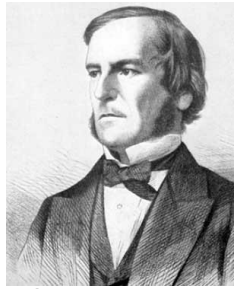
POLITECNICO
DI TORINO

Dipartimento
di Automatica e Informatica

Boolean Algebra and Logic Functions

Paolo Camurati

Boolean Algebra



- It was introduced in 1854 by George Boole to algebraically analyze propositional calculus problems (in order to study the laws of thought)
- It was then used as a foundation for the formal logic that serves as the basis for scientific reasoning
- It is a mathematical model used to express logical conditions and for the synthesis and analysis of digital systems
- Boolean Algebra is an abstract model: there are many Boolean Algebras!

Boolean Algebras

A Boolean algebra is an algebraic structure consisting of:

- a set of elements B that includes at least 0 and 1
- two binary operations (= operating on 2 operands) {+, ·}
- a unary operation (= operating on 1 operand) {'}

that satisfy Huntingdon's axioms (1904):

1. B contains at least two different elements a and b with $a \neq b$
2. *Closedness:* $\forall$ a,b $\in$ B
   - (i) $a + b \in B$
   - (ii) $a \cdot b \in B$
   - *(iii)* $a' \in B$

3. *Commutativity*: $\forall$ a,b $\in$ B
   - (i) $a + b = b + a$
   - (ii) $a \cdot b = b \cdot a$
4. *Identity*: there exist 2 values 0, 1 $\in$ *B* such that
   - (i) $a + 0 = a$
   - (ii) $a \cdot 1 = a$
5. *Distributivity*:
   - (i) $a + (b \cdot c) = (a + b) \cdot (a + c)$
   - (ii) $a \cdot (b + c) = a \cdot b + a \cdot c$
6. *Complement*:
   - (i) $a + a' = 1$
   - (ii) $a \cdot a' = 0$

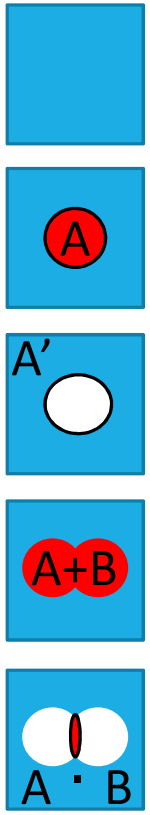
Examples of Boolean Algebras

By specifying the elements of B and the operations, an interpretation is defined and a model of the algebra is obtained.

Set algebra:

Let a universe set S be given. The powerset of S is the set of subsets of S, including S itself and the empty set ∅.

- B is the powerset of S
- ‘ is the complement operation with respect to the universe S as a whole
- + is the set union operation
- · is the set intersection operation



Switching Algebra

Claude Shannon (1938) introduced switching algebra to describe the properties of electrical circuits with switches.

- B = {0, 1}
- binary operations: OR (+), AND (·)
- unary operation: NOT ('or ¯ or ~ or overline‾).

Switching algebra (hereinafter called with abuse of notation Boolean Algebra):

- is used to express the logical conditions in programming languages
- is the basis for designing and analyzing digital systems.

Switches and Switching Algebra

Inputs:

- 1 corresponds to a closed switch
- 0 corresponds to an open switch

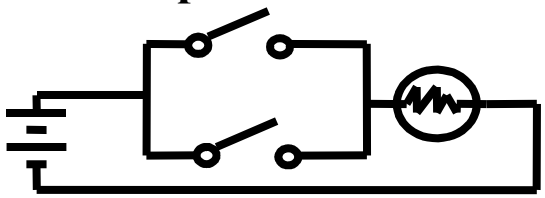
Output:

- 1 corresponds to lamp on
- 0 corresponds to lamp off

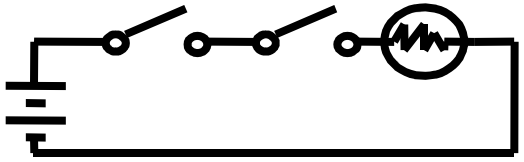
NOT uses a switch such that:

- 1 is the open switch
- 0 is the closed switch

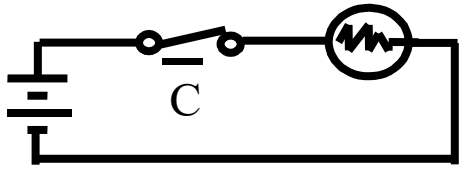
Switches in parallel => OR



Switches in series => AND



Normally closed switch => NOT



Boolean Variables

- The two (binary) values in B have several names:
  - True/False
  - On/Off
  - Yes/No
  - 1/0
- We use 1 and 0
- Boolean variable: variable that takes values in B
- Examples of variable identifiers: A, B, x, Y, y, z

Examples of notation

Let A, B, x, Y, y, z be binary variables:

- $Y = A \cdot B$ reads “Y equals A AND B.”
- $z = x + y$ reads “z equals x OR y.”
- $X = A'$ reads “X equals NOT A.”

