Boolean Algebra

- Basic mathematics for the study of logic design is **Boolean Algebra**
- Basic laws of Boolean Algebra will be implemented as switching devices called **logic gates.**
- Networks of Logic gates allow us to manipulate digital signals
  - Can perform numerical operations on digital signals such as addition, multiplication
  - Can perform translations from one binary code to another.

Boolean Variables, Functions

- A **boolean variable** can take on two values
  - Will use the values '0' and '1'
  - Could just as easily use 'T', 'F or H,L or ON,OFF
- **Boolean operations** transform Boolean Variables.
  - Basic operations are NOT, AND, OR
- We can make more complicated **Boolean Functions** from the basic boolean operations

NOT operation

The NOT operation (or inverse, or complement operation) replaces a boolean value with its complement:

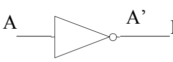
0' = 1 , 1' = 0

Truth Table

A	Y
0	1
1	0

A' is read as *NOT A* or *Complement A*

A A'



Inverter symbol

F(A) = A' boolean representation

AND operation

The AND operation is a function of two variables (A, B)

F(A,B) = A • B boolean function representation

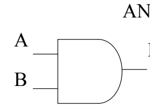
When both A **and** B are '1', then F is '1'.

0 • 0 = 0, 0 • 1 = 0, 1 • 0 = 0, 1 • 1 = 1

Truth Table

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

AND

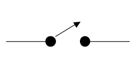


AND operation (cont).

Will usually drop the '• ' in the equation and just write the equation as:

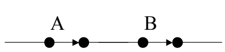
F(A,B) = AB boolean function representation

Can also view AND operation as two switches in series:



Switch Open (0)

Switch Closed (1)



When Switch A closed (1) and Switch B closed (1), circuit is closed (1)

OR operation

The OR operation is a function of two variables (A, B)

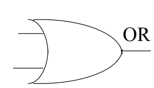
F(A,B) = A + B boolean function representation

When either A **or** B are '1', then F is '1'.

0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, 1 + 1 = 1

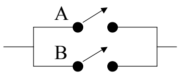
Truth Table

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

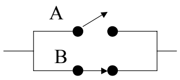


OR operation (cont).

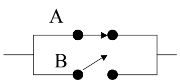
Can view OR operation as two switches in parallel:



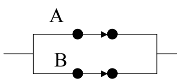
Neither switch A or switch B is closed, so circuit is open (0)



Switch B closed (1), so circuit is closed (1)



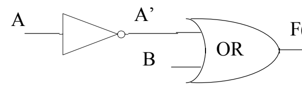
Switch A closed (1), so circuit is closed (1)



Switch A or Switch B is closed, circuit is closed (1)

Boolean Functions

More complex boolean functions can be created by combining basic operations



F(A,B) = A' + B

A	B	A'	F = A' + B
0	0	1	1
0	1	1	1
1	0	0	0
1	1	0	1

Basic Theorems

	Duals
X + 0 = X	X * 1 = X
X + 1 = 1	X * 0 = 0
X + X = X	X * X = X
(X')' = X	(X')' = X
X + X' = 1	X * X' = 0

Duality

A *dual* of a boolean expression is formed by replacing ANDs with ORs, ORs with ANDs, '1's with '0's, and '0's with '1's. Variables and their complements are left alone.

If two boolean expressions are equal, then their *duals* are equal!

Helpful in remembering boolean laws. Only need to remember one set, can generate the 2nd set by taking the dual!

Proving a Theorem

How do we prove X + 0 = X is correct?

One way is to replace all boolean variables with values of '0', '1' and use basic operations:

For X = 0, 0 + 0 = 0
0 = 0

For X = 1, 1 + 0 = 1
1 = 1

So, X + 0 = X is valid.

Prove X + X' = 1

For X = 0, 0 + (0)' = 1
0 + 1 = 1
1 = 1

For X = 1, 1 + (1)' = 1
1 + 0 = 1
1 = 1

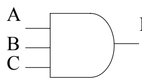
So, X + X' = 1 is valid.

