Chapter 5

Logic Circuit Simplification:

Boolean algebra and DeMorgan's Theorem

Introduction

Boolean algebra is the mathematics of digital systems. It is used to help analyze a logic circuit and express its operation mathematically. Therefore, Boolean algebra can be used to simplify or to find an equivalent logic circuit. The main operations of Boolean algebra are the conjunction and denoted as ∧, Boolean addition, the disjunction *or* denoted as ∨, Boolean multiplication, and the negation *not* denoted as ¬.

Laws and Rules of Boolean algebra

Laws of Boolean Algebra

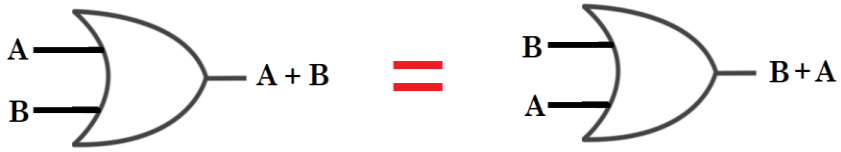
The basic laws of Boolean algebra are the ***commutative law, associative law,*** and ***distribute law.***

1) Commutative law of addition and multiplication

This law states that the order in which the Boolean variables are ORed or ANDed do not change the result.

OR Commutative law

$$A + B = B + A$$



AND Commutative law

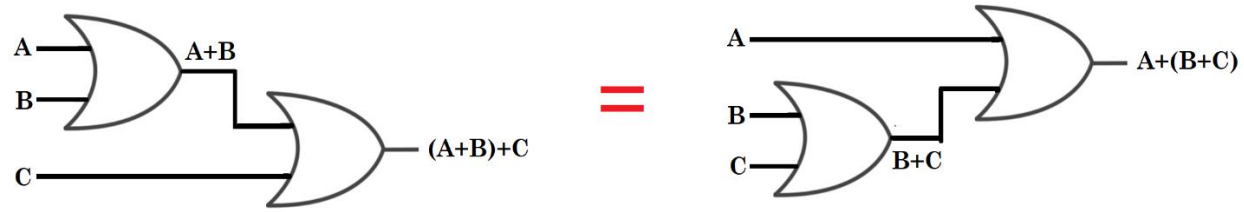
$$A \cdot B = B \cdot A$$

2) Associative laws of addition and multiplication

The law states that when ORed or ANDed more than two Boolean variables, the result is the same regardless of the grouping of the variables.

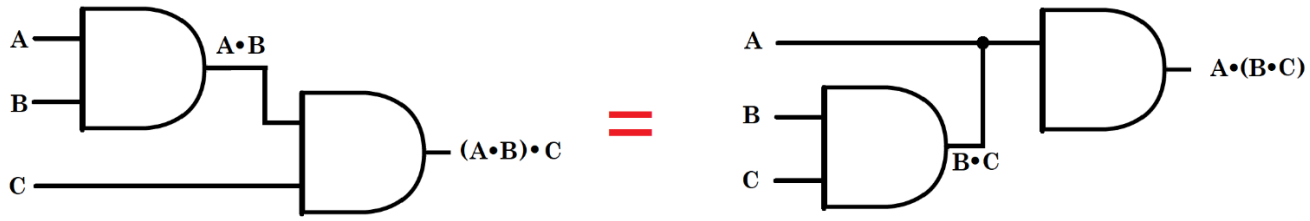
OR Associative law

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



AND Associative law

$$(\boldsymbol{A}\cdot\boldsymbol{B})\ \cdot\ \boldsymbol{C}\ =\ \boldsymbol{A}\ \cdot\ (\boldsymbol{B}\cdot\boldsymbol{C})$$

