

COMP 202. Introduction to Electronics

Dr. N. B. Gyan

Central University, Miotso. Ghana

Brief Background

- The digital circuitry in digital computers and other digital systems is designed, and its behaviour is analysed, with the use of a mathematical discipline known as Boolean algebra.
- The name is in honour of an English mathematician George Boole, who proposed the basic principles of this algebra in 1854 in his treatise, *An Investigation of the Laws of Thought on Which to Found the Mathematical Theories of Logic and Probabilities*.

Brief Background

- In 1938, Claude Shannon, a research assistant in the Electrical Engineering Department at M.I.T., suggested that Boolean algebra could be used to solve problems in relay-switching circuit design.
- Boolean algebra turns out to be a convenient tool in two areas:
 - Analysis: It is an economical way of describing the function of digital circuitry.
 - Design: Given a desired function, Boolean algebra can be applied to develop a simplified implementation of that function.

- The basic logical operations are AND, OR, and NOT, which are symbolically represented by dot, plus sign, and overbar (also complement/prime symbol)
 - $A \text{ AND } B = A \cdot B$
 - $A \text{ OR } B = A + B$
 - $\text{NOT } A = \bar{A}$
 - $A \text{ NAND } B = \text{NOT } (A \text{ AND } B) = \overline{AB}$
 - $A \text{ NOR } B = \text{NOT } (A \text{ OR } B) = \overline{A + B}$
- The operation AND yields true (binary value 1) if and only if both of its operands are true. The operation OR yields true if either or both of its operands are true. The unary operation NOT inverts the value of its operand.
- The NAND function is the complement (NOT) of the AND function, and the NOR is the complement of OR.



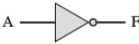
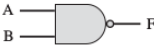


- For example, consider the equation:
- $D = A + (\bar{B} \cdot C)$
- This translates as D is equal to 1 if A is 1 or if both $B = 0$ and $C = 1$. Otherwise D is equal to 0.

Identities

Basic Postulates		
$A \cdot B = B \cdot A$	$A + B = B + A$	Commutative Laws
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C) = (A + B) \cdot (A + C)$	Distributive Laws
$1 \cdot A = A$	$0 + A = A$	Identity Elements
$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$	Inverse Elements
Other Identities		
$0 \cdot A = 0$	$1 + A = 1$	Associative Laws DeMorgan's Theorem
$A \cdot A = A$	$A + A = A$	
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$	
$\overline{A \cdot B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$	

Logic Gates

- The fundamental building block of all digital logic circuits is the gate. Logical functions are implemented by the interconnection of gates.
- A gate is an electronic circuit that produces an output signal that is a simple Boolean operation on its input signals. The basic gates used in digital logic are AND, OR, NOT, NAND, NOR, and XOR.
- When one or more of the values at the input are changed, the correct output signal appears almost instantaneously, delayed only by the propagation time of signals through the gate (known as the gate delay).

Name	Graphical Symbol	Algebraic Function	Truth Table															
AND		$F = A \cdot B$ or $F = AB$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	0	1	0	0	1	1	1
A	B	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = A + B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	1
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		$F = \bar{A}$ or $F = A'$	<table><tr><th>A</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	F	0	1	1	0									
A	F																	
0	1																	
1	0																	
NAND		$F = \overline{AB}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = \overline{A + B}$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	0
A	B	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
XOR		$F = A \oplus B$	<table><tr><th>A</th><th>B</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																

- Typically, not all gate types are used in implementation. Design and fabrication are simpler if only one or two types of gates are used.
- Thus, it is important to identify *functionally complete sets* of gates. This means that any Boolean function can be implemented using only the gates in the set.
- The following are functionally complete sets:
 - AND, OR, NOT
 - AND, NOT
 - OR, NOT
 - NAND
 - NOR

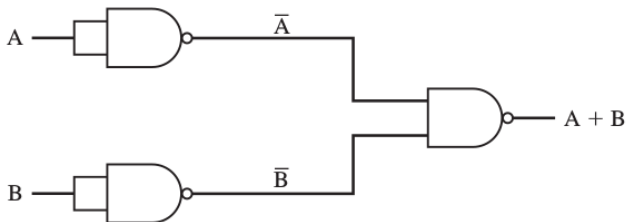
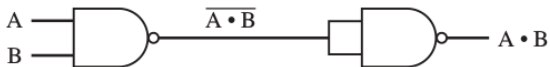
- It should be clear that AND, OR, and NOT gates constitute a functionally complete set, because they represent the three operations of Boolean algebra.
- For the AND and NOT gates to form a functionally complete set, there must be a way to synthesize the OR operation from the AND and NOT operations. This can be done by applying DeMorgan's theorem:

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

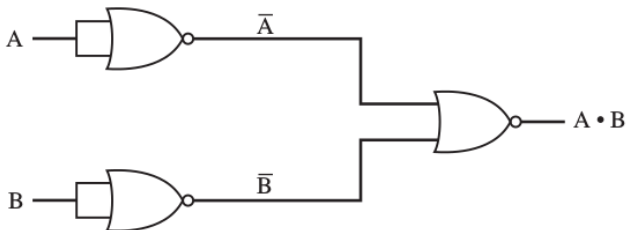
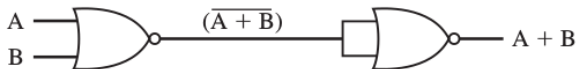
$$A \text{ OR } B = \text{NOT} ((\text{NOT } A) \text{ AND } (\text{NOT } B))$$

- Similarly, the OR and NOT operations are functionally complete because they can be used to synthesize the AND operation.

NAND



NOR



Exercises

Simulate the above diagrams in digital

Find software here:

<https://github.com/hneemann/Digital/releases/download/v0.29/Digital.zip>