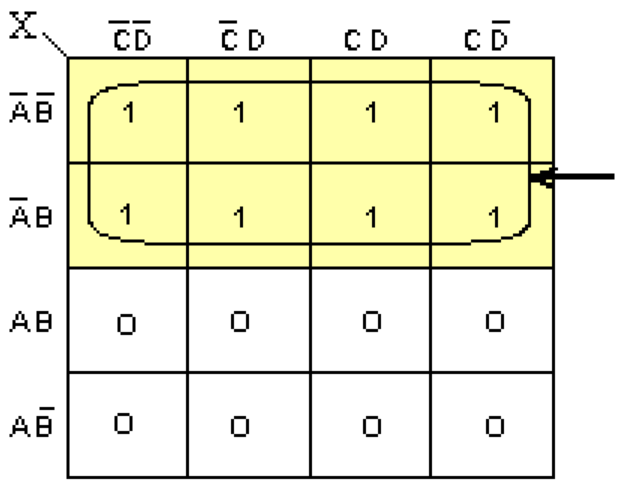


S	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	0	0	1
$\bar{A}B$	0	0	0	0
AB	0	0	0	0
$A\bar{B}$	1	0	0	1

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

Looping 4 1’s eliminates 2 variables.

Looping a **group of eight 1's** or an octet eliminates 3 variables.

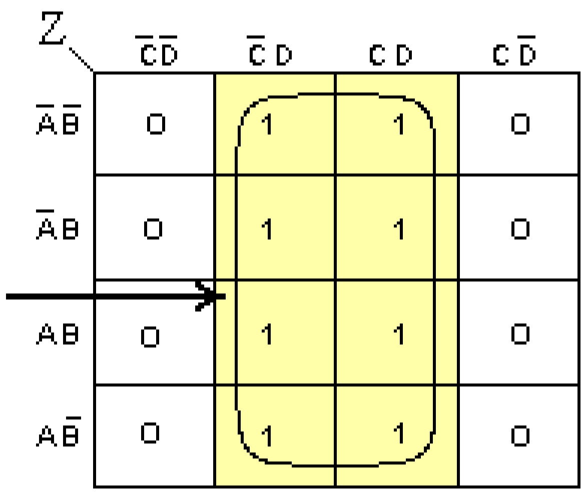


X	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	1	1	1	1
$\overline{A}B$	1	1	1	1
AB	0	0	0	0
$A\overline{B}$	0	0	0	0

The B C and D variables appear in both complemented & uncomplemented form in this loop of 8 1's. The term is thus $\overline{A}$

$$X = \overline{A}$$

Here the A B and C variables are eliminated leaving only the vaiable D in its uncomplemented form.



Z	$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\,\overline{B}$	0	1	1	0
$\overline{A}B$	0	1	1	0
AB	0	1	1	0
$A\overline{B}$	0	1	1	0

$$Z = D$$

Simplification process using K-Map

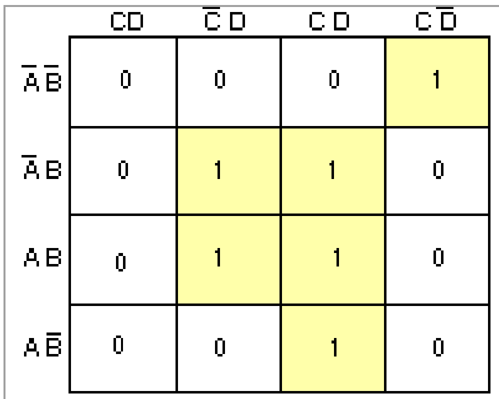
The procedures on using a K-map (for up to 4 variables) to simplify a Boolean expression may be stated as follows:

1.	Construct the K-map and fill in the 1's in the appropriate cells.
2.	Examine the K-map for isolated 1's, i.e. 1's that are not adjacent to any other 1's. These correspond to Boolean terms that cannot be simplified.
3.	Loop any octet (eight 1's), even if it contains some 1's that have already been looped.
4.	Loop any quads (four 1's) that contain one or more 1's that have not been looped.
5.	Loop any pairs (two 1's) necessary to include any 1's that have not yet been looped.
6.	Form the sum-of-product expression from the loops generated

Example 1:

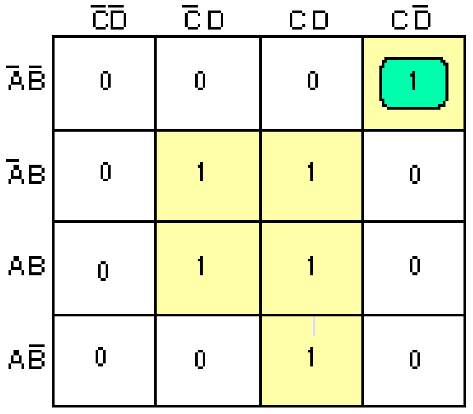
Given the following K-map, loop the 1's and obtain the simplified Boolean expression:

Step 1:
Construct the K-map and fill in the 1's.



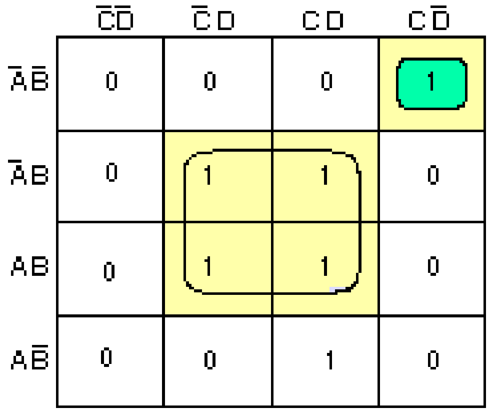
	$\bar C\bar D$	$\bar C D$	$C D$	$C\bar D$
$\bar A\bar B$	0	0	0	1
$\bar A B$	0	1	1	0
$A B$	0	1	1	0
$A\bar B$	0	0	1	0

Step 2:
Examine for isolated 1's



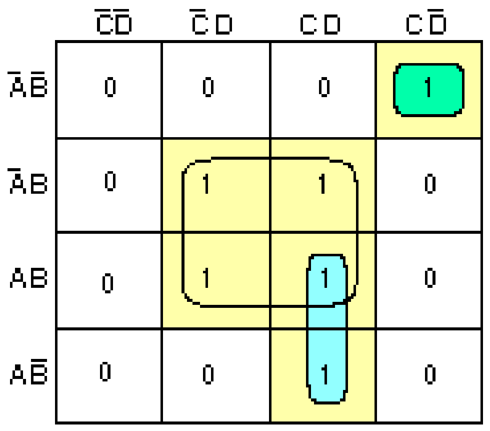
	$\bar C\bar D$	$\bar C D$	$C D$	$C\bar D$
$\bar A\bar B$	0	0	0	1
$\bar A B$	0	1	1	0
$A B$	0	1	1	0
$A\bar B$	0	0	1	0

Step 3: Loop 8 1's if possible, if not quads.



	$\bar C\bar D$	$\bar C D$	$C D$	$C\bar D$
$\bar A\bar B$	0	0	0	1
$\bar A B$	0	1	1	0
$A B$	0	1	1	0
$A\bar B$	0	0	1	0

Step 4: Loop any remaining pairs.



	$\bar C\bar D$	$\bar C D$	$C D$	$C\bar D$
$\bar A\bar B$	0	0	0	1
$\bar A B$	0	1	1	0
$A B$	0	1	1	0
$A\bar B$	0	0	1	0