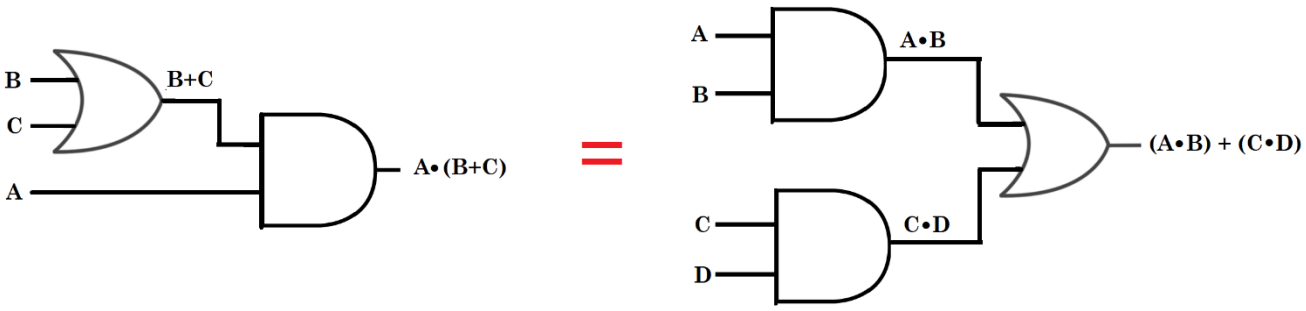


3) Distributive law

This law permits the multiplying or factoring out of an expression

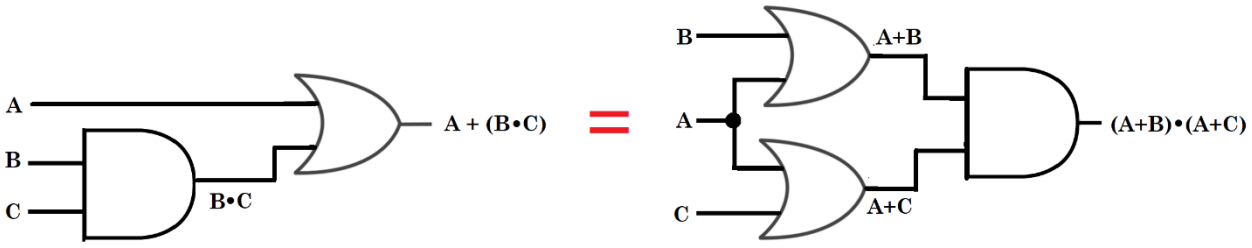
OR Distributive law

$$\boldsymbol{A}\cdot(\boldsymbol{B}+\boldsymbol{C})\ =\boldsymbol{A}\ \cdot\boldsymbol{B}+\boldsymbol{A}\cdot\boldsymbol{C}$$



AND Distributive law

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

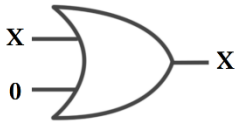


Laws of Boolean Algebra

4) Boolean Addition

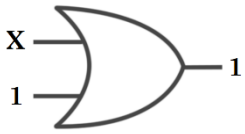
Boolean addition is equivalent to the OR operation and the basic rules are:

OR operation: case a ➔ X + 0 = X



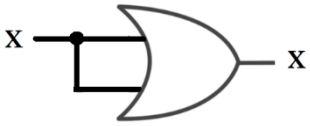
If one input is set to 0 and input X is 1, the OR output is 1. Otherwise, if input X is changed to 0, the OR is 0. Therefore, 0 added to anything does not affect its value.

OR operation: case b ➔ X + 1 = 1



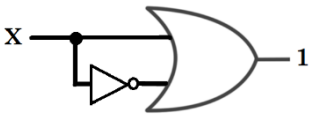
If one input is set to 1 and input X is 1, the OR output is 1. But, if input X is changed to 0, the OR output is also 1. Therefore, if any variable is ORed with 1, the result will always be 1.

OR operation: case c ➔ X+X = X



If both inputs are connected to X, the OR output depends on the variable X.

OR operation: case d ➔ $X + \bar{X} = 1$



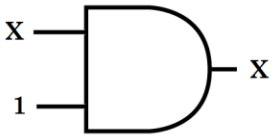
If one input is connected to X and the other input to the inverse of X, when X is set to 1, the inverse of X is 0, then the OR output is 1, and vice-verse. Therefore, the OR output of input X with its inverse, $\bar{X}$, is always 1.

5) Boolean multiplication

Boolean multiplication is equivalent to the AND operation and the basic rules are:

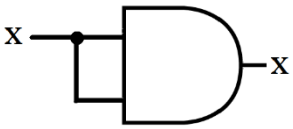
AND operation: case a ➔ $X \cdot 0 = 0$

If one input is set to 0 and input X is 1 or 0, the AND output is 0. Therefore, any variable ANDed with 0, the result must be 0.