Commutative, Associative Laws

	Dual Laws
Commutative	
X + Y = Y + X	X * Y = Y * X
Associative	Associative
(X + Y) + Z = X + (Y + Z)	(XY) Z = X(YZ)

If '+' is viewed as addition, and '*' as multiplication, then the Commutative, Associative laws in normal algebra are the same as in boolean algebra.

Three Input AND Function



F = ABC = (AB) C = A (BC)

A B C	AB	F = ABC
0 0 0	0	0
0 0 1	0	0
0 1 0	0	0
0 1 1	0	0
1 0 0	0	0
1 0 1	0	0
1 1 0	1	0
1 1 1	1	1

Distributive Law

A(B + C) = AB + AC (valid in normal algebra)

Dual:

A + BC = (A + B) (A + C) (only valid in Boolean algebra!)

Note that the 2nd form is NOT valid in normal algebra! This tends to make one forget about it. Remember the first form, then take the *DUAL* of it to get the second form.

Prove A + BC = (A + B) (A + C)

Use Truth Table method for both sides

A B C	BC	A+BC	A+B	A+C	(A+B)(A+C)
0 0 0	0	0	0	0	0
0 0 1	0	0	0	1	0
0 1 0	0	0	1	0	0
0 1 1	1	1	1	1	1
1 0 0	0	1	1	1	1
1 0 1	0	1	1	1	1
1 1 0	0	1	1	1	1
1 1 1	1	1	1	1	1

Results are same

Other Simplification Theorems

	Duals
XY + XY' = X	(X +Y)(X+Y') = X
X + XY = X	X(X + Y) = X
(X + Y')Y = XY	XY' + Y = X + Y

Prove XY + XY' = X via algebraic manipulation

XY + XY' = X (Y + Y') = X (1) = X !!!!!

Simplification

- Simplification tries to reduce the number of terms in a boolean equation via use of basic theorems
- A simpler equation will mean:
  - less gates will be needed to implement the equation
  - could possibly mean a faster gate-level implementation
- Will use algebraic techniques at first for simplification
  - Later, will use a graphical method called K-maps
  - Computer methods for simplification are widely used in industry.

Sum Of Products (SOP) Form

A boolean expression is in **Sum Of Products** form when all products are the products of single variables only.

F = AB' + CD'E + AC'E' (SOP Form)

G = ABC' + DEFG + H (SOP Form)

Y = A + B' + C + D (SOP Form)

Z = (A+B)CD + EF Not SOP Form

= ACD + BCD + EF SOP Form

Use Distributive Law for Multiplying

Problem: Put into SOP form the following equation and simplify:

(A + BC) (A + D + E)

Try just straightforward multiplication of terms:

AA + AD + AE + ABC + BCD + BCE

Simplify (AA = A):

A + AD + AE + ABC + BCD + BCE

Look for simplification via factoring:

A(1 + D + E + BC) + BCD + BCE

A (1) + BCD + BCE

A+ BCD + BCE !!!!!!!!! (Final SOP form)

Use 2nd Distributive Law

Recall 2nd Distributive Law:

(X + Y)(X + Z) = X + YZ

Lets try and use this law, may make things easier:

(A + BC) (A + D + E)

(A + (BC)) (A + (D + E))

Apply 2nd Distributive Law:

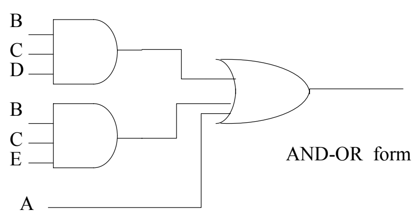
A + BC (D + E)

Multiply Out:

A + BCD + BCE (Final SOP form)

Finished!!

A + BCD + BCE as logic gates



SOP can be implemented in two levels of logic assuming that both a variable (A) and its complement (A') are available (Dual Rail Inputs). SOP is a **TWO-LEVEL** form (AND-OR)

Product of Sums (POS) Form

A boolean expression is in **Product Of Sums** form when all sums are the sums of single variables.