Remark: the statement:

- 1 + 1 = 2 (read “1 plus 1 equals 2”) holds when + is an arithmetic sum operator
- 1 + 1 = 1 (read “1 or 1 equals 1”) holds when + is the OR operator in Boolean Algebra

Basic Theorems

Idempotency:	$a + a = a$	$a \cdot a = a$
Null Element:	$a + 1 = 1$	$a \cdot 0 = 0$
Absorption:	$a + (a \cdot b) = a$	$a \cdot (a + b) = a$
	$a \cdot (a' + b) = a \cdot b$	$a + (a' \cdot b) = a + b$
Involution:	$(a')' = a$	
Associativity:	$a + (b + c) = (a + b) + c$	$a \cdot (b \cdot c) = (a \cdot b) \cdot c$
De Morgan's Law:	$(a + b)' = a' \cdot b'$	$(a \cdot b)' = a' + b'$
Combination:	$(a \cdot b) + (a \cdot b') = a$	$(a + b) \cdot (a + b') = a$
Consensus:	$(a \cdot b) + (a' \cdot c) + (b \cdot c) = (a \cdot b) + (a' \cdot c)$	
	$(a + b) \cdot (a' + c) \cdot (b + c) = (a + b) \cdot (a' + c)$	
Factorization:	$(a + b) \cdot (a' + c) = (a \cdot c) + (a' \cdot b)$	
	$(a \cdot b) + (a' \cdot c) = (a + c) \cdot (a' + b)$	

Properties

- If there is no ambiguity, symbol · can be omitted
- Symbol ’ is used for NOT
- The dual of an algebraic expression is obtained by exchanging + and · and exchanging 0 and 1.
- Theorems appear in dual pairs. When there is only one theorem in a line, it is called self-dual, i.e. dual expression = original expression
- The fundamental theorems are demonstrated starting from Huntingdon's postulates.

Demonstrations

Absorption theorem: $a + (a' \cdot b) = a + b$

- Perfect Induction (all cases)

a	a'	b	a + a'b	a + b
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1

- Using postulates and already demonstrated theorems

$a + a'b = (a + a') \cdot (a + b)$	distribuitivity
$(a + a') \cdot (a + b) = 1 \cdot (a + b)$	complement
$1 \cdot (a + b) = a + b$	identity

Consensus theorem: ab + a'c + bc = ab + a'c

- Using postulates and already demonstrated theorems

ab + a'c + bc	= ab + a'c + 1·(bc)	identity
ab + a'c + 1·(bc)	= ab + a'c + (a + a')·(bc)	complement
ab + a'c + (a + a')·(bc)	= ab + a'c + abc + a'bc	distributivity
ab + a'c + abc + a'bc	= ab + abc + a'c + a'bc	commutativity
ab + abc + a'c + a'bc	= ab + a'c	absorption

Definition of Boolean Operators

AND	OR	NOT
$0 \cdot 0 = 0$	$0 + 0 = 0$	$0' = 1$
$0 \cdot 1 = 0$	$0 + 1 = 1$	$1' = 0$
$1 \cdot 0 = 0$	$1 + 0 = 1$	
$1 \cdot 1 = 1$	$1 + 1 = 1$	

Boolean Expressions

- Boolean expressions are all and only those formed through the use of:
- constants 0, 1
- Boolean variables x1, x2, ..., xn
- AND, OR, NOT operators

- applying the following rules:
- a constant is an expression
- a variable is an expression
- if g and h are Boolean expressions, then so are g + h, g · h and g '

Boolean Operators Precedence

Boolean operators precedence is:

1. brackets
2. NOT
3. AND
4. OR

Consequence: brackets enclose OR expressions

Example: A(B + C)(C + D')

Boolean Functions

Given B={0,1}, f is a logic or Boolean function of n Boolean variables x1, x2, ..., xn :

$$f(x1, x2, ..., xn): B^n \rightarrow B$$

that matches all value assignments to Boolean variables x1, x2, ..., xn to either *0* or *1*.

B^n is the Cartesian product $\underbrace{B \times B \times \ldots B}_{n \text{ times}}$.