Step 5: Form the S-o-P expression from the loops

Term for quad is BD

Term for pair is ACD

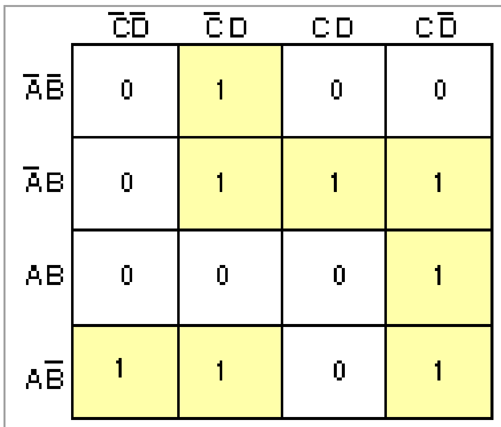
Isolated term is $\bar A\bar B C\bar D$

Hence, the S-o-P is

$$Y = BD + ACD + \bar A\bar B C\bar D$$

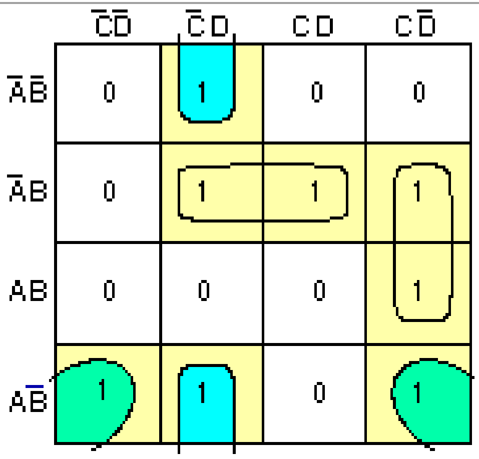
Example 2:

Given the K-Map as shown, loop the 1's and obtain the simplified Boolean expression.

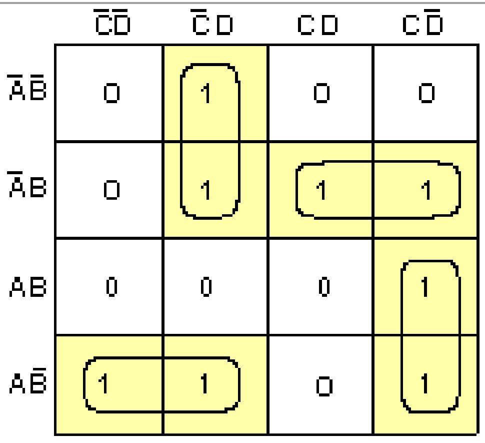


	$\bar C\bar D$	$\bar C D$	$C D$	$C\bar D$
$\bar A\bar B$	0	1	0	0
$\bar A B$	0	1	1	1
$A B$	0	0	0	1
$A\bar B$	1	1	0	1

In this example there are no octets or quads to be looped; only pairs can be formed.



(a)



(b)

Both methods of looping the 1's are equally valid and equally good although they do lead to different Boolean expressions.

Expression for (a) is $A\bar B\bar D + \bar B\bar C D + \bar A B D + B C\bar D$

Expression for (b) is $A\bar B\bar C + \bar A\bar C D + \bar A B C + A C\bar D$

An Exercise:

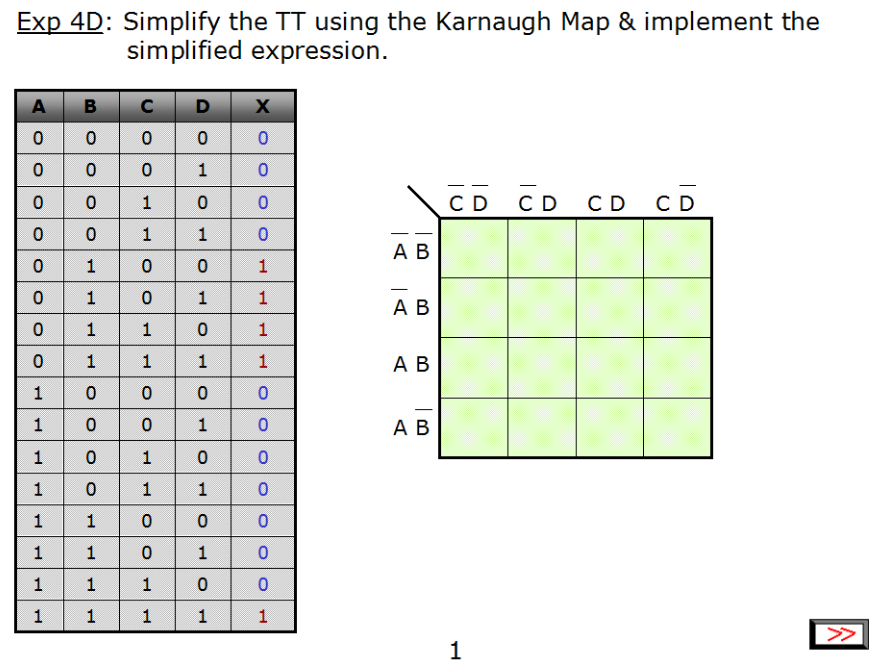
Given the K-map as shown, loop the 1's appropriately and hence obtain the simplified Boolean expressions.