F = (A + B')(C + D' + E)(A + C' + E') (POS Form)

G = A (B + E)(C + D) (POS Form)

Y = AB + AC Not POS Form

= A (B + C) POS Form

Factoring

Use factoring to get to Product of Sums form.

Use basic theorem:

X + YZ = (X + Y) (X + Z) (just reverse of distributive law)

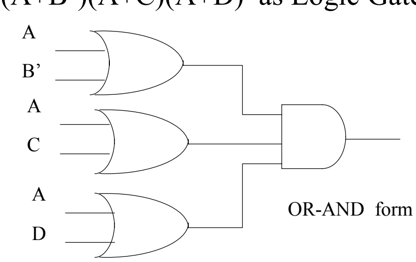
Problem: Put A + B'CD into POS Form:

A + B'CD = (A + B') (A + CD)

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

(A + B') (A + C) (A + D) is final POS form!!!!

(A+B')(A+C)(A+D) as Logic Gates



POS can be implemented in two levels of logic assuming that both a variable (A) and its complement (A') are available (Dual Rail Inputs). POS is a **TWO-LEVEL** form (OR-AND form)

Consensus Theorem

Consensus theorem states:

$$XY + X'Z + YZ = XY + X'Z$$

The **YZ** term is called the *consensus term* and is redundant. The consensus term is formed from a PAIR OF TERMS in which a variable (X) and its complement (X') are present; the consensus term is formed by multiplying the two terms and leaving out the selected variable and its complement.

The consensus of XY, X'Z is YZ .

Prove the Consensus Theorem

Consensus Theorem Proof:

$$\begin{aligned} XY + X'Z + YZ &= XY + X'Z + (X + X')YZ \\ &= XY + X'Z + XYZ + X'YZ \\ &= (XY + XYZ) + (X'Z + X'YZ) \\ &= XY(1 + Z) + X'Z(1 + Y) \\ &= XY + X'Z \end{aligned}$$

You could also use a truth table to prove this.

Dual of the Consensus Theorem

$$(X+Y)(X' + Z)(Y + Z) = (X + Y)(X' + Z)$$

The consensus of (X+Y)(X'+Z) is (Y + Z) .

How do you use the consensus theorem? Simply be suspicious anytime you have two terms that have a variable and its complement. Form the consensus term and see if it is present; if consensus term is present, just get rid of it.

Short Cuts for Multiplying

A short cut theorem for Distribution (Multiplication)

$$(X + Y)(X' + Z) = XZ + X'Y$$

Only works when you have a variable (X) and its complement (X'). To PROVE this, lets do the distribution the long way.

$$(X + Y)(X' + Z) = \underbrace{XX'}_{0} + XZ + X'Y + YZ$$

Redundant by consensus theorem

$$= 0 + XZ + X'Y = XZ + X'Y$$

DeMorgan's Laws

DeMorgan's Laws provide an easy way to find the inverse of a boolean expression:

$$(X + Y)' = X'Y'$$
$$(XY)' = X' + Y'$$

An easy way to remember this is that each TERM is complemented, and that ORs become ANDs; ANDs become ORs.

Easy to prove this via a truth table, see textbook.

Applying DeMorgan's Law

Apply DeMorgan's Law to a more complex expression:

$$\begin{aligned} (AB + C'D)' &= (AB)'(C'D)' \\ &= (A'+B')(C + D') \end{aligned}$$

Note that DeMorgan's law was applied twice.

Another example:

$$\begin{aligned} [(A' + B)C']' &= (A' + B)' + (C')' \\ &= (A')'(B)' + C \\ &= AB' + C \end{aligned}$$

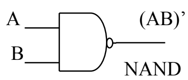
NAND, NOR Gates

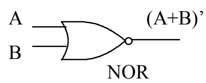
Why do we care about DeMorgan's Law?

There are two other gate types that produce the complement of a boolean function!

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

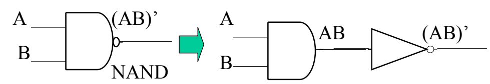
A B	Y
0 0	1
0 1	0
1 0	0
1 1	0



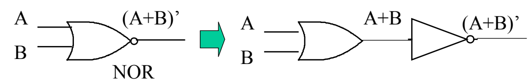


NAND, NOR (cont.)