How many functions of n Boolean variables exist?

- n variables can take $k = 2^n$ possible values, i.e. the number of arrangements with repetitions of 2 elements taken from n (n choose 2)
- for each assignment the function takes either value 0 or 1 (m = 2)
- the number of functions on n Boolean variables equals the number of repeated arrangements of m elements taken from k (m choose k)

$$D'_{m,k} = m^k = 2^{2^n}$$

When n = 1 there are 4 functions:

- f(x) = 0 constant output 0
- f(x) = 1 constant output 1
- f(x) = x input transferred onto output
- f(x) = x' complemented input transferred onto output

When n= 2 there are 16 possible functions:

- f(x,y) = 0 constant 0
- f(x,y) = 1 constant 1
- f(x,y) = x transfer
- f(x,y) = y transfer
- f(x,y) = x' transfer and NOT
- f(x,y) = y' transfer and NOT
- f(x,y) = xy AND
- f(x,y) = (xy)' **NAND**
- f(x,y) = x+y OR
- f(x,y) = (x+y)' **NOR**
- f(x,y) = x'+y implication $x \rightarrow y$
- f(x,y) = x+y' implication $y \rightarrow x$
- f(x,y) = xy' inhibition of y
- f(x,y) = x'y inhibition di x
- f(x,y) = $x \oplus y$ **EXOR**
- f(x,y) = $(x \oplus y)'$ **EXNOR**

NAND, NOR, EXOR, EXNOR: non-elementary Boolean functions.

x	y	0	1	x	y	x'	y'	xy	$x \uparrow y$
0	0	0	1	0	0	1	1	0	1
0	1	0	1	0	1	1	0	0	1
1	0	0	1	1	0	0	1	0	1
1	1	0	1	1	1	0	0	1	0

x	y	x+y	$x \downarrow y$	$x \rightarrow y$	$y \rightarrow x$	xy'	x'y	$x \oplus y$	$(x \oplus y)'$
0	0	0	1	1	1	0	0	0	1
0	1	1	0	1	0	0	1	1	0
1	0	1	0	0	1	1	0	1	0
1	1	1	0	1	1	0	0	0	1

Representation of Boolean Functions

Formalisms:

- Expressions
- Truth tables
- Circuit diagrams.

A representation formalism is said to be canonical if, given two arbitrary functions f and g these are equal if and only if their representations are the same.
Truth tables are canonical representations, expressions and circuit diagrams are not.

Truth Tables

The truth table of a Boolean function f is a tabular list of the values of f for all possible combinations of its arguments. For a function f of n variables, the table has 2^n rows and n +1 columns.

AND		
x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR		
x	y	x+y
0	0	0
0	1	1
1	0	1
1	1	1

NOT	
x	x’
0	1
1	0

NAND and NOR

NAND and NOR are used for building digital circuits using logic gates. From a a technological point of view it is easier to implement NAND and NOR gates than AND, OR and NOT gates.
Associativity is not valid for NAND and NOR.

NAND		
x	y	(xy)’
0	0	1
0	1	1
1	0	1
1	1	0

NOR		
x	y	(x+y)’
0	0	1
0	1	0
1	0	0
1	1	0

Symbols found in literature:
NAND ↑, NOR ↓

EXOR and EXNOR

EXOR is the difference operator $x \neq y = xy' + x'y$

EXNOR is the equality operator $x \equiv y = xy + x'y'$

EXOR		
x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

EXNOR		
x	y	$(x \oplus y)'$
0	0	1
0	1	0
1	0	0
1	1	1

Symbols found in literature:
EXOR $\oplus$, EXNOR $\equiv$.

Implication

It is used to express conditionals like

If the sun shines, we go out for a walk $x \rightarrow y$ x implies y

Case by case analysis:

- x=1 and y=1: $x \rightarrow y$ is true (it equals 1) INTUITIVE!
- x=1 and y=0: $x \rightarrow y$ is false (it equals 0) INTUITIVE!
- x=0 and y=0 or y=1: $x \rightarrow y$ is true (it equals 1) COUNTERINTUITIVE!

If the assumption is false, we can draw any conclusion!

Boolean expression:

$x \rightarrow y = xy + x'$ absorption

$= x' + y$

Implication is not an elementary logic gate.

Functionally Complete Sets

A set of operators is functionally complete if all Boolean functions can be described by an expression that uses only the operators of that set.