	$\bar C \bar D$	$\bar C D$	$C D$	$C \bar D$
$\bar A \bar B$			1	
$\bar A B$			1	
$A B$	1	1	1	1
$A \bar B$			1	

What is your **Answer**?

Example 3:

Power-point ex. on using K-map. Double click on it:



Exp 4D: Simplify the TT using the Karnaugh Map & implement the simplified expression.

A	B	C	D	X
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

	$\bar C \bar D$	$\bar C D$	$C D$	$C \bar D$
$\bar A \bar B$				
$\bar A B$				
$A B$				
$A \bar B$				

1

>>

Don't Cares in K-Map

In some logic systems, there are certain input combinations for which there are no specific output levels.

Usually these input combinations will never occur.

If these input combinations will never occur, we "Don't care" what their corresponding output states will be.

To indicate a "Don't care" output state an 'X' is used in place of the 1 or 0.

When using the K-map, a "don't care" can be treated as logic 1 or logic 0 - whichever is more conducive to circuit minimization.

The classic example involving "Don't cares" is a logic system that deals with the BCD code. With BCD, input combinations of 1010_2 to 1111_2 do not occur and are normally treated as "Don't cares".

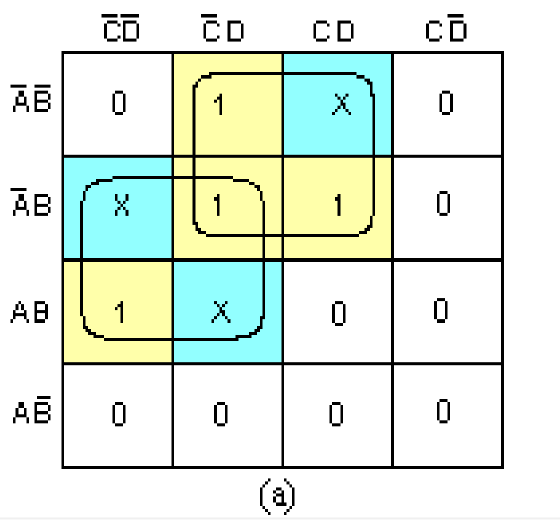
Examples:

In example (a), the three "Don't cares" (X’s) would be considered as 1's as this will lead to better overall simplification.

Two quads can thus be formed instead of 2 pairs & an isolated 1.

Resultant expression is thus:

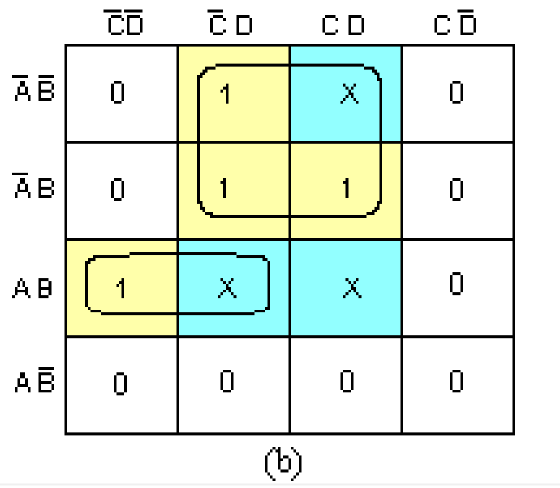
$$B\overline{C} + \overline{A}D$$


$\overline{C}\,\overline{D}$ $\overline{C}D$ CD $C\overline{D}$
$\overline{A}\,\overline{B}$ 0 1 X 0
$\overline{A}B$ X 1 1 0
AB 1 X 0 0
$A\overline{B}$ 0 0 0 0
(a)

In (b), there is no advantage in treating the "Don't care" cell that represents input combination ABCD, as logic 1.