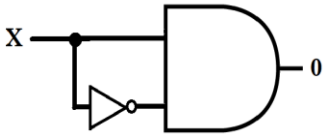
AND operation: case b ➔ $X \cdot 1 = X$

If one input is set to 1 and input X is 0, the AND output is 0. If input X is changed to 1, the AND output is 1. Therefore, if input 1 is ANDed with a variable, the output depends on the variable.



AND operation: case c ➔ $X \cdot X = X$

If both inputs are connected to X, the AND output depends on the variable X.

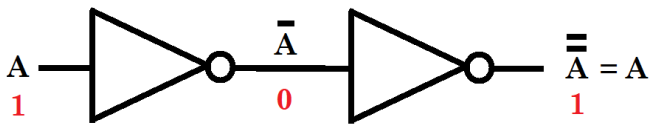


AND operation: case d ➔ $X \cdot \overline{X} = 0$

If one input is connected to X and the other input to the inverse of X; when X is set to 1, the inverse of X is 0, then the AND output is 0, and vice-verse. Therefore, the AND output of input X with its inverse, $\overline{X}$, is always 0.

6) Boolean Inversion

NOT operation: Even number of inversions of a variable is always equal to the variable. Odd number of inversions of a variable is the inverse of the variable.



$\overline{\overline{B}} = B$ $\overline{\overline{C}} = \overline{C}$

7) Special case: factoring distributive law: in an expression, if a variable is by itself and the same variable is in another term/s, the expression is equal to the variable

$$\boldsymbol{A + AB = A}$$

Proof:

$A + AB = A(1 + B) \quad \rightarrow Factoring\ distributive\ law$

$= A(1) \quad \rightarrow 1 + B = 1\ (OR\ operation)$

$= A \quad \rightarrow A(1) = A\ (AND\ operation)$

8) Special case: AND distributive law: in an expression, if a variable is by itself and the inverse of the variable in another term, the expression cancels the inverse of the variable in the another term

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

Proof:

$A + \bar{A}B = (A + \bar{A}) \cdot (A + B) \rightarrow \ AND\ distributive\ law$

$= 1 \cdot (A + B) \quad \rightarrow AND\ operation$

$= \boldsymbol{A + B}$

9) Special case: distributive law: if expressions in parenthesis have one same variable, the expression is equal to the variable plus the product of the other terms in parenthesis

$$\boldsymbol{(A + B) \cdot (A + C) = A + BC}$$

Proof:

$(A + B)(A + C) \rightarrow apply\ distirbutive\ law$

$= \boldsymbol{AA} + AC + AB + BC \rightarrow apply\ AND\ operation\ to\ AA = A$

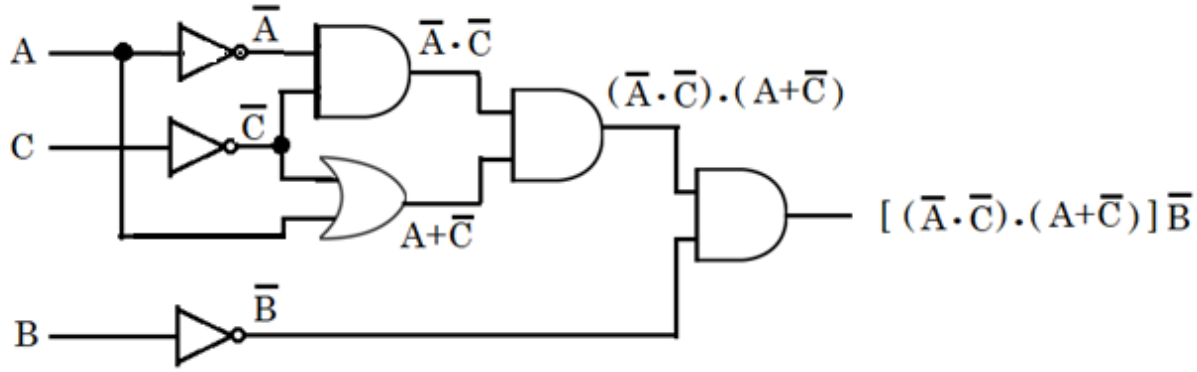
$= \boldsymbol{A + AC + AB} + BC \ \rightarrow apply \boldsymbol{factoring\ distributive\ law\ to}\ A + AC + AB = A$

$= \boldsymbol{A + BC}$

Simplification using Boolean algebra

One application of Boolean algebra is to reduce a circuit equation to its simplest form or change its form to a more convenient or efficient equivalent logic expression.

Example) use Boolean algebra laws and/or rules to simplify the following expression:



$[\bar{A}\bar{C}(A+\bar{C})]\bar{B}$

$[\bar{A}\bar{C}(A+\bar{C})]\bar{B}$ → *apply distributive law to* $\bar{A}\bar{C}(A+\bar{C})$

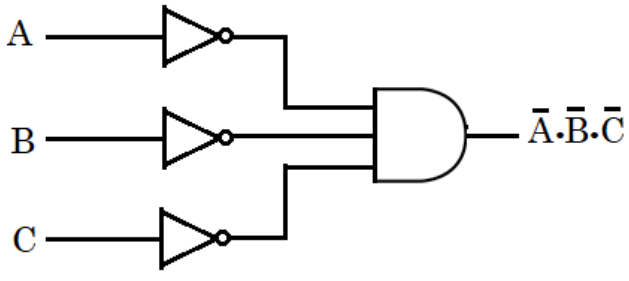
$= [\bar{A}\bar{C}A + \bar{A}\bar{C}\bar{C}]\bar{B}$ → *apply AND operation to* $\bar{A}\bar{C}A = 0.\bar{C}$

$= [0 + \bar{A}\bar{C}\bar{C}]\bar{B}$ → *apply AND operation to* $\bar{A}\bar{C}\bar{C} = \bar{A}\bar{C}$

$= (0 + \bar{A}\bar{C})\bar{B}$ → *apply OR operation to* $0 + \bar{A}\bar{C} = \bar{A}\bar{C}$

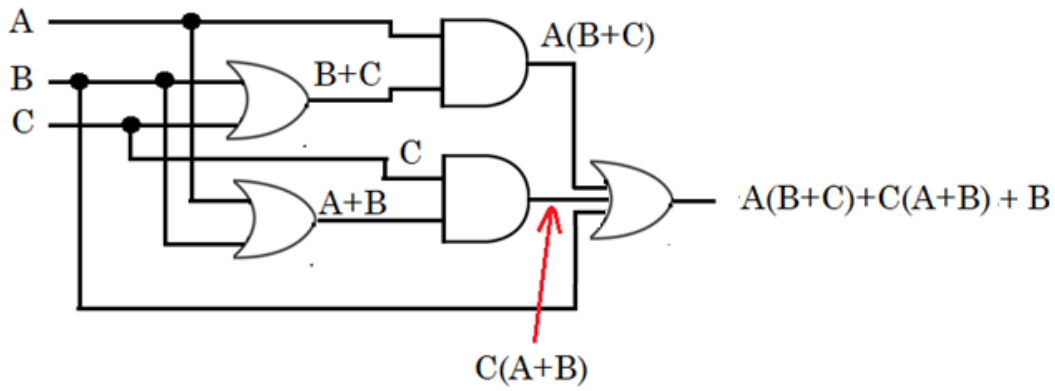
$= (\bar{A}\bar{C})\bar{B}$ → *AND distributive law*

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



Simplified equivalent circuit

Example) use Boolean algebra laws and/or rules to simplify the following expression:



$$\boldsymbol{A(B+C)+C(A+B)+B}$$

$$= AB + AC + AC + CB + B \rightarrow apply\ OR\ operation\ AC + AC = AC$$

$$= AB + AC + CB + B$$

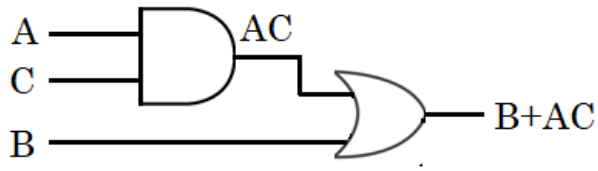
$$= (AB + CB + B) + AC \quad \rightarrow \quad OR\ associative\ law$$

$$= B(1 + A + C) + AC \quad \rightarrow \quad factoring\ distributive\ law: factor\ out\ B$$

$$= B(1 + A + C) + AC \quad \rightarrow \quad apply\ OR\ operation \quad 1 + A + C$$

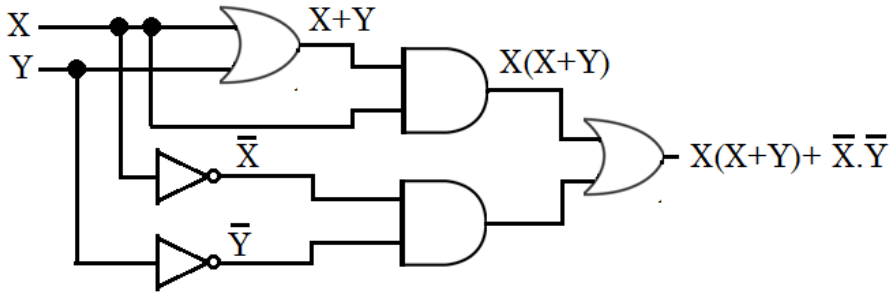
$$= B(1) + AC \quad \rightarrow \quad apply\ AND\ operation \quad B(1)$$

$$\boldsymbol{= B + AC}$$



Simplified equivalent circuit

Example) use Boolean algebra laws and/or rules to simplify the following expression:



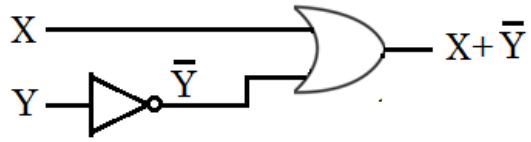
$X(X+Y)+\bar{X}\bar{Y}$ $\rightarrow$ *apply distributive law to* $X(X+Y)=XX+XY$

$=\boldsymbol{XX}+XY+\bar{X}\bar{Y}$ $\rightarrow$ *apply AND operation to* $\boldsymbol{XX}=\boldsymbol{X}$

$=\boldsymbol{X}+\boldsymbol{XY}+\bar{X}\bar{Y}$ $\rightarrow$ *special case*: *factoring distributive law to* $\boldsymbol{X}+\boldsymbol{XY}=X$

$=\boldsymbol{X}+\boldsymbol{\bar{X}\bar{Y}}$ $\rightarrow$ *special case*: *AND distribuitve law to* $\boldsymbol{X}+\boldsymbol{\bar{X}\bar{Y}}=X+\bar{Y}$

$=\boldsymbol{X}+\boldsymbol{\bar{Y}}$



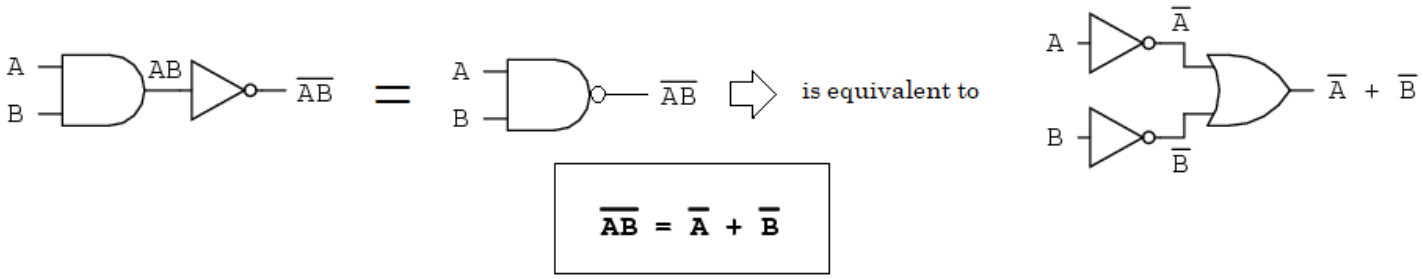
Simplified equivalent circuit

DeMorgan's Theorem

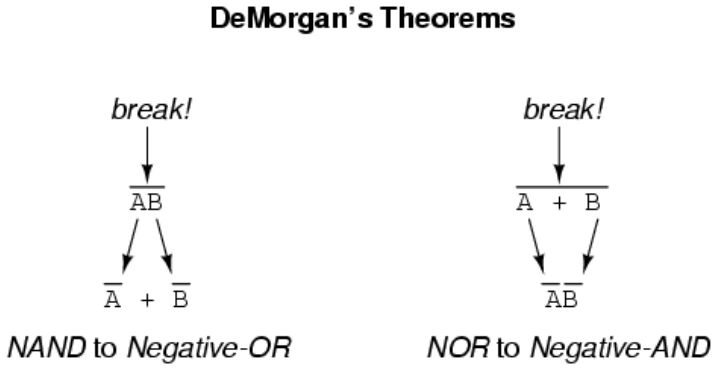
(All About Circuits, 2019)

A mathematician named DeMorgan developed a pair of important rules regarding group complementation in Boolean algebra. By *group* complementation, I'm referring to the complement of a group of terms, represented by a long bar over more than one variable.

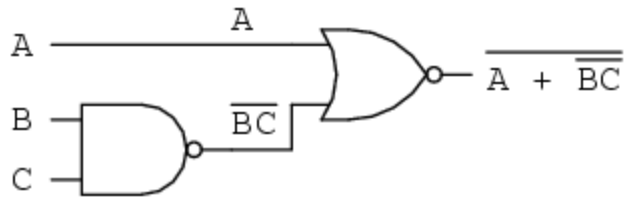
You should recall from the chapter on logic gates that inverting all inputs to a gate reverses that gate's essential function from AND to OR, or vice versa, and also inverts the output. So, an OR gate with all inputs inverted (a Negative-OR gate) behaves the same as a NAND gate, and an AND gate with all inputs inverted (a Negative-AND gate) behaves the same as a NOR gate. DeMorgan's theorems state the same equivalence in "backward" form: that inverting the output of any gate results in the same function as the opposite type of gate (AND vs. OR) with inverted inputs:



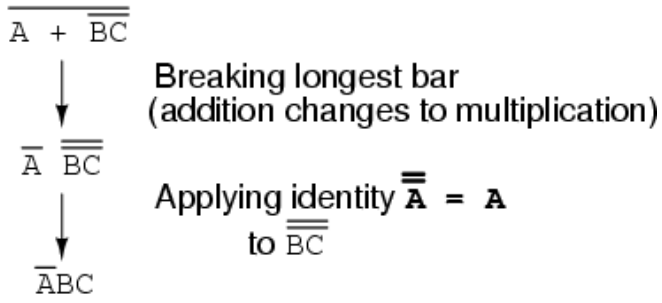
DeMorgan's theorem may be thought of in terms of *breaking* a long bar symbol. When a long bar is broken, the operation directly underneath the break changes from addition to multiplication, or vice versa, and the broken bar pieces remain over the individual variables. To illustrate:



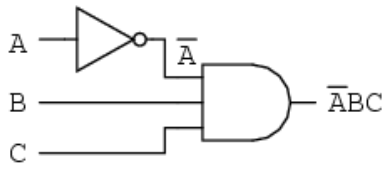
When multiple "layers" of bars exist in an expression, you may only break *one bar at a time*, and it is generally easier to begin simplification by breaking the longest (uppermost) bar first. To illustrate, let's take the expression (A + (BC)')' and reduce it using DeMorgan's Theorems:



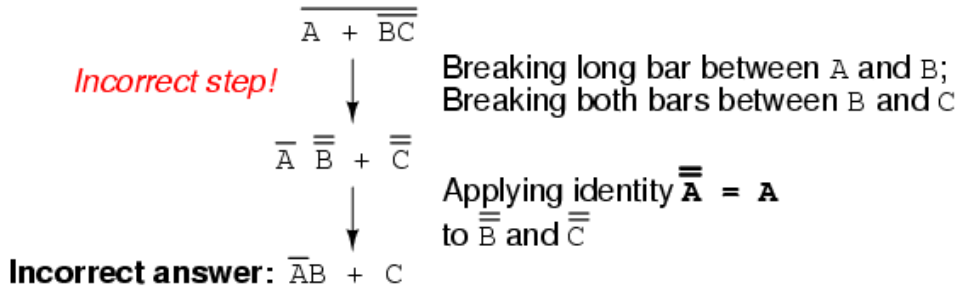
Following the advice of breaking the longest (uppermost) bar first, I'll begin by breaking the bar covering the entire expression as a first step:



As a result, the original circuit is reduced to a three-input AND gate with the A input inverted:

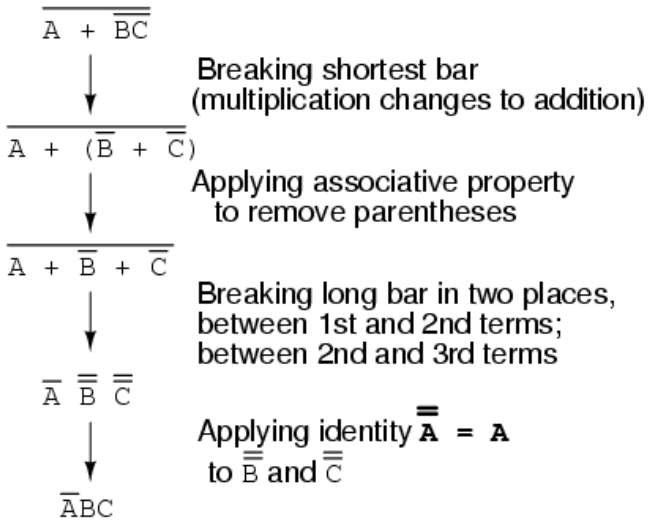


You should *never* break more than one bar in a single step, as illustrated here:



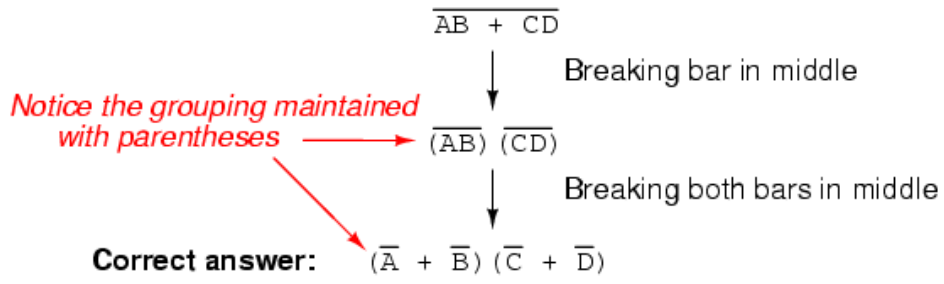
As tempting as it may be to conserve steps and break more than one bar at a time, it often leads to an incorrect result, so don't do it!

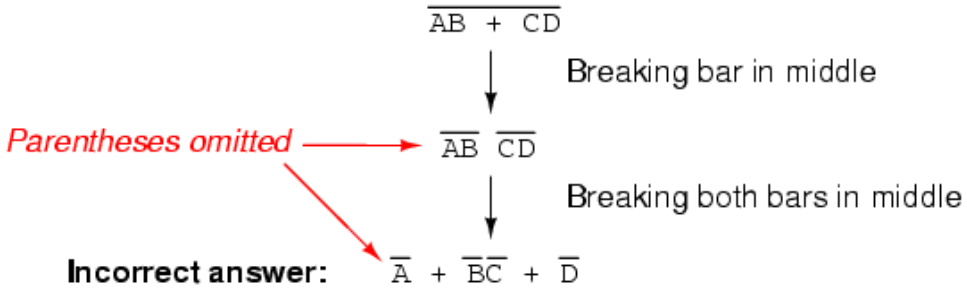
It is possible to properly reduce this expression by breaking the short bar first, rather than the long bar first:



The end result is the same, but more steps are required compared to using the first method, where the longest bar was broken first. Note how in the third step we broke the long bar in two places. This is a legitimate mathematical operation, and not the same as breaking two bars in one step! The prohibition against breaking more than one bar in one step is *not* a prohibition against breaking a bar in more than one place. Breaking in more than one *place* in a single step is okay; breaking more than one *bar* in a single step is not.

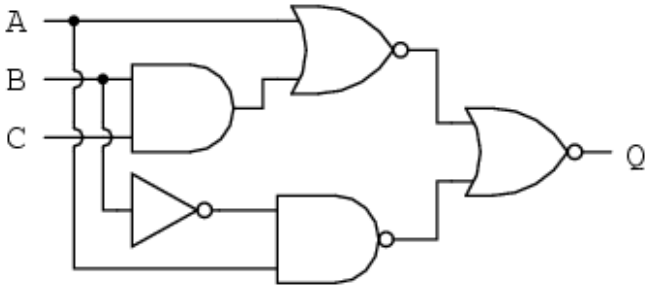
You might be wondering why parentheses were placed around the sub-expression $\bar{B} + \bar{C}$, considering the fact that I just removed them in the next step. I did this to emphasize an important but easily neglected aspect of DeMorgan's theorem. Since a long bar functions as a grouping symbol, the variables formerly grouped by a broken bar must remain grouped lest proper precedence (order of operation) be lost. In this example, it really wouldn't matter if I forgot to put parentheses in after breaking the short bar, but in other cases it might. Consider this example, starting with a different expression:



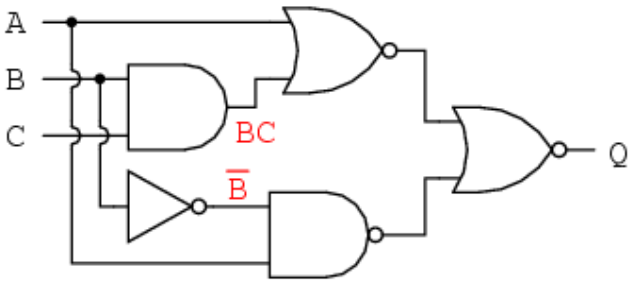


As you can see, maintaining the grouping implied by the complementation bars for this expression is crucial to obtaining the correct answer.