Functionally complete sets:

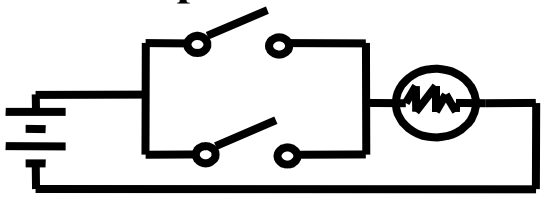
- AND, OR, NOT
- AND, NOT: in fact x + y = (x'y ')' (De Morgan)
- OR, NOT: in fact xy = (x '+ y') '(De Morgan)
- NAND: in fact x’ = (x·x)’ = x$\uparrow$x, xy = ((x·y)’)’ = (x$\uparrow$y)’= (x$\uparrow$y)$\uparrow$(x$\uparrow$y)
- NOR: in fact x’ = (x+x)’ = x $\downarrow$ x, x+y = ((x+y)’)’ = (x $\downarrow$ y)’= (x $\downarrow$ y) $\downarrow$(x $\downarrow$ y)

The functional completeness of NAND and NOR has important consequences on the implementation of digital circuits.

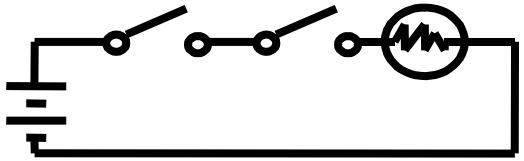
Logic Gates

Boolean algebra operators can be physically realized using electronic circuits that operate as suitably connected switches.

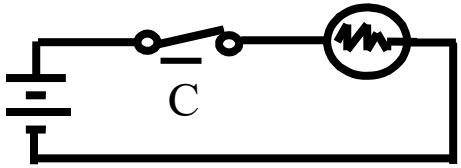
Switches in parallel => OR



Switches in series => AND



Normally closed switch => NOT



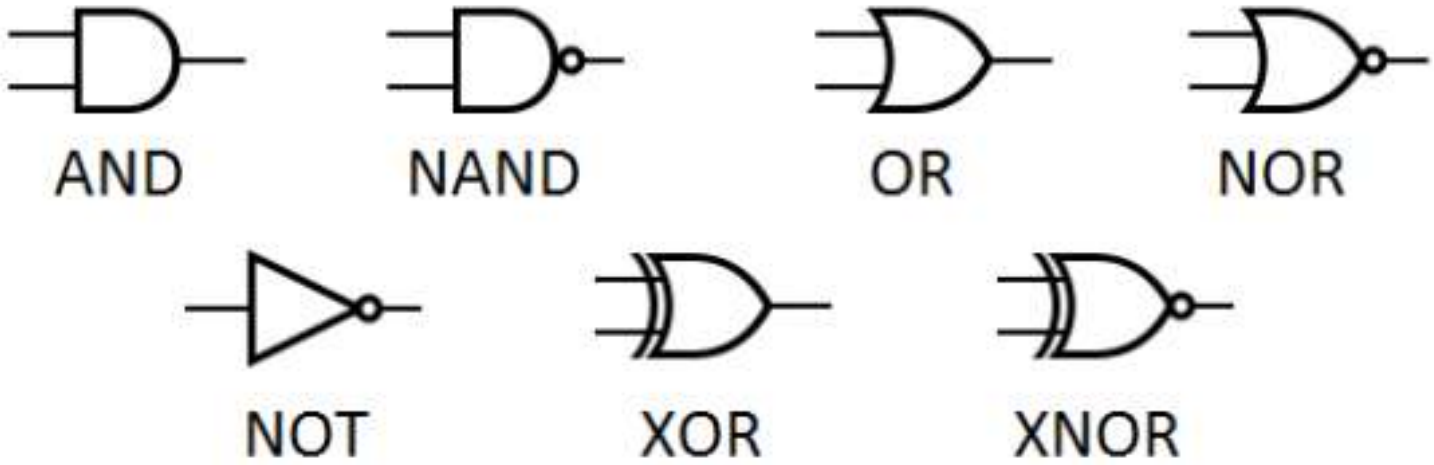
In early computers, switches were operated by magnetic fields produced in coils called relays. Switches allowed or prevented current flow.

Later vacuum tubes to allow or prevent current flow replaced relays.

Today, transistors are used as electronic switches allow or prevent current flow.

Logic gates are electronic devices that physically implement the Boolean operators AND, OR, NOT, NAND, NOR, EXOR, EXNOR.

