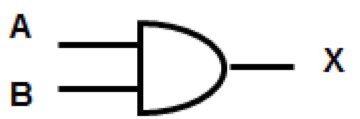
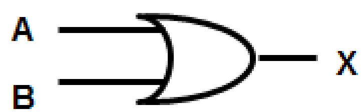
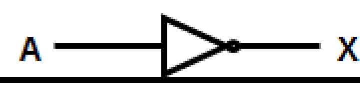
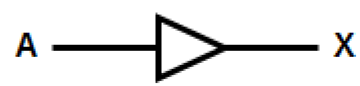
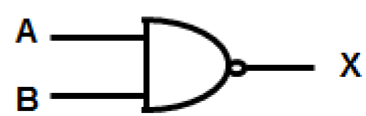
Boolean Algebra and Digital Logic

Basic Logic Block-Gates

Name	Symbol	Function	Truth Table
AND		X = A • B or X = AB	A B | X 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		X = A + B	A B | X 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
I		X = A’	A | X 0 | 1 1 | 0
Buffer		X = A	A | X 0 | 0 1 | 1
NAND		X = (AB)’	A B | X 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		X = (A + B)’	A B | X 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0

XOR Exclusive OR		$X = A \oplus B$ or $X = A'B + AB'$	A B | X 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
XNOR Exclusive NOR or Equivalence		$X = (A \oplus B)'$ or $X = A'B' + AB$	A B | X 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Boolean Algebra

- Boolean algebra is a mathematical system for the manipulation of variables that can have one of two values.
  - In formal logic, these values are "true" and "false."
  - In digital systems, these values are "on" and "off," 1 and 0, or "high" and "low."
- Boolean expressions are created by performing operations on Boolean variables.
  - Common Boolean operators include AND, OR, and NOT.

- A Boolean operator can be completely described using a truth table.
- The AND operator is also known as a Boolean product.
- The OR operator is the Boolean sum.

- A Boolean function has:
  - At least one Boolean variable,
  - At least one Boolean operator, and
  - At least one input from the set {0,1}.
- It produces an output that is also a member of the set {0,1}.

- A *Boolean algebra* is defined as a closed algebraic system containing a set K or two or more elements and the two operators, . and +.
- Useful for identifying and *minimizing* circuit functionality
- Identity elements

  a + 0 = a

  a . 1 = a

    0 is the identity element for the + operation.

    1 is the identity element for the . operation.

Commutativity and Associativity of the Operators

- The Commutative Property:

  $a + b = b + a$

  $a \cdot b = b \cdot a$

- The Associative Property:

  $a + (b + c) = (a + b) + c$

  $a \cdot (b \cdot c) = (a \cdot b) \cdot c$

- The Distributive Property:

  $a + (b . c) = (a + b) . (a + c)$

  $a . (b + c) = (a . b) + (a . c)$

- The Existence of the Complement:

  For every a there exists a unique element called a’ (*complement of a*) such that,

  $a + a' = 1$

  $a . a' = 0$

- To simplify notation, the . operator is frequently omitted. When two elements are written next to each other, the AND (.) operator is implied

$$a + b \cdot c = (a + b) \cdot (a + c)$$

$$a + bc = (a + b)(a + c)$$

Duality

- The principle of *duality* says that if an expression is valid in Boolean algebra, the dual of that expression is also valid.
- To form the dual of an expression, replace all + operators with . operators, all . operators with + operators
- Form the dual of the expression F=a + (bc)

  Dual of F is (a + b)(a + c)

Involution

- Taking the double inverse of a value will give the initial value.

Absorption

- This theorem states:

  $a + ab = a$ $\qquad\qquad\qquad\qquad$ $a(a+b) = a$

- To prove the first half of this theorem:

$$
\begin{aligned}
a + ab &= a \,.\, 1 + ab \\
&= a\,(1 + b) \\
&= a\,(b + 1) \\
&= a\,(1) \\
a + ab &= a
\end{aligned}
$$

DeMorgan’s Theorem

- A key theorem in simplifying Boolean algebra expression is DeMorgan’s Theorem. It states:

  $(a + b)' = a'b'$

  $(ab)' = a' + b'$

- Find the complement of:

