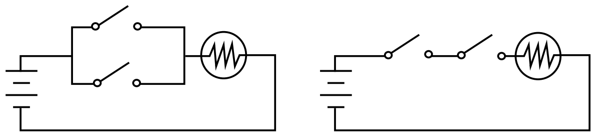
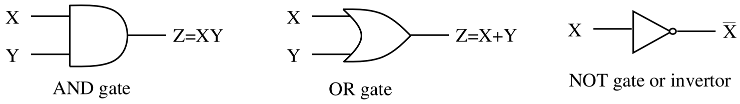
Logic Gates

- **Logic gates:** physical devices that implement binary logic functions.

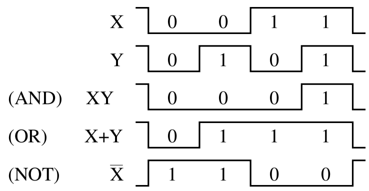


switches in parallel => OR switches in series => AND

- **Logic gate symbols:** physical devices that implement binary logic functions.



(a) Graphic symbols

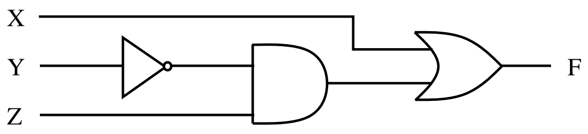


(b) Timing diagram

Logic Gate Diagram

- Boolean equations, truth tables, and logic diagrams express the same function.

Expression: $F = X + \overline{Y}Z$



Logic Circuit Diagram for $F = X + \overline{Y}Z$

Truth table for the function $F = X + \overline{Y}Z$

X	Y	Z	F
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

Boolean Algebra

- Boolean algebra (named after G. Boole for his work in 1854) is a mathematical system for specifying and transforming logic functions.

- **Boolean (switching) algebra:** An algebra structure defined on a set of elements, B, together with two binary operators (denoted $+$ and $\cdot$) provided the following **axioms** are satisfied:
  1. Closure with respect to (wrt) $+$ and $\cdot$
  2. Identity elements exist for $+$ and $\cdot$
  3. Commutative wrt $+$ and $\cdot$
  4. Operator $+$ distributes over $\cdot$, and $\cdot$ distributes over $+$.
  5. Every element of B has a complement.
  6. B has at least two distinct elements.

- Purpose: Studying Boolean algebra to specify the design of digital systems!

Algebric Properties: Axioms 1–3 (by Huntington, 1904)

1. **Closure Property**: A set S is **closed** wrt to a binary operator $\bullet$ if and only if (iff) for every $x, y \in S, x \bullet y \in S$.
   - **Axiom 1(a):** B is closed wrt the operator $+$
   - **Axiom 1(b):** B is closed wrt the operator $\cdot$

2. **Identity Element**: A set S is said to have an **identity element** wrt to a particular binary operator $\bullet$ whenever there exists an element $e \in S$ such that for every $x \in S, e \bullet x = x \bullet e = x$.
   - **Axiom 2(a):** B has an identity element wrt $+$, denoted by 0
   - **Axiom 2(b):** B has an identity element wrt $\cdot$, denoted by 1

3. **Commutativity Property**: A binary operator $\bullet$ defined on a set S is said to be **commutative** iff for every $x, y \in S, x \bullet y = y \bullet x$.
   - **Axiom 3(a):** B is commutative wrt the operator $+$
   - **Axiom 3(b):** B is commutative wrt the operator $\cdot$

Algebric Properties: Axioms 4–6

4. **Distributivity Property**: If $\bullet$ and $\diamond$ are two binary operators on a set S, $\bullet$ is said to be **distributive** over $\diamond$ if for all $x, y, z \in S, x \bullet (y \diamond z) = (x \bullet y) \diamond (x \bullet z)$.
   - **Axiom 4(a):** The operator $\cdot$ is distributive over $+$; **Axiom 4(b):** The operator $+$ is distributive over $\cdot$.
5. **Complement Element**: For every $x \in B$, there exists an element $x' \in B$ such that $x + x' = 1$ and $x \cdot x' = 0$. x' is called the **complement** of x.
6. **Cardinality Bound**: There are at least two elements $x, y \in B$ such that $x \neq y$.

- Other notes:
  - The **Associative Law** $x + (y + z) = (x + y) + z$ and $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ hold and can be derived from the above axioms.
  - Axioms 4(b) and 5 are not available in ordinary algebra.
  - Boolean algebra applies to a finite set of elements; ordinary algebra applies to the infinite set of real numbers.

Duality and Basic Identities

- **Duality Property:** Every algebraic expression deducible from the previous axioms remains valid after **dual** operations.
  - The **DUAL** of an algebraic expression is formed by interchanging AND $(\cdot)$ $\longleftrightarrow$ OR $(+)$ and 0 $\longleftrightarrow$ 1.
  - Exp: $x + y = y + x \longleftrightarrow x \cdot y = y \cdot x$; $x + x' = 1 \longleftrightarrow x \cdot x' = 0$.
- Identities of Boolean algebra:

1.	$x + 0 = x$	2.	$x \cdot 1 = x$	
3.	$x + 1 = 1$	4.	$x \cdot 0 = 0$	
5.	$x + x = x$	6.	$x \cdot x = x$	
7.	$x + \bar{x} = 1$	8.	$x \cdot \bar{x} = 0$	
9.	$\bar{\bar{x}} = x$			
10.	$x + y = y + x$	11.	$xy = yx$	Commutative
12.	$x + (y + z) = (x + y) + z$	13	$x(yz) = (xy)z$	Associative
14.	$x(y + z) = xy + xz$	15.	$x + yz = (x + y)(x + z)$	Distributive
16.	$\overline{x + y} = \bar{x} \cdot \bar{y}$	17.	$\overline{x \cdot y} = \bar{x} + \bar{y}$	DeMorgan's
18.	$x + xy = x$	19.	$x(x + y) = x$	Absorption
20.	$xy + x\bar{y} = x$	21.	$(x + y)(x + \bar{y}) = x$	
22.	$x + \bar{x}y = x + y$	23.	$x(\bar{x} + y) = xy$	

- **Operator precedence:** parenthese () $\rightarrow$ NOT $\rightarrow$ AND $\rightarrow$ OR.

DeMorgan's Law

- Truth tables to verify DeMorgan's Law: $\overline{X+Y} = \bar{X} \cdot \bar{Y}$

A)	X	Y	X+Y	$\overline{\text{X+Y}}$	B)	X	Y	$\overline{\text{X}}$	$\overline{\text{Y}}$	$\overline{\text{X}}\bullet\overline{\text{Y}}$
	0	0	0	1		0	0	1	1	1
	0	1	1	0		0	1	1	0	0
	1	0	1	0		1	0	0	1	0
	1	1	1	0		1	1	0	0	0

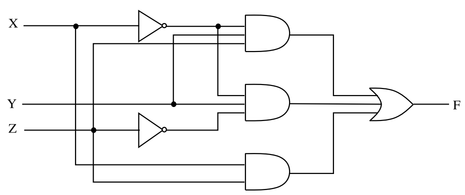
- DeMorgan's Law can be extended to multiple variables.
  - $\overline{X_1 + X_2 + \ldots + X_n} = \bar{X}_1 \bar{X}_2 \ldots \bar{X}_n$.
  - $\overline{X_1 X_2 \ldots X_n} = \bar{X}_1 + \bar{X}_2 + \ldots + \bar{X}_n$.

Algebraic Manipulation to Minimize Literals

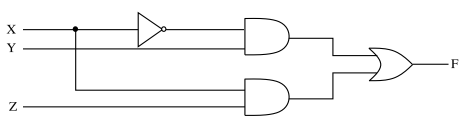
- **Literal:** A single variable within a **term** that may or may not be complemented.
- Minimize # of terms and literals: $\bar{x}yz + \bar{x}y\bar{z} + xz$

$$
\begin{aligned}
F &= \bar{x}yz + \bar{x}y\bar{z} + xz \\
&= \bar{x}y(z + \bar{z}) + xz && \text{Identity 14} \\
&= \bar{x}y \cdot 1 + xz && \text{Identity 7} \\
&= \bar{x}y + xz && \text{Identity 2}
\end{aligned}
$$

- # of terms reduced from 3 $(\bar{x}yz, \bar{x}y\bar{z}, xz)$ to 2 $(\bar{x}y, xz)$.
- # of literals reduced from 8 $(\bar{x}, y, z, \bar{x}, y, \bar{z}, x, z)$ to 4 $(\bar{x}, y, x, z)$.



(a) $F=\bar{X}YZ+\bar{X}Y\bar{Z}+XZ$



(b) $F=\bar{X}Y+XZ$

Consensus Theorem

- $xy + \bar{x}z + yz = xy + \bar{x}z$

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

- Dual: $(x + y)(\bar{x} + z)(y + z) = (x + y)(\bar{x} + z)$
- Exp: $(A + B)(\bar{A} + C) = AC + \bar{A}B$

$$\begin{aligned} (A + B)(\bar{A} + C) &= A\bar{A} + AC + \bar{A}B + BC \\ &= AC + \bar{A}B + BC \\ &= AC + \bar{A}B \end{aligned}$$

Function Complement

- Use DeMorgan's Law to complement a function
  1. Interchange AND and OR operators (**duality operation**).
  2. Complement each literal.
- Applying DeMorgan's Law: $F = \bar{X}Y\bar{Z} + \bar{X}\bar{Y}Z$

$$\begin{aligned} \bar{F} &= \overline{\bar{X}Y\bar{Z} + \bar{X}\bar{Y}Z} = \overline{\bar{X}Y\bar{Z}} \cdot \overline{\bar{X}\bar{Y}Z} \\ &= (X + \bar{Y} + Z)(X + Y + \bar{Z}) \end{aligned}$$

- Taking duals and complementing literals: $F = \bar{X}Y\bar{Z} + \bar{X}\bar{Y}Z$

$$\begin{aligned} F &= \bar{X}Y\bar{Z} + \bar{X}\bar{Y}Z \\ \text{Taking dual:} \quad & (\bar{X} + Y + \bar{Z})(\bar{X} + \bar{Y} + Z) \\ \text{Complementing literals:} \quad & (X + \bar{Y} + Z)(X + Y + \bar{Z}) = \bar{F} \end{aligned}$$

Minterm and Maxterm

- **Minterms:** Terms with all variables present, combined with AND.
  1. For n variables combined with AND, there are 2^n combinations. Each unique combination is called a **minterm**.
  2. Exp: $X \cdot Y \cdot Z$, $\bar{a}bc\bar{d}$.
- **Maxterms:** Terms with all variables present, combined with OR.
  1. For n variables combined with OR, there are 2^n combinations. Each unique combination is called a **maxterm**.
  2. Exp: $X + Y + Z$, $\bar{a} + b + c + \bar{d}$.
- Exp: Two-variable terms

Index	Minterm	Maxterm
0	$\bar{x}\bar{y}$	$x + y$
1	$\bar{x}y$	$x + \bar{y}$
2	$x\bar{y}$	$\bar{x} + y$
3	xy	$\bar{x} + \bar{y}$

Standard Order

- Given n variables, we use an n-bit expansion of the index, i, to indicate normal (true) or complement states for the variables.
- Minterms: "1" $\Rightarrow$ true; "0" $\Rightarrow$ complemented.
  - m_0 (minterm 0): $\bar{x} \cdot \bar{y} \cdot \bar{z}$; m_3 (minterm 3): $\bar{x} \cdot y \cdot z$.
- Maxterms: "0" $\Rightarrow$ true; "1" $\Rightarrow$ complemented.
  - M_0 (maxterm 0): $x + y + z$; M_3 (maxterm 3): $x + \bar{y} + \bar{z}$.
- m_i is the **complement** of M_i ($m_i = \bar{M}_i$).

Minterms and Maxterms

- Minterms for three variables.

X	Y	Z	Product Term	Symbol	m0	m1	m2	m3	m4	m5	m6	m7
0	0	0	$\overline{X}\overline{Y}\overline{Z}$	m0	1	0	0	0	0	0	0	0
0	0	1	$\overline{X}\overline{Y}Z$	m1	0	1	0	0	0	0	0	0
0	1	0	$\overline{X}Y\overline{Z}$	m2	0	0	1	0	0	0	0	0
0	1	1	$\overline{X}YZ$	m3	0	0	0	1	0	0	0	0
1	0	0	$X\overline{Y}\overline{Z}$	m4	0	0	0	0	1	0	0	0
1	0	1	$X\overline{Y}Z$	m5	0	0	0	0	0	1	0	0
1	1	0	$XY\overline{Z}$	m6	0	0	0	0	0	0	1	0
1	1	1	XYZ	m7	0	0	0	0	0	0	0	1

- Maxterms for three variables.

X	Y	Z	Sum Term	Symbol	M0	M1	M2	M3	M4	M5	M6	M7
0	0	0	$X+Y+Z$	M0	0	1	1	1	1	1	1	1
0	0	1	$X+Y+\overline{Z}$	M1	1	0	1	1	1	1	1	1
0	1	0	$X+\overline{Y}+Z$	M2	1	1	0	1	1	1	1	1
0	1	1	$X+\overline{Y}+\overline{Z}$	M3	1	1	1	0	1	1	1	1
1	0	0	$\overline{X}+Y+Z$	M4	1	1	1	1	0	1	1	1
1	0	1	$\overline{X}+Y+\overline{Z}$	M5	1	1	1	1	1	0	1	1
1	1	0	$\overline{X}+\overline{Y}+Z$	M6	1	1	1	1	1	1	0	1
1	1	1	$\overline{X}+\overline{Y}+\overline{Z}$	M7	1	1	1	1	1	1	1	0

Sum of Minterms

- Any Boolean function can be expressed as a sum of minterms.
- Exp: $F = A + \bar{B}C$.
  - Expand the terms with missing variables and collect terms:

$$\begin{aligned} F &= A + \bar{B}C \\ &= A(B+\bar{B})(C+\bar{C}) + (A+\bar{A})\bar{B}C \\ &= ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + \bar{A}\bar{B}C \end{aligned}$$

  - Sum of minterms: $F = m_7 + m_6 + m_5 + m_4 + m_1 \Longrightarrow F(A,B,C) = \sum_m(1,4,5,6,7)$.
- The complement of a function contains those minterms not included in the original function.
  - $F(A,B,C) = \sum_m(1,4,5,6,7) \Longrightarrow \bar{F}(A,B,C) = \sum_m(0,2,3)$.

Product of Maxterms

- Any Boolean function can be expressed as a product of maxterms.
- Exp: $G = AB + \bar{A}\bar{B}$.

$$\begin{aligned} AB + \bar{A}\bar{B} &= (AB + \bar{A})(AB + \bar{B}) \\ &= (\bar{A} + AB)(\bar{B} + AB) \\ &= (\bar{A} + A)(\bar{A} + B)(\bar{B} + A)(\bar{B} + B) \\ &= 1 \cdot (\bar{A} + B)(\bar{B} + A) \cdot 1 \\ &= (\bar{A} + B)(\bar{B} + A) \\ &= M_2 \cdot M_1 \end{aligned}$$

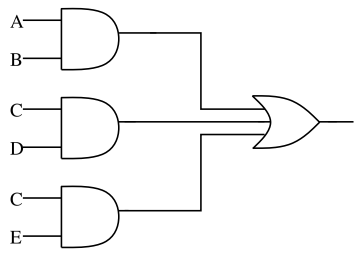
- Product of maxterms: $G = M_2 \cdot M_1 \Longrightarrow G(A,B) = \Pi_M(1,2)$.
- The complement of a function contains those maxterms not included in the original function.
  - $G(A,B) = \Pi_M(1,2) \Longrightarrow \bar{G}(A,B) = \Pi_M(0,3)$.
- $G(A,B) = \Pi_M(1,2) = \sum_m(0,3)$; $\bar{G}(A,B) = \Pi_M(0,3) = \sum_m(1,2)$.

Standard Forms

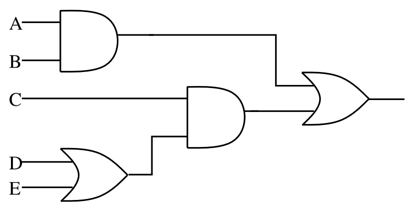
- **Canonical forms** (Sum-of-Minterms or Product-of-Maxterms) have **one and only one** representation.
  - Sum-of-Minterms: $xyz + \bar{x}y\bar{z} + x\bar{y}z$, $A\bar{B} + \bar{A}B$.
  - Product-of-Maxterms: $(x + y + z)(\bar{x} + y + \bar{z})(x + \bar{y} + z)$, $(A + \bar{B})(\bar{A} + B)$
- **Standard Sum-of-Products (SOP) form:** Equations are written as **AND** terms summed with **OR** operators.
  - SOPs: $xyz + \bar{x}y\bar{z} + \bar{y}$, $A\bar{B} + \bar{A}B$
- **Standard Product-of-Sums (POS) form:** Equations are written as **OR** terms, all **AND**ed together.
  - POSs: $(x + y + z)(\bar{x} + y + \bar{z})(\bar{y})$, $(A + \bar{B})(\bar{A} + B)$
- Mixed forms: **not** SOP or POS
  - $(x\bar{y} + z)(\bar{x} + y)$, $AB\bar{C} + C(\bar{A} + B)$

Canonical Sum of Minterms

- Two-level vs. three-level implementation.
  - Exp: $F = AB+CD+CE$ (2-level); $F = AB+C(D+E)$ (3-level)



(a) AB+CD+CE

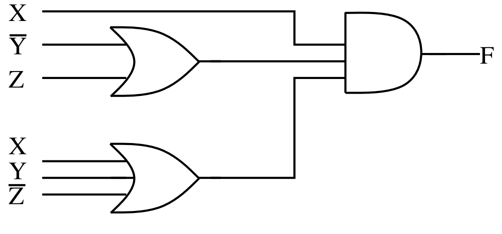


(b) AB+C(D+E)

- A canonical sum-of-minterms form implies a 2-level network of gates implementation (**1st level: AND; 2nd level: OR**).
  - Usually **not** a minimum literal Boolean expression $\Longrightarrow$ more expensive implementation.
  - Exp: $F = ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + \bar{A}\bar{B}C = A + \bar{B}C$ (Why? Duplicate $A\bar{B}C$!)
  - The canonical sum-of-minterms form had **15 literals** and **5 terms**; the reduced SOP form had **3 literals** and **2 terms**.

Canonical Product of Maxterms

- Two-level product-of-sums implementation.
  - Exp: $F = X(\bar{Y} + Z)(X + Y + \bar{Z})$



- A canonical product-of-maxterms form implies a 2-level network of gates implementation (**1st level: OR; 2nd level: AND**).
  - Usually **not** a minimum literal Boolean expression $\Longrightarrow$ more expensive implementation.
  - Exp: $F = (A+B+C)(A+\bar{B}+C)(A+\bar{B}+\bar{C}) = (A+C)(A+\bar{B})$ (Why? Duplicate $(A + \bar{B} + C)$!)
  - The canonical product-of-maxterms form had **9 literals** and **3 terms**; the reduced POS form had **4 literals** and **2 terms**.

Equivalent Literal Cost Network

- **Questions:**
  - Is there only one minimum cost network?
  - How can we obtain **<u>a</u>** or **<u>the</u>** minimum literal expression?
- Minimize $F(A,B,C) = \sum(0,2,3,4,5,7)$

$$\begin{aligned} F &= \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + ABC \\ &= \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + ABC + A\bar{B}\bar{C} + A\bar{B}C \\ &= \bar{A}\bar{C}(B+\bar{B}) + BC(\bar{A}+A) + A\bar{B}(C+\bar{C}) \\ &= \bar{A}\bar{C} + BC + A\bar{B} \end{aligned}$$

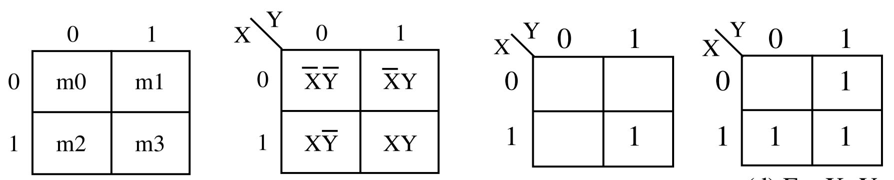
- Pairing F's terms differently

$$\begin{aligned} F &= \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + ABC \\ &= \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + \bar{A}BC + \bar{A}B\bar{C} + A\bar{B}C + ABC \\ &= \bar{B}\bar{C}(\bar{A}+A) + \bar{A}B(C+\bar{C}) + AC(B+\bar{B}) \\ &= \bar{B}\bar{C} + \bar{A}B + AC \end{aligned}$$

- **Both have the same numbers of literals and terms!**

Karnaugh Map Simplification

- Diagram is a collection of squares.
- Each square represents a minterm.
- Collection of squares is a graphical representation of the Boolean function.
- **Adjacent squares differ in only one variable.**
- Pattern recognition is used to derive alternative algebraic expressions for the same function.
- Exp: Two-variable map: $(m_0, m_1), (m_0, m_2), (m_1, m_3), (m_2, m_3)$ are adjacent pairs.
  - $F(X,Y) = \sum(1,2,3) = \bar{X}Y + X\bar{Y} + XY = X + Y$.



(a) (b) (c) XY (d) F = X+Y

Three-Variable Maps

m0	m1	m3	m2
m4	m5	m7	m6

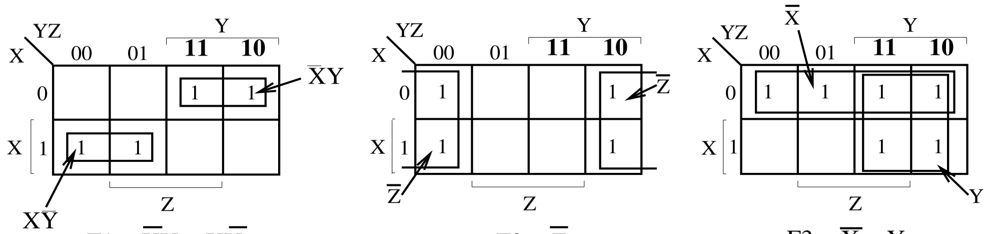
X \ YZ	00	01	11 (Y)	10 (Y)
0	$\bar{X}\bar{Y}\bar{Z}$	$\bar{X}\bar{Y}Z$	$\bar{X}YZ$	$\bar{X}Y\bar{Z}$
X 1	$X\bar{Y}\bar{Z}$	$X\bar{Y}Z$	XYZ	$XY\bar{Z}$

Z

- If the binary value for an index differs in one bit position, the minterms are adjacent.
  - $(m_0, m_2), (m_4, m_6)$, etc are adjacent pairs.
- For 3-variable maps, combining squares:
  - One square $\leftrightarrow$ a minterm with 3 variables.
  - Two squares $\leftrightarrow$ a product term with 2 variables.
  - Four squares $\leftrightarrow$ a product term with 1 variable.
  - Eight squares $\leftrightarrow$ logic 1.

Three-Variable Map Example

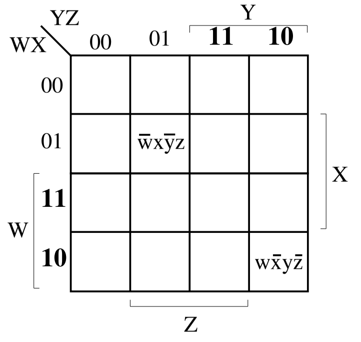
- Two squares: $F(X,Y,Z) = \sum(2,3) = \bar{X}Y$
- Four squares: $G(X,Y,Z) = \sum(0,2,4,6) = \bar{Z}$
- Four squares: $H(X,Y,Z) = \sum(2,3,6,7) = Y$
- Some 3-variable functions:



F1 = $\bar{X}$Y + X$\bar{Y}$ F2 = $\bar{Z}$ F3 = $\bar{X}$ + Y

Four-Variable Maps

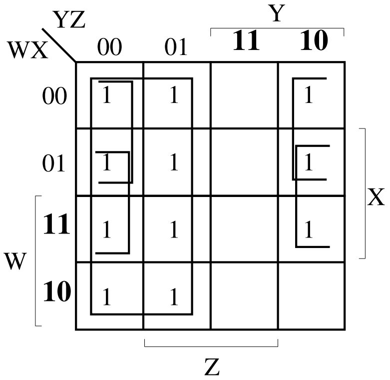
m0	m1	m3	m2
m4	m5	m7	m6
m12	m13	m15	m14
m8	m9	m11	m10



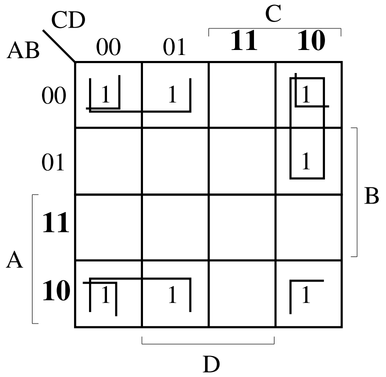
- If the binary value for an index differs in one bit position, the minterms are adjacent.
  - $(m_0, m_2), (m_0, m_8), (m_1, m_9)$, etc are adjacent pairs.
- For 4-variable maps, combining squares:
  - One square $\leftrightarrow$ a minterm with 4 variables.
  - Two squares $\leftrightarrow$ a product term with 3 variables.
  - Four squares $\leftrightarrow$ a product term with 2 variables.
  - Eight squares $\leftrightarrow$ a product term with 1 variable.
  - Sixteen squares $\leftrightarrow$ logic 1.

Four-Variable Map Examples

- $F(W,X,Y,Z) = \sum(0,1,2,4,5,6,8,9,12,13,14) = \bar{Y} + \bar{W}\bar{Z} + X\bar{Z}$.
- $G(A,B,C,D) = \bar{A}\bar{B}\bar{C} + \bar{B}C\bar{D} + A\bar{B}\bar{C} + \bar{A}BC\bar{D}$.



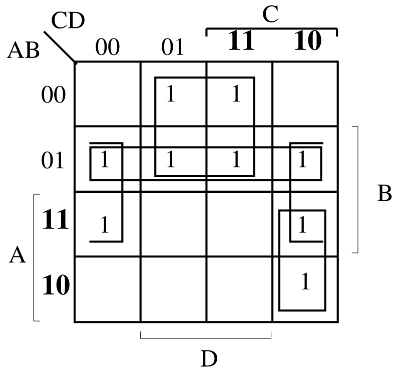
$F = \bar{Y} + \bar{W}\bar{Z} + X\bar{Z}$



$G = \bar{B}\bar{D} + \bar{B}\bar{C} + \bar{A}C\bar{D}$

Prime Implicant

- A product term is an **implicant** of a function F if F has the value 1 for all minterms of the product term.
- **Prime Implicant:** a product term obtained by combining the maximum possible number of adjacent squares in the map.
  - Prime implicants: $\bar{A}D, B\bar{D}, \bar{A}B, AC\bar{D}$
- **Essential Prime Implicant:** the **only prime implicant** that covers one or more terms.
  - Essential prime implicants: $\bar{A}D, B\bar{D}, AC\bar{D}$
- A set of prime implicants that **covers all minterms**: For each minterm, there is at least one selected prime implicant that covers the minterm.

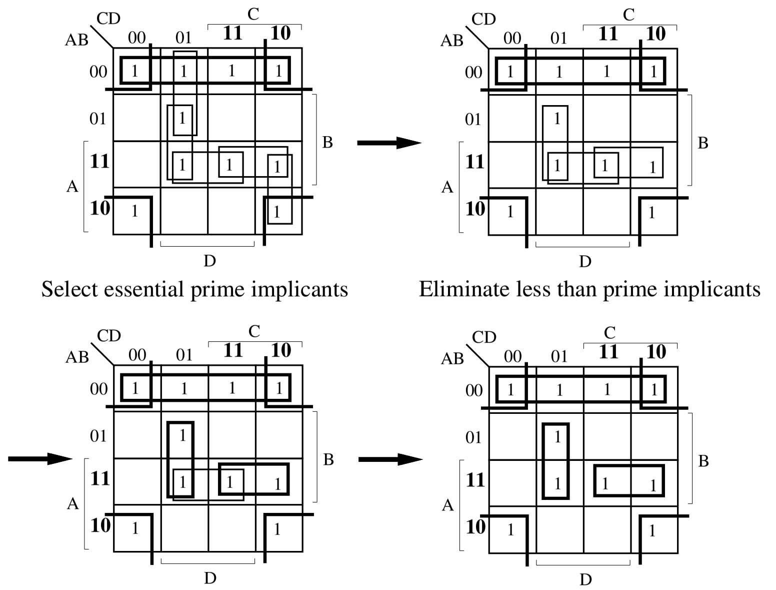


Systematic Simplification

1. Select all essential prime implicants.
2. Find and delete all <u>less than</u> prime implicants.
3. Repeat 1 and 2 until all minterms are covered.
4. If cycles occur
   4. Arbitrarily select a prime implicant and generate a cover.
   5. Delete the selected prime implicant and generate a new cover.
   6. Select the cover with fewer literals.
   7. If a new cycle occurs, repeat Steps 4, 5, and 6.

- **<u>Less than</u>** prime implicants: P_i is <u>less than</u> P_j if P_i contains at least as many literals as P_j and P_j covers at least all of the yet uncovered minterms that P_i covers.
- **<u>Secondary essential</u>** prime implicants: Once the less than prime implicants are removed from consideration, new prime implicants become essential and are called secondary essential prime implicants.
- **<u>Redundant</u>** prime implicants: Prime implicants whose minterms have been completely covered by the prime implicants selected and are removed from consideration.

Simplification Example



Select essential prime implicants

Eliminate less than prime implicants

Select secondary essential prime implicants

Remove redundant prime implicants

Higher Order K-Maps

- For five variables, use **two adjacent K-maps**.

V = 0 (WX \ YZ)

m0	m1	m3	m2
m4	m5	m7	m6
m12	m13	m15	m14
m8	m9	m11	m10

V = 1 (WX \ YZ)

m16	m17	m19	m18
m20	m21	m23	m22
m28	m29	m31	m30
m24	m25	m27	m26

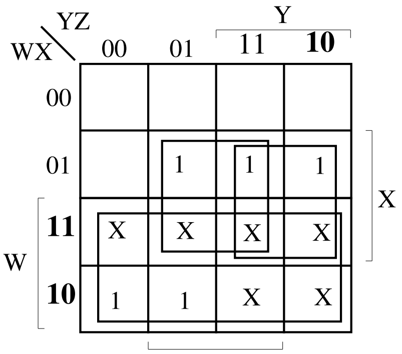
- Can extend the problem to six variables by using **4 K-maps**.

Don't Cares in K-Maps

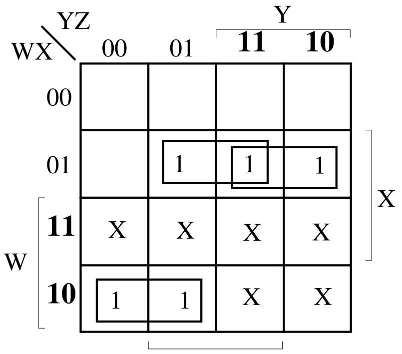
- **Incompletely specified functions:** Functions that have unspecified outputs for some input combinations.
- **Don't-care conditions:** Unspecified terms of a function; they might due to that their input values will never occur.
- By including don't cares in the map, we might arrive at a lower cost logic network.
- Don't cares are usually denoted with an **X**.
- Don't cares may be treated as either 1 or 0, depending on the desired minimization strategy.
- Example: Symbols 1010, 1011, 1100, 1101, 1110, 1111 will never occur as BCD digits. So they are don't cares.

Don't Care Examples

- $F(W,X,Y,Z)$: 5 or greater over a BCD input.
- Considering don't cares $d(W,X,Y,Z) = \sum_d(10,11,12,13,14,15) \Longrightarrow F_1(W,X,Y,Z) = W + XZ + XY$.
- **Without** considering don't cares $\Longrightarrow F_2(W,X,Y,Z) = \bar{W}XZ + \bar{W}XY + W\bar{X}\bar{Y}$.
- Don't cares contribute to making the function F simpler.



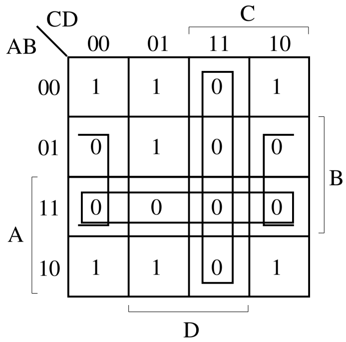
Considering don't cares



Treat don't cares as 0's.

Product-of-Sums Simplification

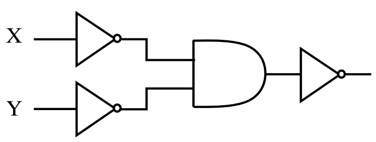
- How to implement a minimal literal, product-of-sums form?
  1. Find the **complement** of a function in sum-of-products form.
  2. Use DeMorgan's Law to complement the function and convert it to product-of-sums form.
- Implementing the complement of a function means using the **zeros** on the K-Map.
- Exp: $F(A,B,C,D) = \sum_m (0,1,2,5,8,9,10) \Longrightarrow \bar{F} = AB + CD + B\bar{D} \Longrightarrow F = (\bar{A}+\bar{B})(\bar{C}+\bar{D})(\bar{B}+D)$.

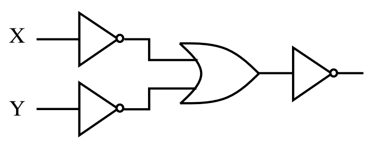


Functionally Complete Sets

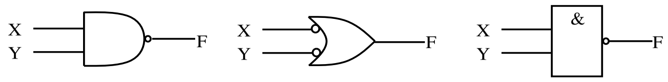
- A set of logic operations is said to be **functionally complete** if any Boolean function can be expressed in terms of the set of operations.
- We could implement any Boolean equation with the three primitives: AND, OR, NOT.
- To prove a set of logic operations is functionally complete, need to show AND, OR, NOT can be implemented by the operations defined in the set.
- Sets {NOT, AND}, {NOT, OR} are functionally complete, but set {AND, OR} is not.

$X+Y = \overline{\overline{X+Y}} = \overline{\bar{X}\bar{Y}}$ $\qquad$ $XY = \overline{\overline{XY}} = \overline{\bar{X}+\bar{Y}}$





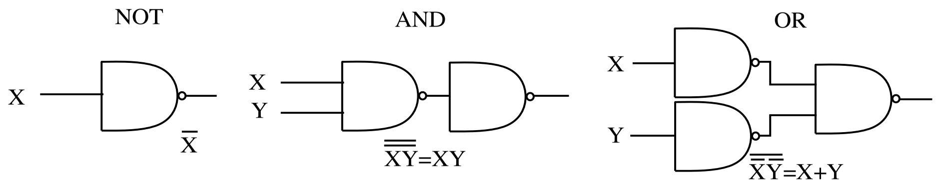
NAND Gates



$F = \overline{XY}$

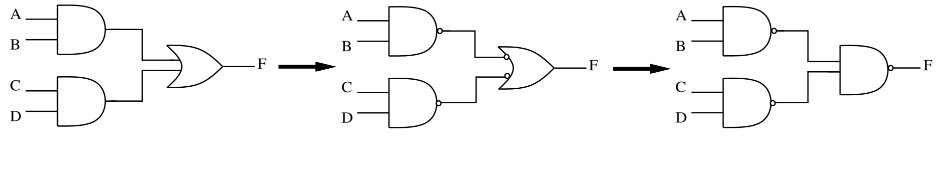
X	Y	F
0	0	1
0	1	1
1	0	1
1	1	0

- NAND (**<u>N</u>**ot **<u>AND</u>**): AND function followed by an inverter.
- NAND symbol: **AND-Invert** $\rightarrow$ DeMorgan's Law $\rightarrow$ **Invert-OR**.
- Set {NAND} is functionally complete.



NAND Implementations of Sum-of-Products

- NAND gates can be used to implement a sum-of-products form.
  - 1st level: AND-Invert symbols.
  - 2nd level: Invert-OR symbols.
- Exp: $F = AB + CD = \overline{\overline{\overline{AB} + \overline{CD}}} = \overline{\overline{AB} \cdot \overline{CD}}$

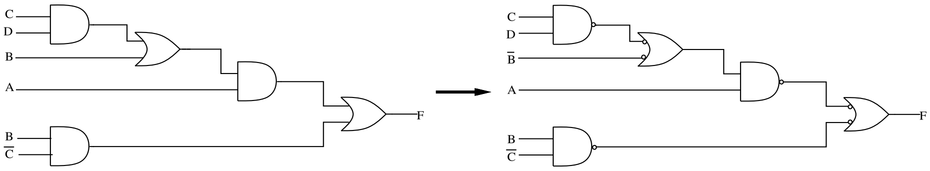


- For a term with only one literal, must use an inverter to implement.

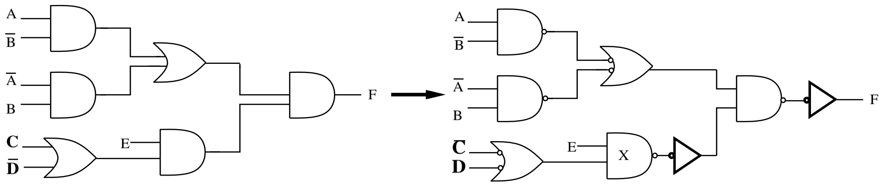
Multi-level NAND Implementations

- Convert a multilevel AND-OR diagram into an all-NAND diagram:
  1. Convert all AND gates to NAND gates with **AND-NOT** symbols.
  2. Convert all OR gates to NAND gates with **NOT-OR** symbols.
  3. For every bubble that does not come with another bubble along the same line, **insert a NOT gate** or **complement the input literal**.

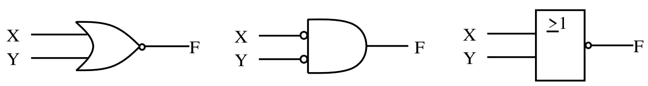
- Exp: $F = A(CD + B) + B\bar{C}$



- Exp: $G = (A\bar{B} + \bar{A}B)(E(C + \bar{D}))$



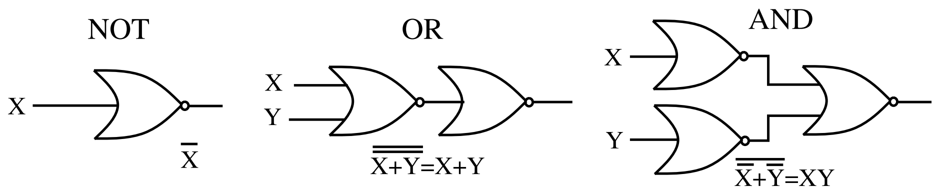
NOR Gates



$F = \overline{X + Y}$

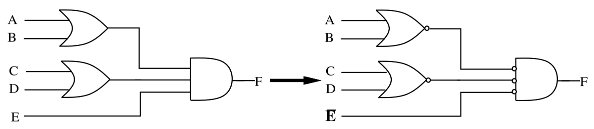
X	Y	F
0	0	1
0	1	0
1	0	0
1	1	0

- NOR (<u>**N**</u>ot <u>**OR**</u>): OR function followed by an inverter.
- NOR symbol: **OR-Invert** → DeMorgan's Law → **Invert-AND**.
- Set {NOR} is functionally complete.

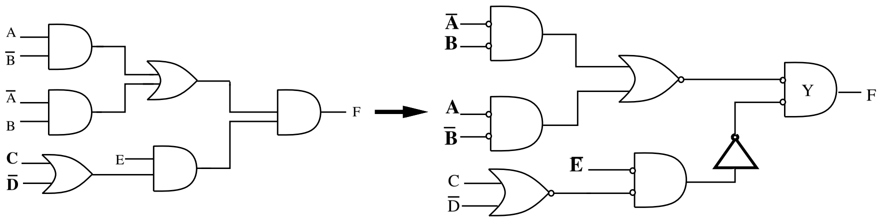


NOR Implementations

- NOR gates can be used to implement a product-of-sums form.
  - Exp: $F = (A+B)(C+D)E$

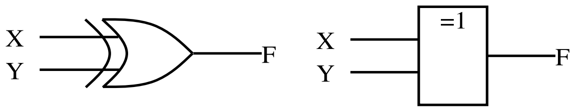


- Convert a multilevel AND-OR diagram into an all-NOR diagram:
  1. Convert all AND gates to NOR gates with **NOT-AND** symbols.
  2. Convert all OR gates to NOR gates with **OR-NOT** symbols.
  3. For every bubble that does not come with another bubble along the same line, **insert a NOT gate** or **complement the input literal**.
- Exp: $G = (A\bar{B} + \bar{A}B)(E(C+\bar{D}))$



XOR and XNOR Gates

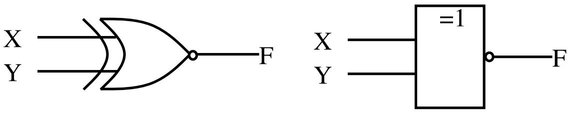
- **Exclusive-OR (XOR):** $X \oplus Y = X\bar{Y} + \bar{X}Y$.



$F = X\bar{Y} + \bar{X}Y$
$= X \oplus Y$

X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0

- Uses for the XOR gates: adders/subtracters, parity generators/checkers, pseudo-random number generators, etc.
- **Exclusive-NOR (XNOR)** (a.k.a. **equivalence):** $X \equiv Y = \overline{X \oplus Y} = XY + \bar{X}\bar{Y}$.
- **Questions:** Are XOR or XNOR Gates alone functionally complete?



$F = XY + \bar{X}\bar{Y}$
$= \overline{X \oplus Y}$

X	Y	F
0	0	1
0	1	0
1	0	0
1	1	1

XOR and XNOR Identities

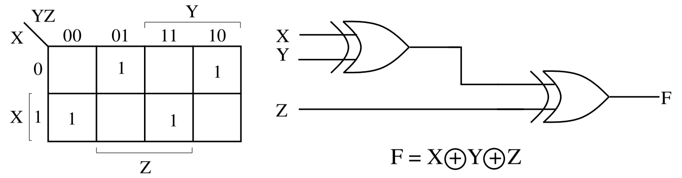
$X \oplus 0 = X$	$X \oplus 1 = \bar{X}$
$X \oplus X = 0$	$X \oplus \bar{X} = 1$
$X \oplus \bar{Y} = \overline{X \oplus Y}$	$\bar{X} \oplus Y = \overline{X \oplus Y}$
$X \oplus Y = Y \oplus X$	$(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z) = X \oplus Y \oplus Z$

- If $XY = 0$, then $X \oplus Y = X + Y$.
  - $\forall p, q, m_p m_q = 0 \Longrightarrow m_i \oplus m_j \oplus m_k = m_i + m_j + m_k$.
- Exp: Simplify $F = A\bar{B} \oplus AC \oplus BC$

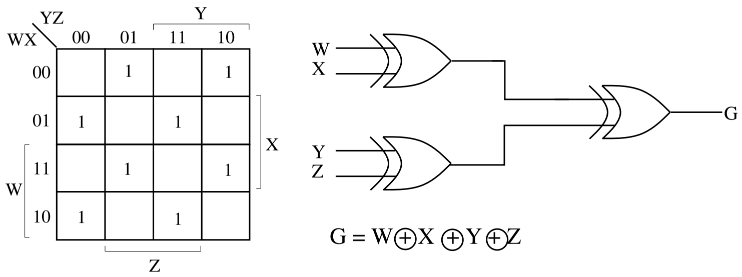
$$\begin{aligned} F &= (A\bar{B}C + A\bar{B}\bar{C}) \oplus (A\bar{B}C + ABC) \oplus (\bar{A}BC + ABC) \\ &= A\bar{B}C \oplus A\bar{B}\bar{C} \oplus A\bar{B}C \oplus ABC \oplus \bar{A}BC \oplus ABC \\ &= A\bar{B}\bar{C} \oplus \bar{A}BC \\ &= A\bar{B}\bar{C} + \bar{A}BC \end{aligned}$$

Odd Function

- **Odd function:** The function value is 1 iff the number of ones in the input variable set is odd.
- $X \oplus Y \oplus Z = \underbrace{\bar{X}\bar{Y}Z}_{001} + \underbrace{X\bar{Y}\bar{Z}}_{100} + \underbrace{\bar{X}Y\bar{Z}}_{010} + \underbrace{XYZ}_{111}$
- **Even function:** Replacing the output gate with an XNOR gate.



F = X⊕Y⊕Z

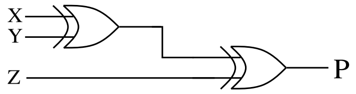


G = W⊕X ⊕Y⊕Z

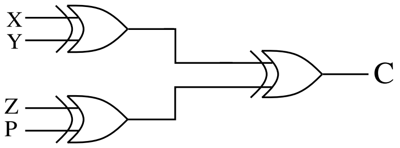
Parity Generators/Checkers

Three-Bit Message			Parity Bit
X	Y	Z	P
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

- We can use a parity tree to generate parity.
  - A 3-bit even parity generator.



- We can use a parity tree to check the code word for correct parity.
  - A 4-bit tree: $C = 0$ if the even parity is correct; $C = 1$ if an error occurs.



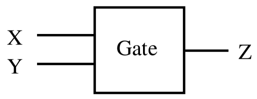
Logic Levels

- Two-valued Boolean logic are assigned to two voltage levels, **high** (H) and **low** (L).
- **Positive logic system:** 0 is assigned to L, and 1 to H.
- **Negative logic system:** 0 is assigned to H, and 1 to L.
- Modern IC designs predominately use positive logic.

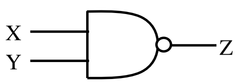
X	Y	Z
L	L	H
L	H	H
H	L	H
H	H	L

X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

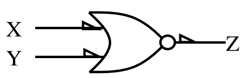
X	Y	Z
1	1	0
1	0	0
0	1	0
0	0	1



Gate behavior



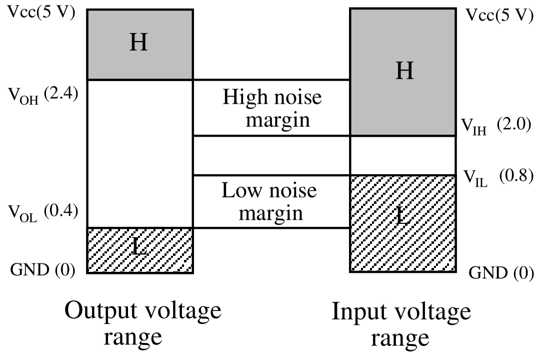
Positive logic NAND



Negative logic NOR

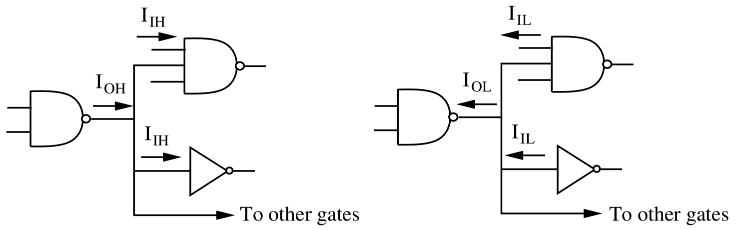
Noise Margin

- **Noise:** All undesirable voltage variations that are superimposed on normal operating voltage levels.
- **Noise Margin:** The maximum noise voltage level that a logic signal can tolerate without errors.
- High-level noise margin: $V_{OH} - V_{IH}$.
- Low-level noise margin: $V_{IL} - V_{OL}$.



Fan-in/-out

- **Fan-in:** Number of inputs available on a gate.
- **Fan-out (standard load):** Number of inputs that each output can drive.
- Fan-out = $\min\{\frac{I_{OH}}{I_{IH}}, \frac{I_{OL}}{I_{IL}}\}$.
- Exp: $I_{OH} = 600\mu A, I_{IH} = 40\mu A, I_{OL} = 16\mu A, I_{IL} = 1.6\mu A \Longrightarrow \min\{\frac{600}{40}, \frac{16}{1.6}\} = 10$.



Gate as a current source

Gate as a current sink

Power Dissipation

- **Power supply (V_{CC}):** Voltage required to allow a circuit to operate.
- **Power dissipation:** The amount of power a circuit consumes.
- Each gate can be in 3 states: **high**, **transition**, **low** states.
- Respective currents: I_{CCH} (high); I_{CCT} (transition); I_{CCL} (low).
- In modern technologies (CMOS, etc): $I_{CCT} \gg I_{CCH}$ and $I_{CCT} \gg I_{CCL} \Longrightarrow$ Average power dissipation:

$$P_{avg} = V_{CC} \times I_{CCT}.$$

- In older logic families (TTL, etc): $I_{CCH} \gg I_{CCT}$ and $I_{CCL} \gg I_{CCT} \Longrightarrow$ Average power dissipation:

$$P_{avg} = V_{CC} \times (I_{CCH} + I_{CCL})/2.$$

- The power dissipation is an increasingly important issue in modern IC designs.
  - The power dissipation of a circuit defines the battery life.
  - It is proportional to the heat generated by a chip.

Propagation Delay

- **Rise time:** The delay for a signal to switch from 10% to 90% of its normal value.
- **Fall time:** The delay for a signal to switch from 90% to 10% of its normal value.
- ***H*-to-*L* propagation delay (t_{PHL}):** The time for the **output** signal to reach 50% of its normal value on the H-to-L transition after the input signal reached 50% of its normal signal. t_{PLH} is defined similarly.
- In modern technologies, the **propagation delay** t_{pd} is measured by

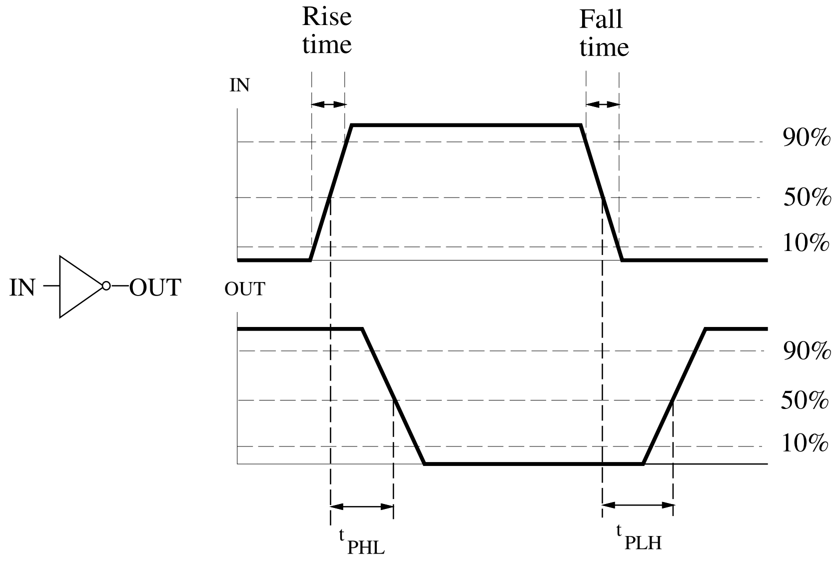
$$t_{pd} = \frac{(t_{PHL} + t_{PLH})}{2}.$$

- In older technologies, the **propagation delay** t_{pd} is measured by

$$t_{pd} = \max\{t_{PHL}, t_{PLH}\}.$$

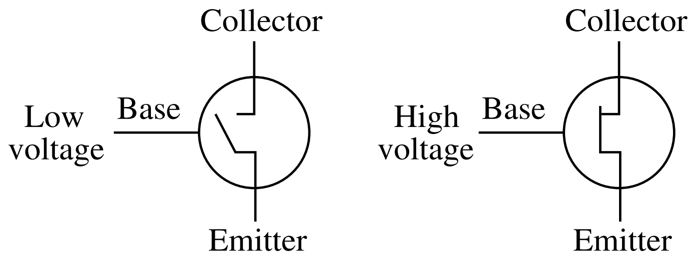
Propagation Delay for an Inverter

- High-to-low (t_{PHL}) and low-to-high (t_{PLH}) propagation delays are measured on **output transitions**.



Digital Logic Families: Bipolar Transistors

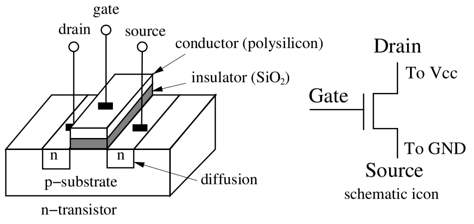
- Gates could be constructed using two transitor technologies: **bipolar** and **metal-oxide-semiconductor (MOS)**; based on physical characteristics, they are organized into **families**.
- **Bipolar technologies:** using resistors, diodes, transistors.
  - RTL (Resistor-Transistor Logic) and DTL (Diode-Transistor Logic): used in 1960s.
  - TTL (Transistor-Trnasistor Logic): dominated digital designs in 1970s and 1980s.
  - ECL (Emitter-Coupled Logic): mainly for high-speed applications, e.g., supercomputers.



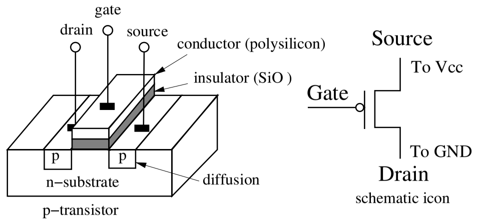
Bipolar transistor model

Digital Logic Families: MOS Transistors

- **MOS technologies:** a layer of mental above a layer of semiconductor with a layer of silicon dioxide serving as an insulation between them.
  - MOS, CMOS (Complementary MOS), BiCMOS (Bipolar CMOS) dominate modern digital designs.
  - CMOS dominates nMOS and pMOS, due to CMOS's lower power dissipation, high fan-out, simple process, and high regularity.



The nMOS switch passes "0" well.



The pMOS switch passes "1" well.

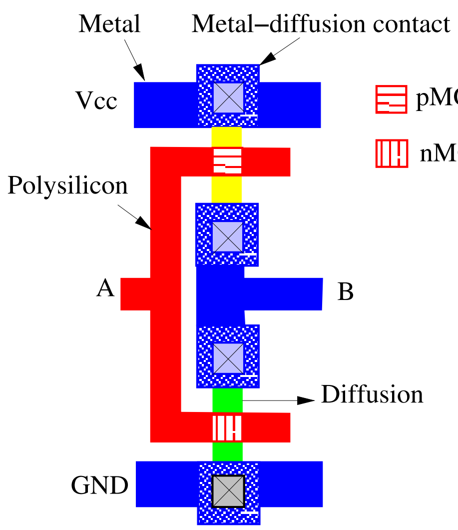
Characteristics of 2-input NAND

Logic family	# of resistors	# of diodes	# of transistors	Noise margin (V)	Fan out	Power dissipation (mW)	Delay (ns)
RTL	3	-	2	0.4	5	20	25
DTL	4	3	2	1.0	8	12	30
TTL	4	1	5	0.4	10	10	9
Schottky TTL	5	2	7	0.4	10	19	3
ECL	8	2	6	0.15	25	25	2
CMOS	-	-	4	0.5	30–50	0.1–0.3	1

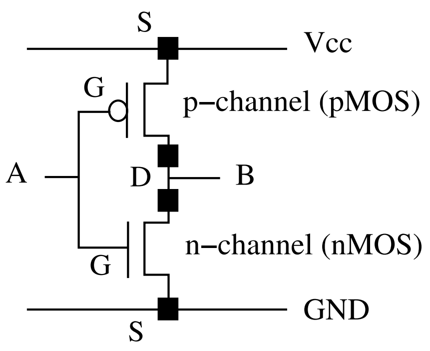
Level of Integration

- The number of gates put in a single chip differentiates integrated circuits (ICs).
- **Small-scale integration (SSI):** # of gates ≤ 10.
  - Simple logic functions, basic storage elements, etc.
- **Medium-scale integration (MSI):** 10–100 gates.
  - 4-bit adders, 8-to-1 multiplexers, counters, etc.
- **Large-scale integration (LSI):** 100 to a few 1000 gates.
  - Programmable logic array (PLA), PROM, small processors, etc.
- **Very large-scale integration (VLSI):** several thousand to millions of gates.
  - Processors, large memory array, etc.
- **Ultra large-scale integration (ULSI):** # of gates $>$ 10 million.
  - Advanced processors, system chips, etc.

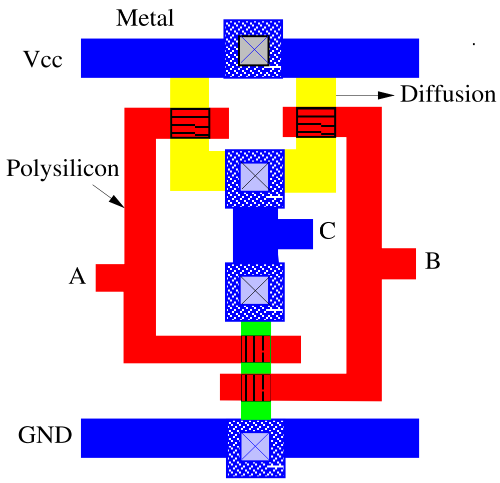
CMOS Inverter Gate



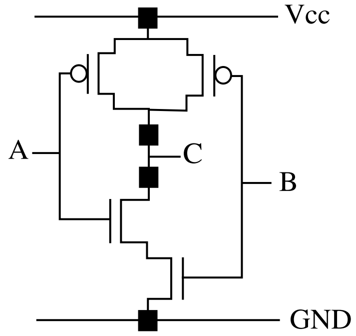
A	B
1	0
0	1



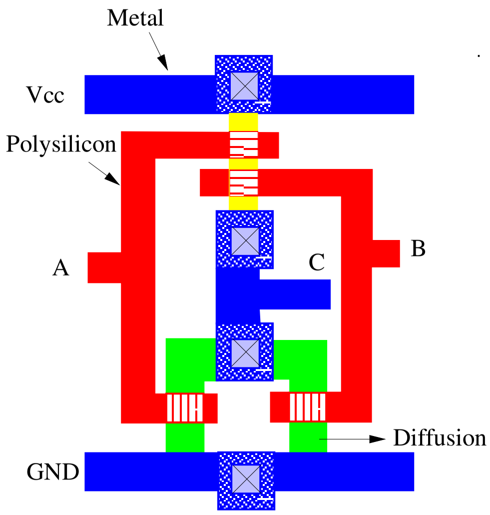
CMOS NAND Gate



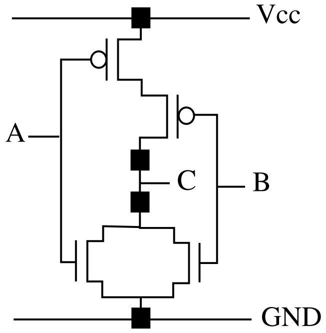
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0



CMOS NOR Gate

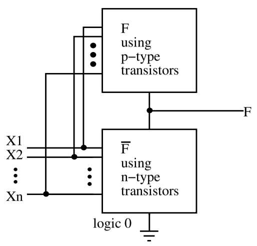


A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

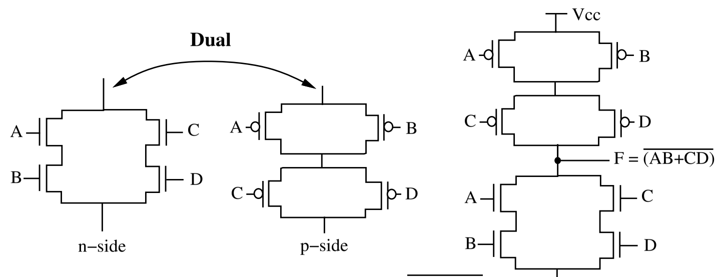


Construction of Compound Gates

- Example: $F = \overline{A \cdot B + C \cdot D}$.
- Step 1: Invert F to derive n-network ($\overline{F} = A \cdot B + C \cdot D$).
- Step 2: Make connections of transistors:
  - AND $\Longleftrightarrow$ Series connection; OR $\Longleftrightarrow$ Parallel connection.
- Step 3: Expand F to derive p-network ($F = \overline{AB + CD} = \overline{AB} \cdot \overline{CD} = (\overline{A} + \overline{B}) \cdot (\overline{C} + \overline{D})$); each input is inverted.
- Step 4: Make connections of transistors (same as Step 2).
- Step 5: Connect the n-network to GND (typically, 0V) and the p-network to V_{CC} (5V, 3.3V, or 2.5V).

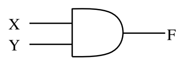
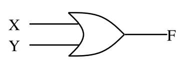
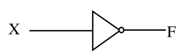
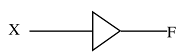
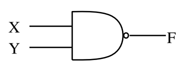
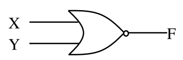
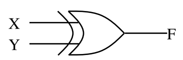
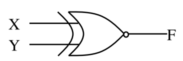


General Structure

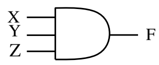
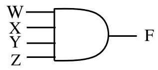
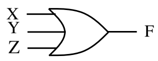
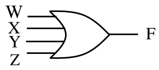
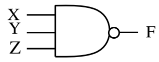
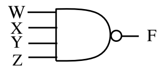
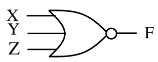
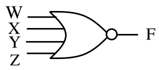


CMOS compound gate for F = $\overline{(AB + CD)}$

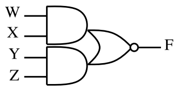
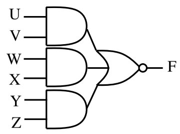
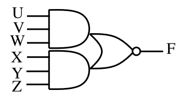
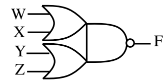
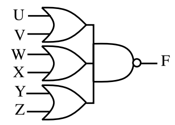
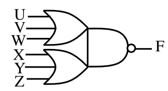
Basic CMOS Logic Library

Name	Distinctive shape	Algebraic equation	Cost (# of transistors)	Gate delay (ns)
AND		F=XY	**6**	**2.4**
OR		F=X+Y	**6**	**2.4**
NOT (Inverter)		$F=\overline{X}$	**2**	**1.0**
Buffer (Driver)		F=X	**4**	**2.0**
NAND		$F=\overline{XY}$	**4**	**1.4**
NOR		$F=\overline{X+Y}$	**4**	**1.4**
Exclusive–OR (XOR)		$F=X\overline{Y}+\overline{X}Y$ $=X\oplus Y$	**14**	**4.2**
Exclusive–NOR (XNOR)		$F=XY+\overline{X}\,\overline{Y}$ $=\overline{X\oplus Y}$	**12**	**3.2**

Multiple-input CMOS Logic Library

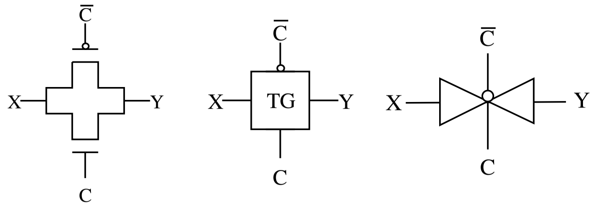
Name	Graphic symbol	Algebraic equation	Cost (# of transistors)	Gate delay (ns)
3–input AND		F = XYZ	**8**	**2.8**
4–input AND		F = WXYZ	**10**	**3.2**
3–input OR		F = X + Y + Z	**8**	**2.8**
4–input OR		F = W + X + Y + Z	**10**	**3.2**
3–input NAND		$F = \overline{XYZ}$	**8**	**1.8**
4–input NAND		$F = \overline{WXYZ}$	**10**	**2.2**
3–input NOR		$F = \overline{X + Y + Z}$	**8**	**1.8**
4–input NOR		$F = \overline{W + X + Y + Z}$	**10**	**2.2**

Multiple-operator CMOS Logic Library

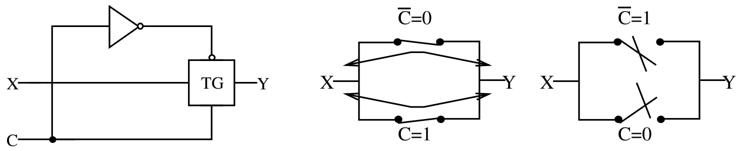
Name	Graphic symbol	Algebraic equation	Cost (# of transistors)	Gate delay (ns)
2–wide, 2–input AOI		$F = \overline{WX+YZ}$	**8**	**2.0**
3–wide, 2–input AOI		$F = \overline{UV + WX + YZ}$	**12**	**2.4**
2–wide, 3–input AOI		$F = \overline{UVW + XYZ}$	**12**	**2.2**
2–wide, 2–input OAI		$F = \overline{(W + X)(Y + Z)}$	**8**	**2.0**
3–wide, 2–input OAI		$F = \overline{(U + V)(W + X)\ (Y + Z)}$	**12**	**2.2**
2–wide, 3–input OAI		$F = \overline{(U + V + W)(X + Y + Z)}$	**12**	**2.4**

CMOS Transmission Gate (TG)

- A parallel n-channel and p-channel device driven by **opposite** control signals has two states:
  - **OFF (open)** when the drive signal is “0”
  - **ON (closed)** when the drive signal is “1”
- Current can flow either way through the TG.



Three types of notation.

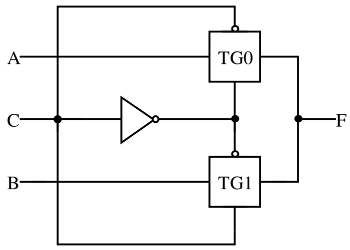


Control signals are mutually complemented.

C = 1: X and Y are connected;
C = 0: X and Y are disconnected.

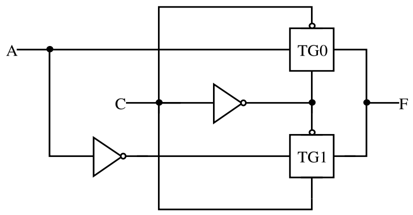
CMOS Logic with TGs

- Transmission gate-based multiplexer: ($C = 1 \Longrightarrow$ select B; $C = 0 \Longrightarrow$ select A)



F(A, B, C) = A$\overline{C}$ + BC

- Transmission gate-based XOR logic:



A	C	TG1	TG0	F
0	0	No path	Path	0
0	1	Path	No path	1
1	0	No path	Path	1
1	1	Path	No path	0

F(A, C) = A$\oplus$C