Logic Gate Symbols



Logic Diagrams

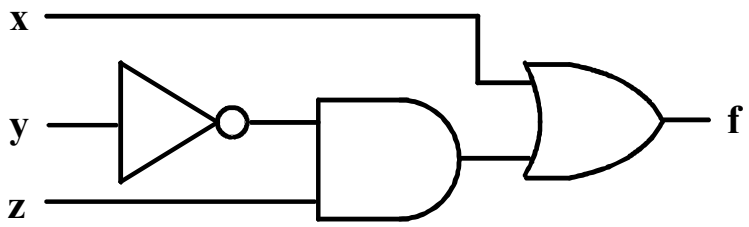
- A logic diagram is composed by logic gates connected by wires.
- Boolean expressions, truth tables and logic diagrams describe the same function!
- Truth tables are unique, not so are logical expressions and diagrams. This allows flexibility in the implementation of functions.

Example

Boolean Function

$f(x, y, z) = x + y'z$

Logic Diagram



Truth Table

f			
x	y	z	x+y'z
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Digital (Logic) Circuits

Given a set of input symbols coded on n bits ($n \geq 0$) and a set of output symbols coded on m bits ($m > 0$), a digital (logic) circuit is a hardware processing system that accepts a sequence of input symbols and transforms it into a sequence of output symbols.

The circuit is combinational if the output symbol depends solely on the input symbol at that same time. The network therefore has no memory.

The circuit is sequential if the output symbol depends on both the current and the past input symbols. The network has memory. The state stores relevant information of the input sequence that determines the evolution of the circuit over time. The number of states in sequential networks is finite.

Combinational Circuits: the Adder

If you want to add 2 n-bit numbers, what would be the size of the truth table? $\mathbf{2^{(n+n)}}$

- n = 2: 2^4 = 16 rows
- n = 4: 2^8 = 256 rows
- n = 8: 2^{16} = 65536 rows
- n = 16: 2^{32} = 4 billion rows.

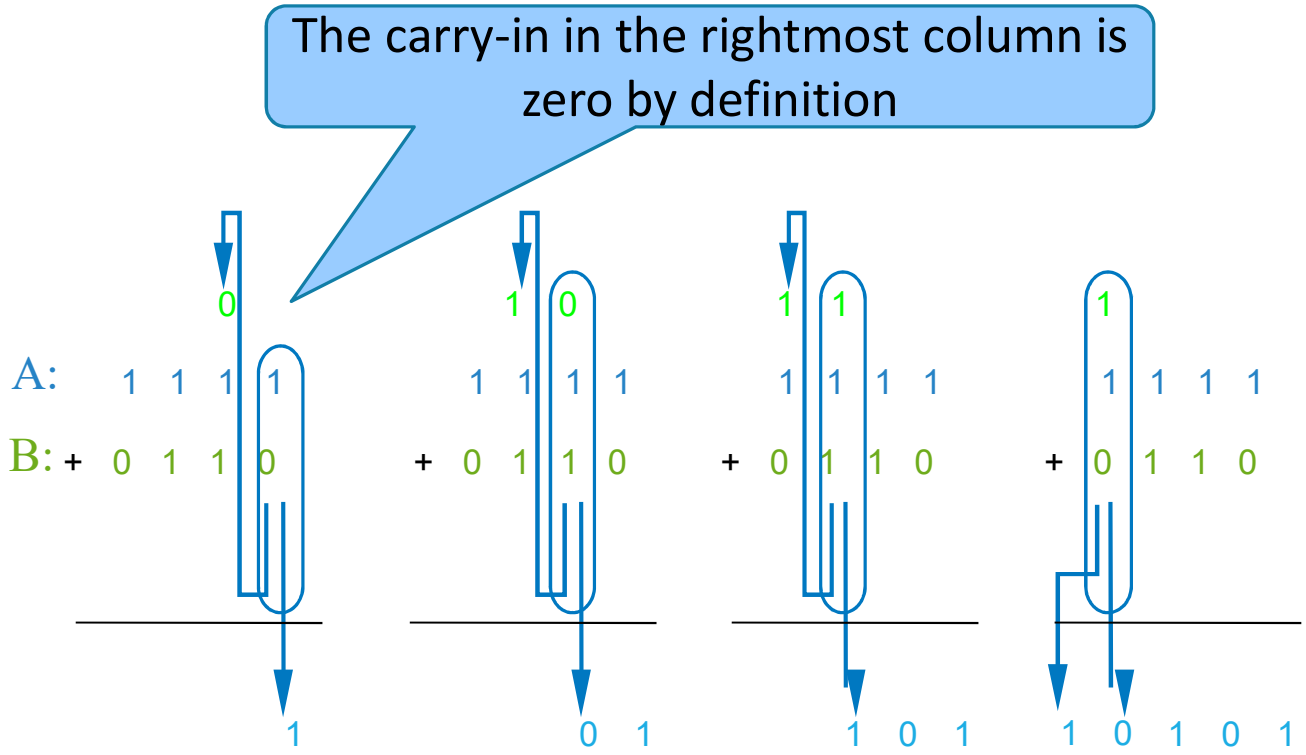
Conclusion: we need an alternative design method, which imitates the manual procedure of adding:

- put numbers in columns (bits of the same weight)
- proceed column by column from the least significant weight (rightmost column)
- compute the sum of the column and the carry that is propagated to the next column

Informatics Unit T1 Binary sum

- Basic rules:

0	+	0	=	0	
0	+	1	=	1	
1	+	0	=	1	
1	+	1	=	0	(carry = 1)


The carry-in in the rightmost column is zero by definition
A:
B:
0
1 1 1 1
+ 0 1 1 0
1
1 0
1 1 1 1
+ 0 1 1 0
0 1
1 1
1 1 1 1
+ 0 1 1 0
1 0 1
1
1 1 1 1
+ 0 1 1 0
1 0 1 0 1

To add the columns, let us design the following functional blocks using truth tables:

- Half-Adder: 1-bit X and Y inputs, 1-bit Cout (carry out) and S sum outputs (for the rightmost column)
- Full-Adder: 1-bit X, Y and Cin (carry in) inputs, 1-bit carry Cout and sum S outputs (for the other columns)

and let us connect them to implement a Ripple Carry Adder: a functional block for the sum of a pair of integers X and Y on n bits with Cin on 1 bit.

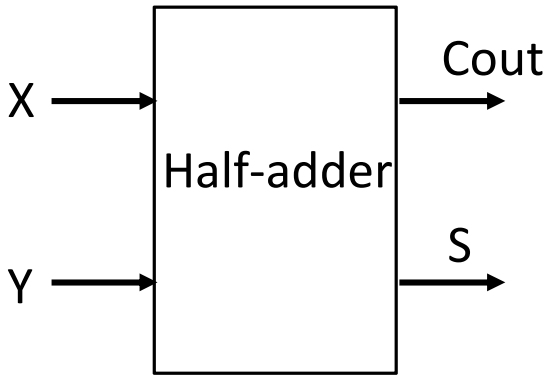
Half-adder

Operations:

X	0	0	1	1
+	+	+	+	+
Y	0	1	0	1
Cout S	0 0	0 1	0 1	1 0

Truth table:

Half-adder			
X	Y	Cout	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Boolean Function

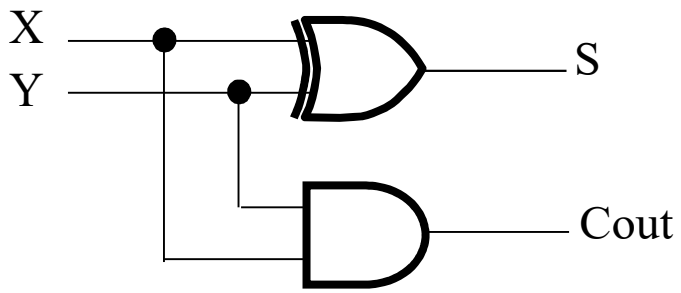
S is 1 if and only if (X = 0 AND Y = 1) OR (X = 1 AND Y = 0)

$$S = XY' + X'Y = X \oplus Y$$

Cout is 1 if and only if X = 1 AND Y = 1

$$Cout = XY$$

Logic Diagram



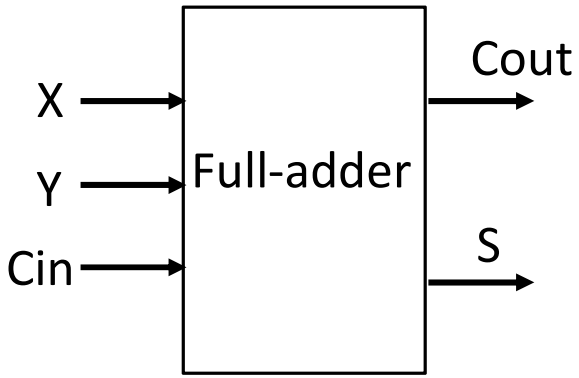
Full-adder

Operations:

Cin	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**
+	+	+	+	+	+	+	+	+
X	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**
+	+	+	+	+	+	+	+	+
Y	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
Cout S	**0 0**	**0 1**	**0 1**	**1 0**	**0 1**	**1 0**	**1 0**	**1 1**

Truth table:

Full-adder				
X	Y	Cin	Cout	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