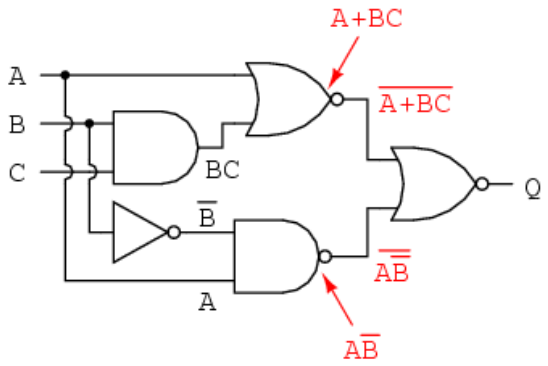
Let's apply the principles of DeMorgan's theorems to the simplification of a gate circuit:



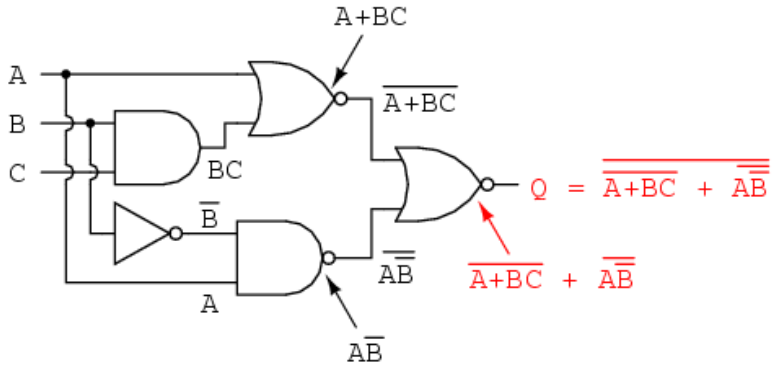
As always, our first step in simplifying this circuit must be to generate an equivalent Boolean expression. We can do this by placing a sub-expression label at the output of each gate, as the inputs become known. Here's the first step in this process:



Next, we can label the outputs of the first NOR gate and the NAND gate. When dealing with inverted-output gates, I find it easier to write an expression for the gate's output *without* the final inversion, with an arrow pointing to just before the inversion bubble. Then, at the wire leading out of the gate (after the bubble), I write the full, complemented expression. This helps ensure I don't forget a complementing bar in the sub-expression, by forcing myself to split the expression-writing task into two steps:



Finally, we write an expression (or pair of expressions) for the last NOR gate:



Now, we reduce this expression using the identities, properties, rules, and theorems (DeMorgan's) of Boolean algebra:

$\overline{\overline{A + BC} + \overline{A\overline{B}}}$

↓ Breaking longest bar

$\overline{(\overline{A + BC})}\ \overline{(\overline{A\overline{B}})}$

↓ Applying identity $\overline{\overline{A}} = A$ wherever double bars of equal length are found

$(A + BC)(A\overline{B})$

↓ Distributive property

$AA\overline{B} + BCA\overline{B}$

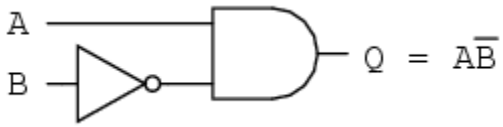
↓ Applying identity $AA = A$ to left term; applying identity $A\overline{A} = 0$ to B and $\overline{B}$ in right term

$A\overline{B} + 0$

↓ Applying identity $A + 0 = A$

$A\overline{B}$

The equivalent gate circuit for this much-simplified expression is as follows:



References

All About Circuits. (2019). Retrieved from DeMorgan's Theorems: https://www.allaboutcircuits.com/textbook/digital/chpt-7/demorgans-theorems/

Electronics Tutotials. (2019). Retrieved from Combinational Logic Circuits: https://www.electronics-tutorials.ws/combination/comb_1.html