NAND (NOT AND) - can be thought of as an AND gate followed by an inverter.



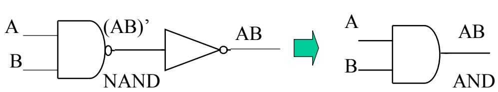
NOR (NOT OR) - can be thought as an OR gate followed by an inverter.

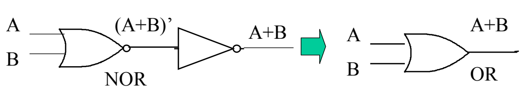


Actually….

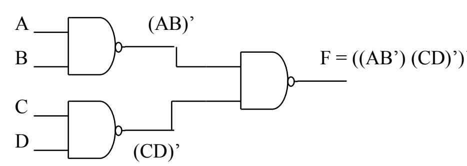
In the real world, an AND gate is made from an NAND gate followed by an inverter!!!

An OR gate is made from a NOR gate followed by an inverter!!!





What is this logic function in SOP form?

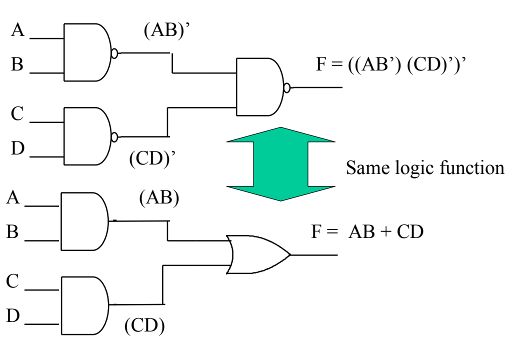


Hmmmmmmmm…. Lets use DeMorgan's Law

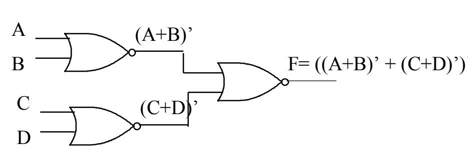
F = ((AB)'(CD)')' = ((AB)')' + ((CD)')' = AB + CD !!!!

An interesting result…... SOP Form!!

NAND-NAND form = AND-OR form



What is this logic function in POS form?

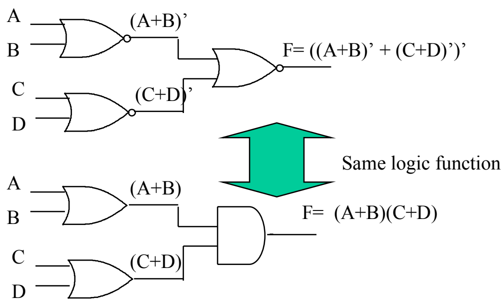


Hmmmmmmmm…. Lets use DeMorgan's Law

F = ((A+B)' + (C+D)')'
= ((A+B)')' ((C+D)')' = (A+B) (C+D) !!!!!

An interesting result…... POS Form!!!

NOR-NOR form = OR-AND form



Two Level Form Summary

Any logic function in SOP form (Sum of Products) can be implemented in the two level gate forms of AND-OR, NAND-NAND.

Any logic function in POS form (Product of Sums) can be implemented in the two level gate forms of OR-AND, NOR-NOR.

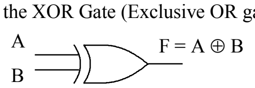
There are actually FOUR more two level gate forms but we will not talk about these.

XOR Function

One last gate type is the XOR Gate (Exclusive OR gate).

XOR

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



XOR gate is usual in logic circuits that do binary addition/subtraction.

Note that:

$$F = A \oplus B = A'B + AB'$$

What do you need to know?

- Basic Boolean Theorems
- Proving boolean theorems (algebraically, truth table)
- Duality
- Boolean equation to gate network and vice-versa
- Algebraic Simplification
- Consensus Theorem, De'Morgans Laws
- SOP form, POS form
- Two level forms AND-OR, NAND-NAND, OR-AND, NOR-NOR
- XOR Gate