Boolean function

S is 1 if and only if

(Cin = 1 AND X = 0 AND Y = 0) OR
(Cin = 0 AND X = 0 AND Y = 1) OR
(Cin = 0 AND X = 1 AND Y = 0) OR
(Cin = 1 AND X = 1 AND Y = 1)

$$S = CinX'Y' + Cin'X'Y + Cin'XY' + CinXY$$

Let us resort to Boolean Algebra rules to manipulate the expression:

$S = CinX'Y' + Cin'X'Y + Cin'XY' + CinXY$ distributivity

$= Cin(XY + X'Y') + Cin'(X'Y + XY')$ definition of EXOR/EXNOR

$= Cin(X \oplus Y)' + Cin'(X \oplus Y)$ definition of EXOR

$= Cin \oplus (X \oplus Y)$

Cout is 1 if and only if (Cin = 1 AND X = 0 AND Y = 1) OR
(Cin = 1 AND X = 1 AND Y = 0) OR
(Cin = 1 AND X = 1 AND Y = 1) OR
(Cin = 0 AND X = 1 AND Y = 1) OR

$$Cout = CinX'Y + CinXY' + CinXY + Cin'XY$$

Let us resort to Boolean Algebra rules to manipulate the expression:

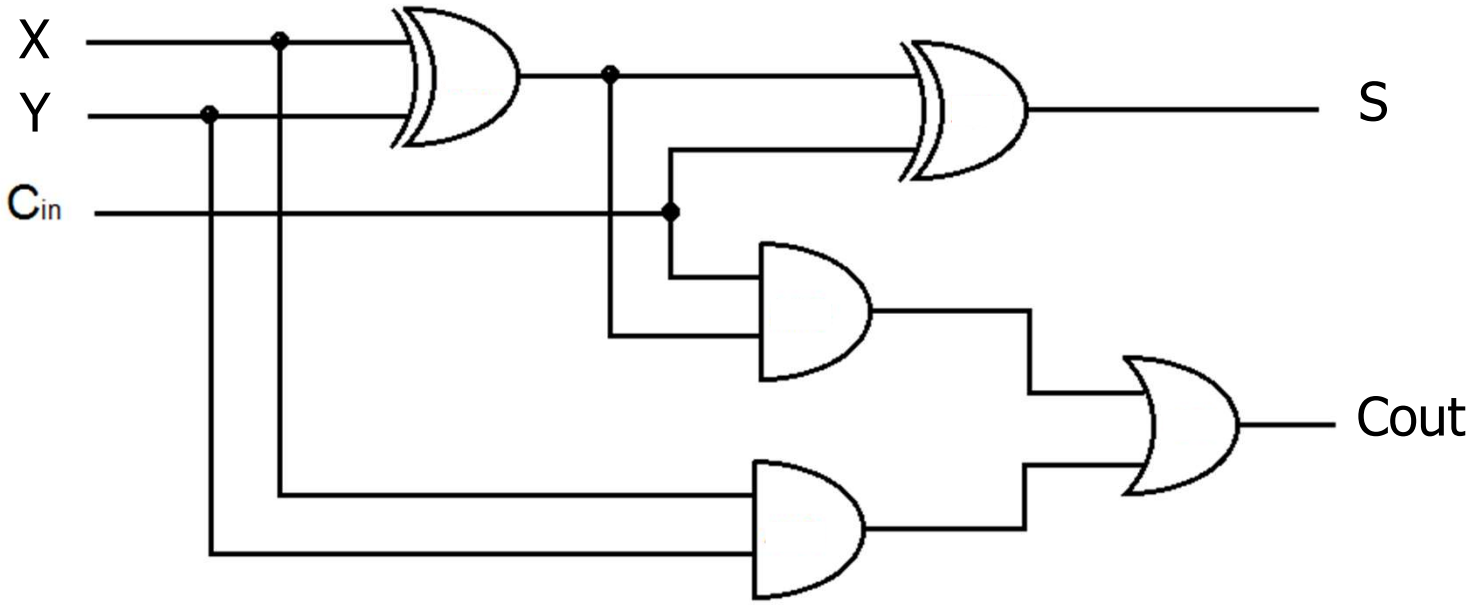
$Cout = CinX'Y + CinXY' + CinXY + Cin'XY$