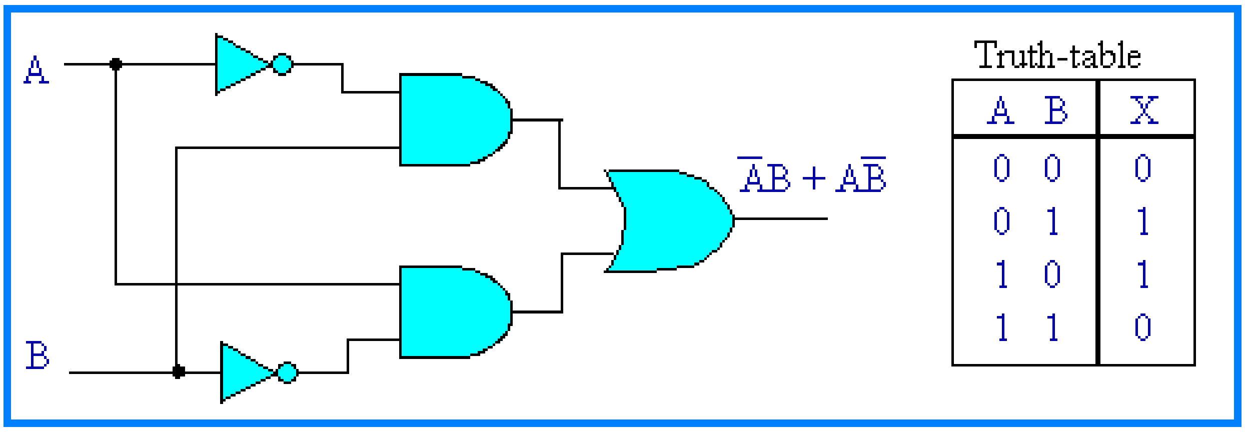
Two of the don’t cares which could be looped with the adjacent 1’s are treated as 1’s. A simpler expression is thus obtained than if the don’t cares were treated as 0’s.

Thus: $\overline{A}D + AB\overline{C}$


$\overline{C}\,\overline{D}$ $\overline{C}D$ CD $C\overline{D}$
$\overline{A}\,\overline{B}$ 0 1 X 0
$\overline{A}B$ 0 1 1 0
AB 1 X X 0
$A\overline{B}$ 0 0 0 0
(b)

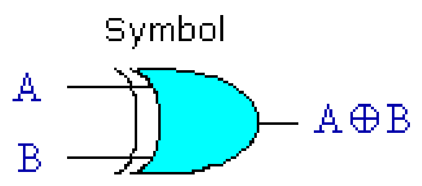
Exclusive - OR Circuit

- This circuit's output goes *HIGH* whenever the two inputs are at opposite levels.



A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

The XOR circuit is usually considered as a basic logic gate and is given the symbol and Boolean equation as shown:

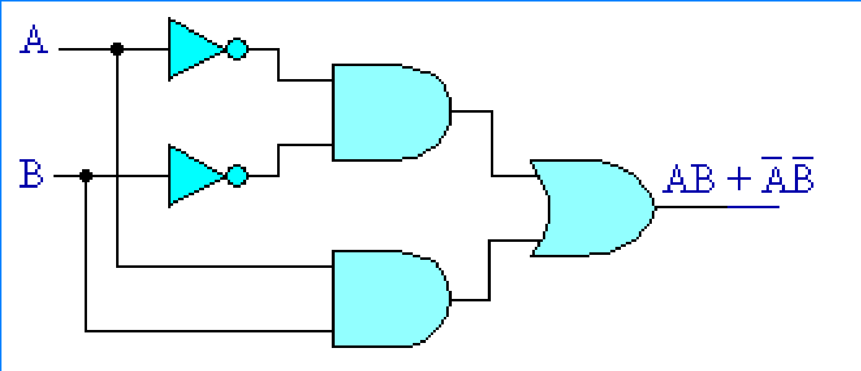


Several ICs are available that contain EX-OR gates. These are listed below as *quad* EX-OR chips because each IC contains four EX-OR gates.

7486	Quad EX-OR	(TTL family)
74C86	Quad EX-OR	(CMOS family)
4HC86	Quad EX-OR	(high-speed CMOS)

Exclusive - NOR Circuit

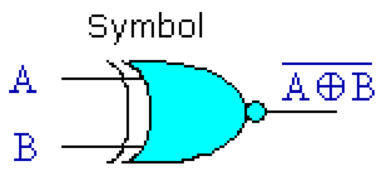
- This circuit's output goes *HIGH* whenever the two inputs are at the same logic levels.



Truth-table

A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

Like the XOR circuit, the XNOR circuit is also normally considered as a basic gate with a symbol and Boolean expression of its own:



Several ICs are available that contain EX-NOR gates. These are listed below as *quad* EX-OR chips because each IC contains four EX-NOR gates.

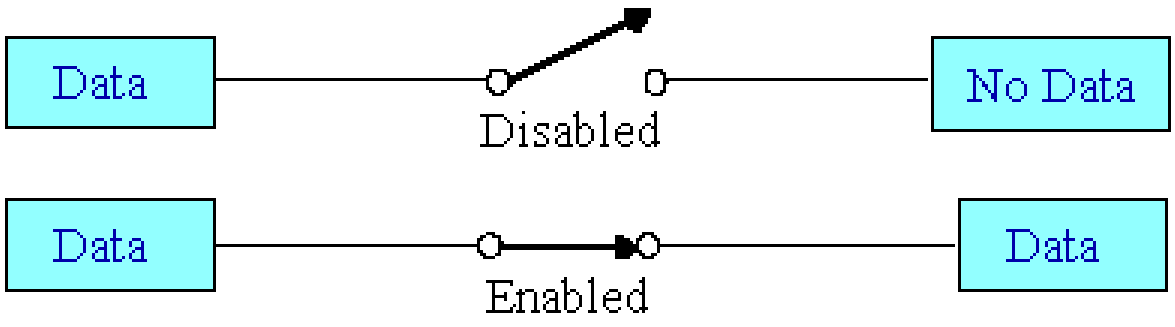
74LS266	Quad EX-NOR	(TTL family)
74C266	Quad EX-NOR	(CMOS family)
74HC266	Quad EX-NOR	(high-speed CMOS)

Inhibit Circuits (Enable/Disable Circuits)

An inhibit (or enable/disable) circuit is a digital circuit which is used for the purpose of controlling the flow of data.

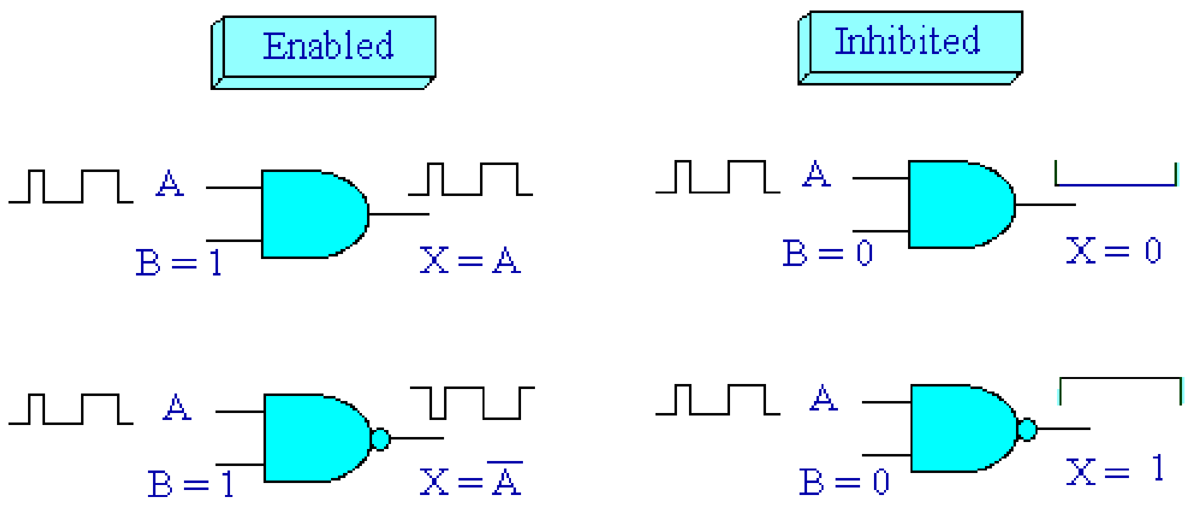
Any logic gate can be used as an inhibit circuit.

In general, an inhibit circuit has	- a data input - a control (enable) input - a data output
Its basic operation is to	- stop data flow when disabled. - allows data to pass through when enabled.



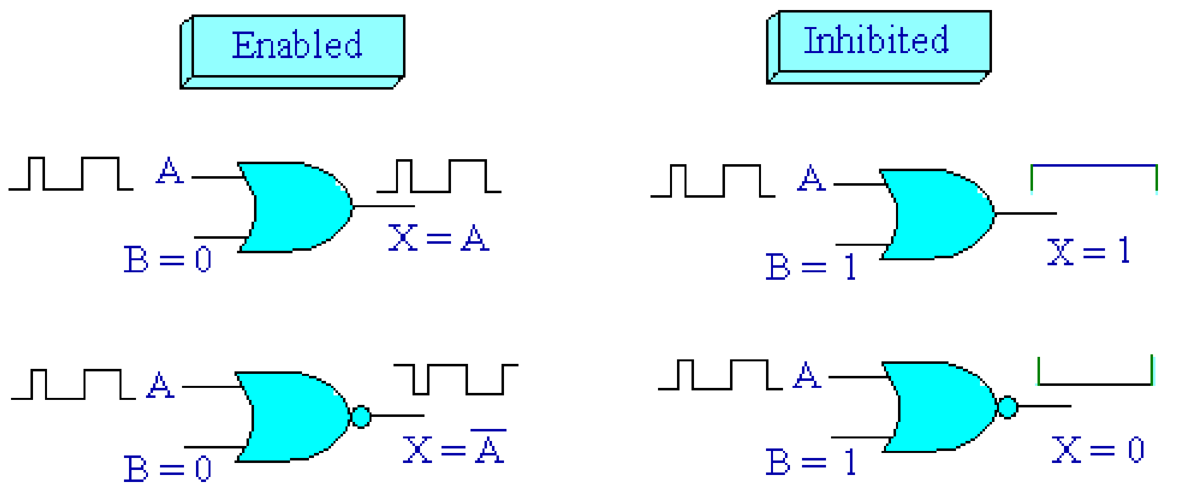
Is conceptually equivalent to an electronic switch.

Inhibit circuit using AND and NAND gates.



In both circuits, the output is enabled when the control input B = 1. In the case of the NAND gate, the enabled output signal is also inverted.

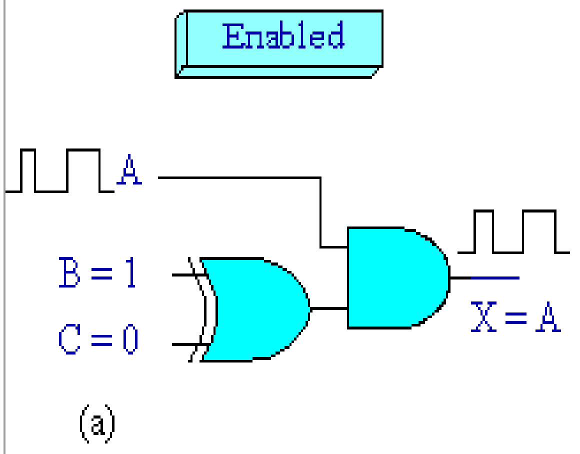
Inhibit circuit using OR and NOR gates.

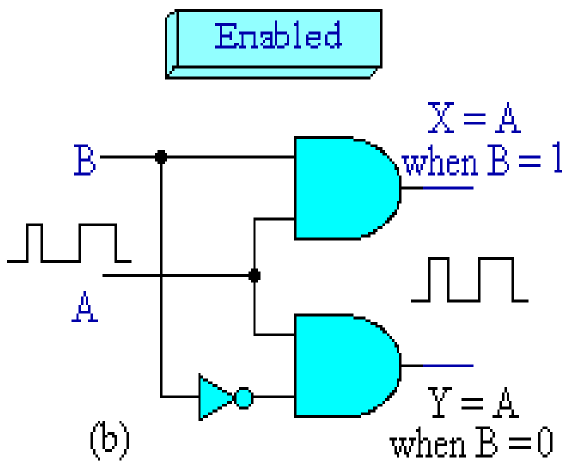


In both circuits, the output is enabled when the control input B = 0. In the case of the NOR gate, the enabled output signal is also inverted.

Other examples of inhibit circuits are:

(a) An inhibit circuit with 2 control inputs:
In this circuit, inputs B and C are the control inputs. In order for the output AND gate to be enabled, B != C. i.e.,
For X = A,
B must = 1 and C = 0,
or
B = 0 and C must = 1.





(b) An inhibit circuit with two outputs. With a NOT gate between the 2 control inputs of the AND gates, A can only appear at the output of the enabled gate, i.e.
when B = 1, X = A and,
when B = 0, Y = A .

NB: This circuit is actually 1-2 Demultiplexer.