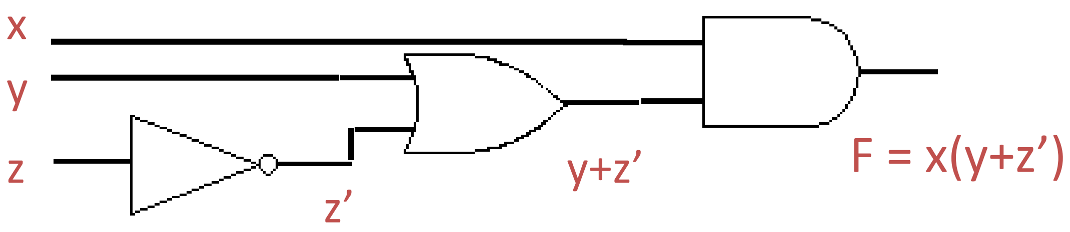
$$F = (AB' + C)D' + E$$

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

Boolean Functions

- Boolean algebra deals with binary variables and logic operations.
- Function results in binary 0 or 1

x	y	z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



F = x(y+z’)

Simplifying Boolean Functions

- Simplify the following Boolean function to a minimum number of terms: $F = xy + x'z + yz$

$$
\begin{aligned}
F_3 &= xy + x'z + yz \\
&= xy + x'z + yz(x + x') \\
&= xy + x'z + xyz + x'yz \\
&= xy(1 + z) + x'z(1 + y) \\
&= xy + x'z
\end{aligned}
$$

- ***Any Boolean Expression can be represented in two forms:***
  - ***sum of products (SOP)***
  - ***Product of Sum(POS)***

SOP Form

- Each variable in a Boolean expression is a literal
- Boolean variables can appear in normal (x) or complement form (x')
- Each AND combination of terms is a minterm

Minterms

x	y	z	Minterm	
0	0	0	x'y'z'	m_0
0	0	1	x'y'z	m_1
…				
1	0	0	xy'z'	m_4
…				
1	1	1	xyz	m_7

POS form

- Each OR combination of terms is a <u>maxterm</u>

Maxterms

x	y	z	Maxterm	
0	0	0	x+y+z	M_0
0	0	1	x+y+z'	M_1
…				
1	0	0	x'+y+z	M_4
…				
1	1	1	x'+y'+z'	M_7

- Note that each maxterm is the complement of its corresponding minterm and vice versa.

Minterms and Maxterms for Three Binary Variables

x	y	z	minterms		Maxterms	
0	0	0	$x'y'z'$	m_0	$x+y+z$	M_0
0	0	1	$x'y'z$	m_1	$x+y+z'$	M_1
0	1	0	$x'yz'$	m_2	$x+y'+z$	M_2
0	1	1	$x'yz$	m_3	$x+y'+z'$	M_3
1	0	0	$xy'z'$	m_4	$x'+y+z$	M_4
1	0	1	$xy'z$	m_5	$x'+y+z'$	M_5
1	1	0	xyz'	m_6	$x'+y'+z$	M_6
1	1	1	xyz	m_7	$x'+y'+z'$	M_7

Given the truth table, express $F1$ in sum of minterms

x	y	z	F_1	F_2
0	0	0	0	1
0	0	1	$\boxed{1}$	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

$$F_1(x, y, z) = \sum(1,4,5,6,7) = m_1 + m_4 + m_5 + m_6 + m_7$$
$$= (x'y'z) + (xy'z') + (xy'z) + (xyz') + (xyz)$$

- For product of maxterms:

x	y	z	F_1	F_2
0	0	0	$\boxed{0}$	1
0	0	1	1	0
0	1	0	$\boxed{0}$	1
0	1	1	$\boxed{0}$	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

$$F_1(x, y, z) = \Pi(0,2,3) = M_0 \cdot M_2 \cdot M_3$$

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

Minimization with Karnaugh Maps

Gate-Level Minimization

- The Boolean functions also can be simplified by map method as Karnaugh map or K-map.
- The map is made up of squares, with each square representing one minterm of the function.
- This produces a circuit diagram with a minimum number of gates and the minimum number of inputs to the gate.
- It is sometimes possible to find two or more expressions that satisfy the minimization criteria.

- A Kmap is a matrix consisting of rows and columns that represent the output values of a Boolean function.
- The output values placed in each cell are derived from the minterms of a Boolean function.
- A *minterm* is a product term that contains all of the function’s variables exactly once, either complemented or not complemented.

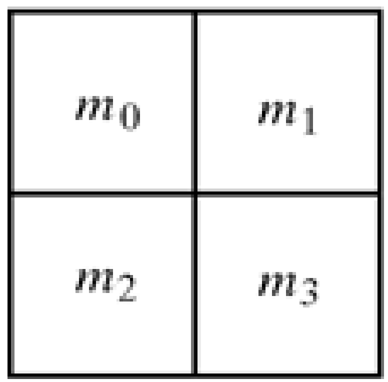
- K-maps are often used to simplify logic problems with 2, 3 or 4 variables.
- **Cell = 2^n ,where n is a number of variables**

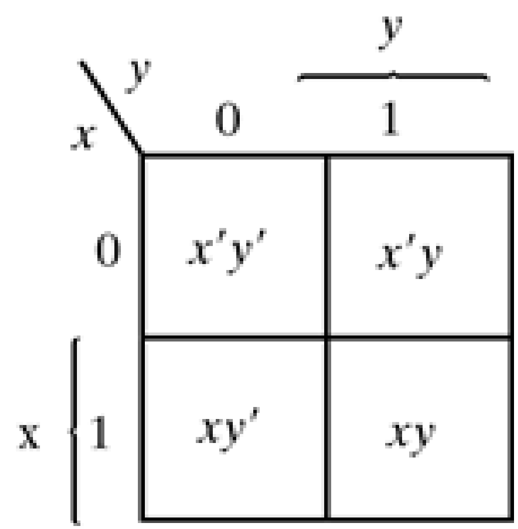
Two-Variable K map

- A two-variable function has four possible minterms. We can re-arrange these minterms into a Karnaugh map (K-map).

x	y	minterm
0	0	x'y'
0	1	x'y
1	0	xy'
1	1	xy

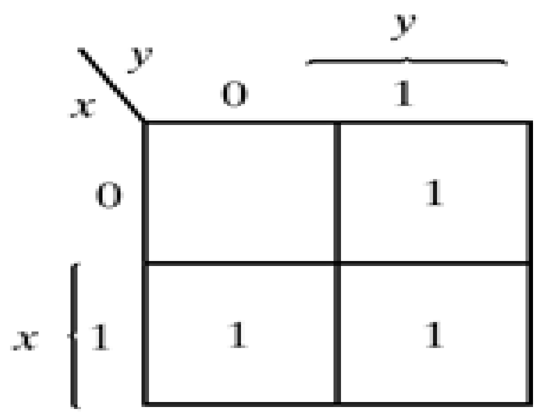
x \ y	0	1
0	x'y'	x'y
1	xy'	xy


m_0
m_1
m_2
m_3


y
x
y
0
1
0
1
x
$x'y'$
$x'y$
xy'
xy

- For the Boolean expression

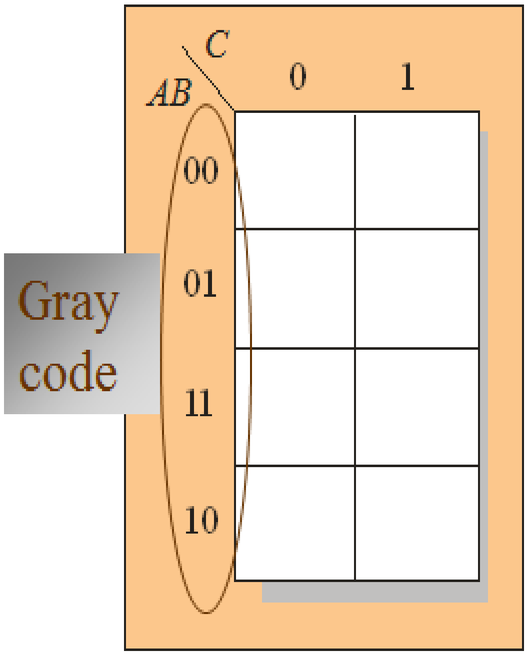
  $m_1 + m_2 + m_3 = x'y + xy' + xy$, 2 variable K map is



Three-Variable K map

- For a three-variable expression with inputs x, y, z, the arrangement of minterms follows Gray code.
- For simplifying Boolean functions, we must recognize the basic property possessed by adjacent squares.
- Each cell differs from an adjacent cell by only one variable

- Cells are usually labeled using 0’s and 1’s to represent the variable and its complement
- The numbers are entered in gray code, to force adjacent cells to be different by only one variable.
- Ones are read as the true variable and zeros are read as the complemented variable.



A	B	C	F
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	1

$A \backslash BC$	00	01	11	10
0	0	1	0	1
1	1	1	1	1

$$F = AB'C' + AB'C + ABC + ABC' + A'B'C + A'BC'$$

- Minterm which are identical, except for one variable, are considered to be adjacent to one another.
- $m_5+m_7= xy'z + xyz = xz(y' + y) = xz$

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

x \ yz	00	01	11	10
0	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
1	$xy'z'$	$xy'z$	xyz	xyz'

- With this ordering, any group of 2, 4 or 8 adjacent squares on the map contains common literals that can be factored out.

			y	y
	x'y'z'	x'y'z	x'yz	x'yz'
x	xy'z'	xy'z	xyz	xyz'
		z	z	

$$
\begin{aligned}
& x'y'z' + xy'z' + x'yz' + xyz' \\
= \ & z'(x'y' + xy' + x'y + xy) \\
= \ & z'(y'(x' + x) + y(x' + x)) \\
= \ & z'(y'+y) \\
= \ & z'
\end{aligned}
$$

Four-variable K map

Can accommodate 16 minterms that are produced by a four-input function.

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

$wx \backslash yz$	00	01	11	10
00	$w'x'y'z'$	$w'x'y'z$	$w'x'yz$	$w'x'yz'$
01	$w'xy'z'$	$w'xy'z$	$w'xyz$	$w'xyz'$
11	$wxy'z'$	$wxy'z$	$wxyz$	$wxyz'$
10	$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$

- $F = \sum m(0, 2, 3, 5, 6, 7, 8, 10, 11, 14, 15)$

The function can be written as:

$$F = A'B'C'D' + A'B'CD' + A'B'CD + A'BC'D + A'BCD' + A'BCD + AB'C'D' + AB'CD' + AB'CD + ABCD' + ABCD$$

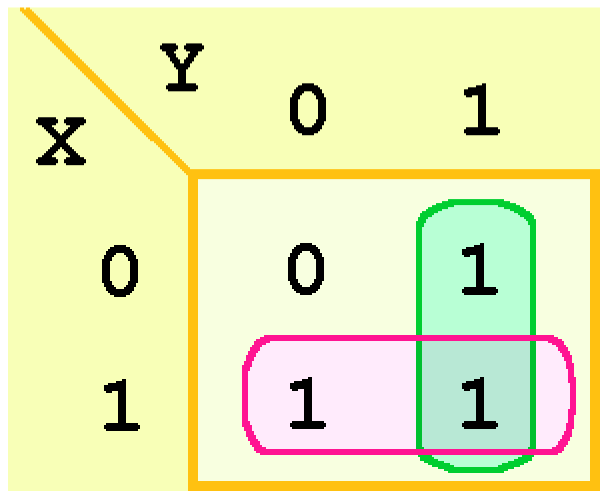
AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	0	0	1	1
10	1	0	1	1

Insert 1 in those cells where the function F has a value of 1. Put 0 in the other cells.

Simplification using 2 variable K map

- Is done by finding adjacent 1s in the Kmap that can be collected into groups that are powers of two.

- The rules of Kmap simplification are:
  - Groupings can contain only 1s; no 0s.
  - Groups can be formed only at right angles; diagonal groups are not allowed.
  - The number of 1s in a group must be a power of 2 – even if it contains a single 1.
  - The groups must be made as large as possible.
  - Groups can overlap and wrap around the sides of the Kmap.


Y
X
0
1
0
1
0
1
1
1

Simplification using 3 variable K map

$$F(X,Y,Z) = \bar{X}\bar{Y}\bar{Z} + \bar{X}\bar{Y}Z + \bar{X}YZ + \bar{X}Y\bar{Z} + X\bar{Y}\bar{Z} + XY\bar{Z}$$

X \ YZ	00	01	11	10
0	1	1	1	1
1	1	0	0	1

X \ YZ	00	01	11	10
0	1	1	1	1
1	1	0	0	1

reduced function is

$$F(X,Y,Z) = \bar{X} + \bar{Z}$$

$$F = AB'C' + AB'C + ABC + ABC' + A'B'C + A'BC'$$

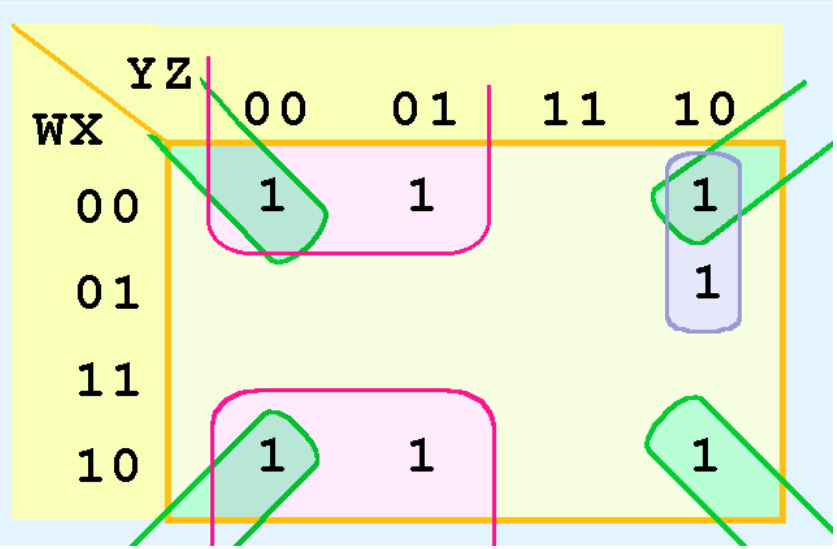
A \ BC	00	01	11	10
0	0	1	0	1
1	1	1	1	1

$$F = A + B'C + BC'$$

Simplification using 4 variable K map

$$F(W,X,Y,Z)=\bar{W}\bar{X}\bar{Y}\bar{Z}+\bar{W}\bar{X}\bar{Y}Z+\bar{W}\bar{X}Y\bar{Z}+\bar{W}XY\bar{Z}+W\bar{X}\bar{Y}\bar{Z}+W\bar{X}\bar{Y}Z+W\bar{X}Y\bar{Z}$$

WX \ YZ	00	01	11	10
00	1	1		1
01				1
11				
10	1	1		1



$$F(W,X,Y,Z)=\bar{W}\bar{Y}+\bar{X}\bar{Z}+\bar{W}Y\bar{Z}$$

Don’t Care Conditions

- Real circuits don’t always need to have an output defined for every possible input.
- If a circuit is designed so that a particular set of inputs can never happen, we call this set of inputs a *don’t care* condition.
- In a Kmap, a don’t care condition is identified by an *X* in the cell of the minterm(s) for the don’t care inputs
- In performing the simplification, we are free to include or ignore the *X*’s when creating our groups.

Karnaugh maps: Don't cares

- f(A,B,C,D) = Σ m(1,3,5,7,9) + d(6,12,13)

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

- In some situations, we don’t care about the value of a function for certain combinations of the variables.
  - these combinations may be impossible in certain contexts
  - or the value of the function may not matter in when the combinations occur

Map Simplification with Don’t Cares

AB \ CD	00	01	11	10
00	0	1	0	0
01	x	x	x	1
11	1	1	1	x
10	x	0	1	1

$F = A'C'D + B + AC$

Alternative covering.

AB \ CD	00	01	11	10
00	0	1	0	0
01	x	x	x	1
11	1	1	1	x
10	x	0	1	1

$F = A'B'C'D + ABC' + BC + AC$

Example

- Use a K-Map to simplify the following Boolean expression

$$F(a,b,c,d) = \sum m(0,2,6,8,12,13,15) + d(3,4,9)$$

$$F(a,b,c,d)=\sum m(0,2,6,8,12,13,15)+d(3,4,9)$$

cd \ ab	00	01	11	10
00	1	d	1	1
01			1	d
11	d		1	
10	1	1		

cd \ ab	00	01	11	10
00	1	d	1	1
01			1	d
11	d		1	
10	1	1		

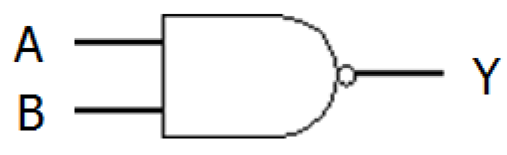
$$F = a\bar{c} + \bar{a}\bar{d} + abd$$

Universal Logic Gates

- NAND
- NOR

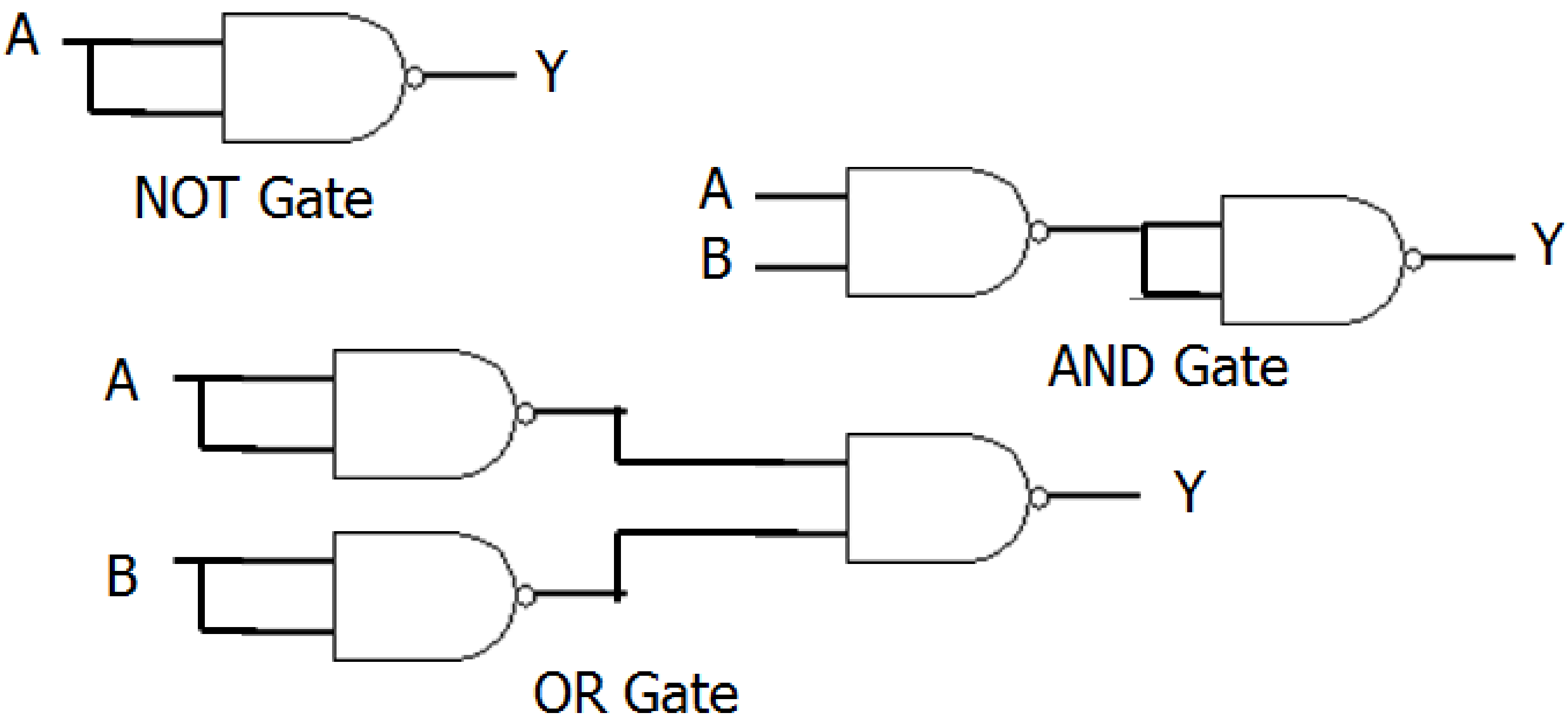
The NAND Gate

- Therefore, we can use a NAND gate to implement all three of the *elementary operators* (AND,OR,NOT).
- Therefore, ANY switching function can be constructed using only NAND gates. Such a gate is said to be *primitive* or *functionally complete*.



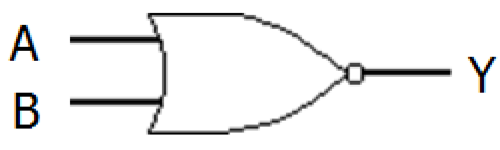
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NAND Gates into Other Gates



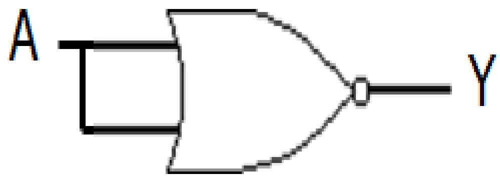
The NOR Gate

- Just like the NAND gate, the NOR gate is functionally complete, ie, any logic function can be implemented using just NOR gates.

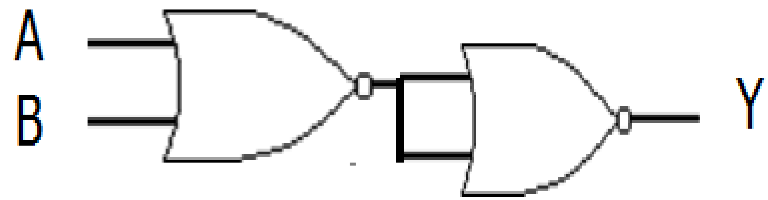


A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

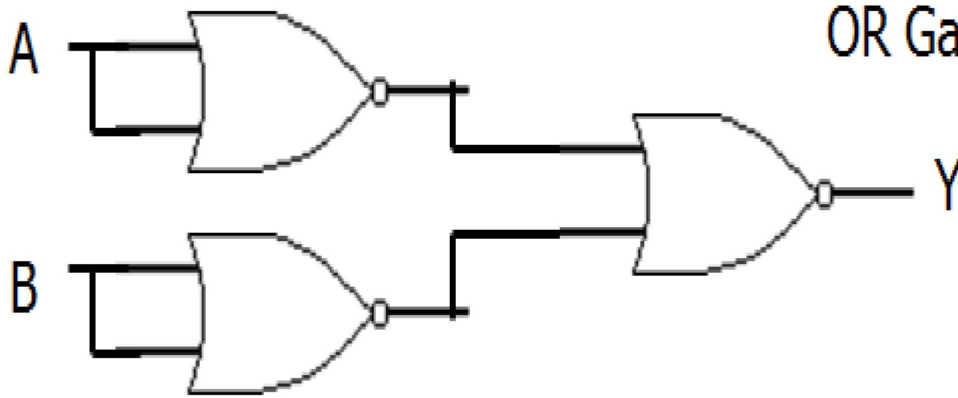
NOR Gates into Other Gates



NOT Gate

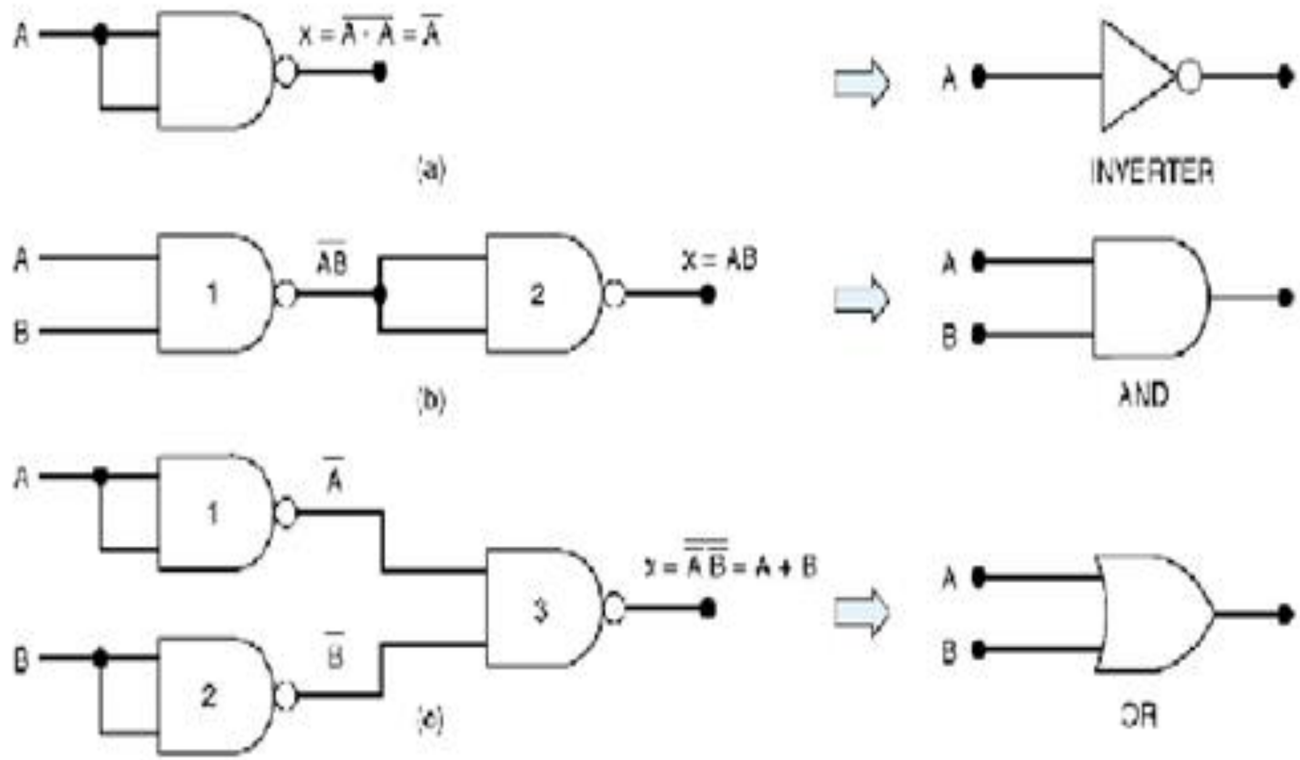


OR Gate

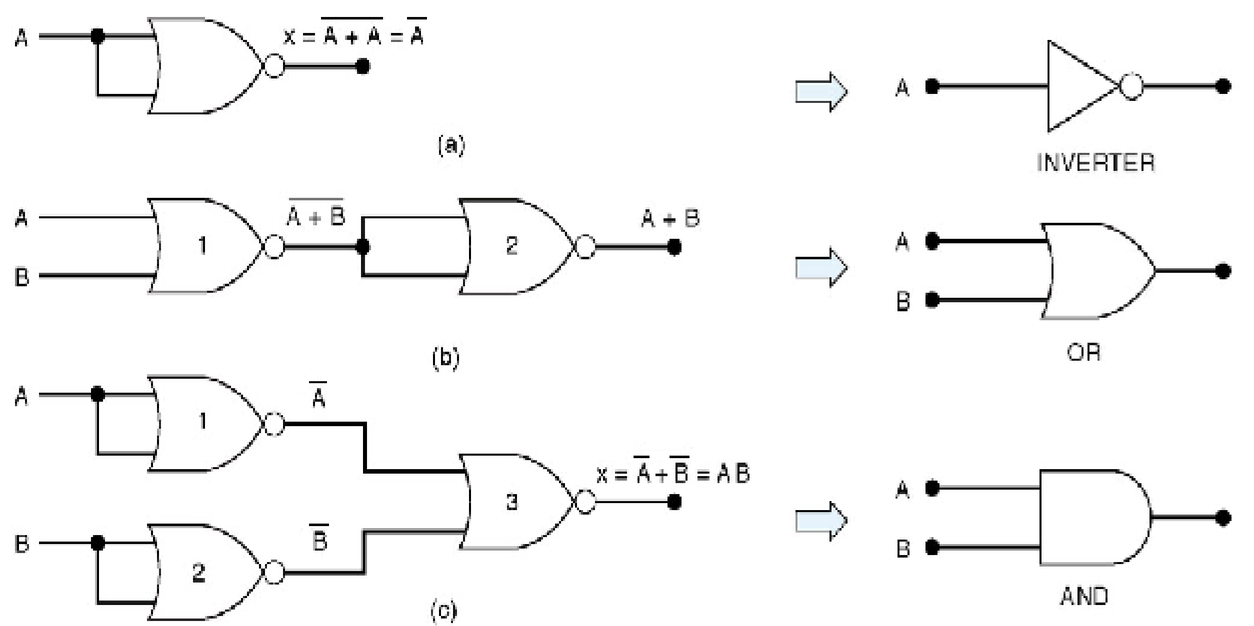


AND Gate

Universality of NAND gates



Universality of NOR gate



- Boolean algebra defines how binary variables with NAND, NOR can be combined
- DeMorgan's rules are important.
  - Allow conversion to NAND/NOR representations