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

$= Cin(X \oplus Y) + XY(1)$ EXOR/complement

$= Cin(X \oplus Y) + XY$ identity

Logic Diagram



Binary Adders

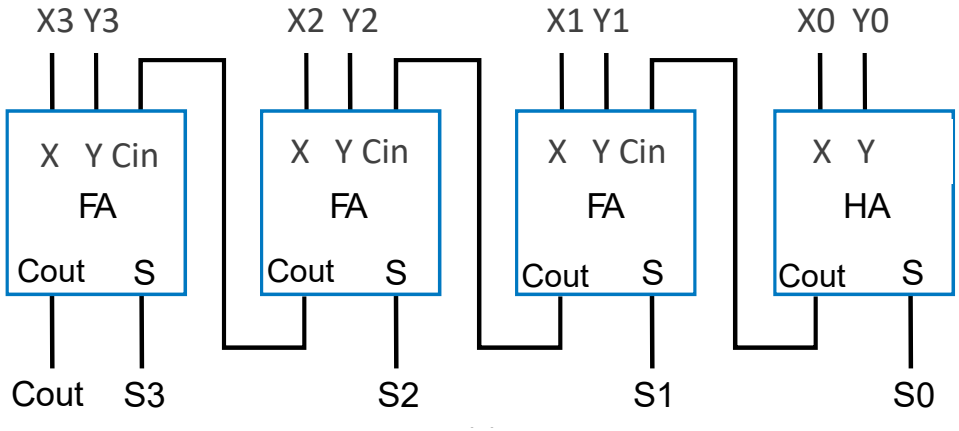
To add operands on several bits, we "group" the logic signals together in vectors and use functional blocks that work on vectors

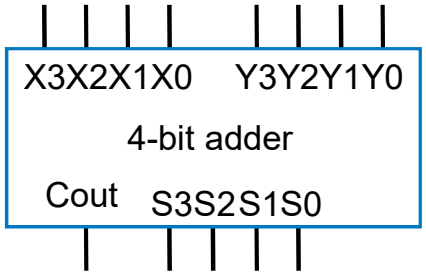
Example: 4-bit ripple carry adder:
it add vectors X(3: 0) and Y(3: 0) to get a sum vector S(3: 0)
The carry out from the i-th cell becomes the carry in of the i + 1-th cell

Description	Subscript 3 2 1 0	Name
Cin	0 1 1 0	C_i
Augend	1 0 1 1	X_i
Addend	0 0 1 1	Y_i
Sum	1 1 1 0	S_i
Carry	0 0 1 1	C_{i+1}

4-bit Ripple Carry adder

- A 4-bit Ripple Carry Adder consists of 1 Half-adder and 3 Full Adders in cascade
- Ripple carry: the carry propagates from right to left through the blocks

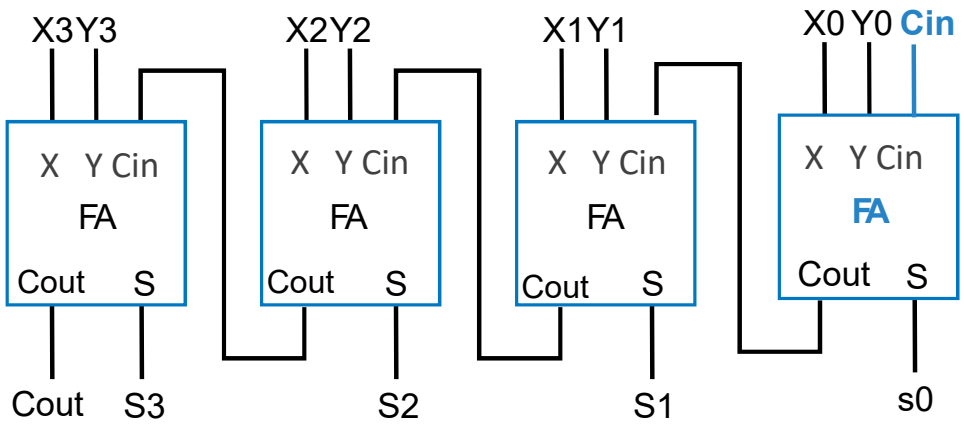


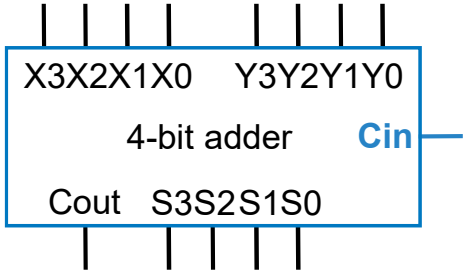


- By replacing the Half-adder with a Full-adder, we implement

  X + Y + Cin

- Useful for cascade connection of multiple adders





- By cascading k 4 bit adders, 4k-bit adders are obtained.
- Example: 8-bit adder obtained as a cascade of k = 2 4-bit adders: