

Chapter 2

Boolean Algebra and Logic Gates

Boolean Algebra

- Boole
 - In 1854, George Boole introduced a systematic treatment of logic and developed for this purpose an algebraic system now called *Boolean algebra*.
- Huntington
 - In 1904, E.V. Huntington formulated the postulates for the formal definition Boolean algebra.
 - For the formal definition of Boolean Algebra, we shall employ *Huntington's First Set of Postulates*.
- Shannon
 - In 1938, C. E. Shannon introduced two-valued Boolean algebra called *Switch algebra*, in which he demonstrated that the properties of binary electrical circuits can be represented by this algebra.

Most Common Postulates for Algebra

- **Closure (封閉性)**
 - $x*y$ is also in S for any x,y in S ($*$ operator)
- **Associative law (結合律)**
 - $(x*y)*z = x*(y*z)$
- **Commutative law (交換律)**
 - $x*y = y*x$
- **Identity element (單位元素)**
 - $e*x=x*e =x$, $x + 0 = x$
- **Inverse**
 - $a + (-a) = 0$
- **Distributive law (分配律)**
 - $x*(y + z) = (x*y) + (x*z)$

Two-valued Boolean Algebra

- $B = \{0,1\}$
- The rules of operations

AND			OR			NOT	
x	y	$x \cdot y$	x	y	$x + y$	x	x'
0	0	0	0	0	0	0	1
0	1	0	0	1	1	1	0
1	0	0	1	0	1		
1	1	1	1	1	1		

- Closure 成立 (result is "0" or "1")
- Identity elements

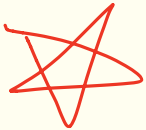
$$(1) +: 0 \quad 0+0=0 \quad 1+0=1 \Rightarrow x+0=x \text{ (0對+)}$$

$$(2) \cdot: 1 \quad 0 \cdot 1=0 \quad 1 \cdot 1=1 \Rightarrow x \cdot 1=x \text{ (1對\cdot)}$$

Distributive Laws

- Distributive laws 成立

$$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$$



Truth Table 真值表 使用真值表証明

x	y	z	$y + z$	$x \cdot (y + z)$	$x \cdot y$	$x \cdot z$	$(x \cdot y) + (x \cdot z)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

Table 2.1
Postulates and Theorems of Boolean Algebra

Postulate 2	(a)	$x + 0 = x$	identity element	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$		(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$		(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$		(b)	$x \cdot 0 = 0$
Theorem 3, involution		$(x')' = x$			
Postulate 3, commutative	(a)	$x + y = y + x$		(b)	$xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$		(b)	$x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$		(b)	$x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$		(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$		(b)	$x(x + y) = x$

$(xy)' \neq x'y'$

- By means of truth table to prove

x	y	xy	x + xy
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

DeMorgan's Theorems

- DeMorgan's Theorems

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

x	y	$x+y$	$(x+y)'$	x'	y'	$x' y'$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

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

x	y	$x \cdot y$	$(x \cdot y)'$	x'	y'	$x' + y'$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Operator Precedence

- **Operator Precedence** 運算子優先順序

- **parentheses** 括弧

- **NOT**

- **AND**

- **OR**

- **examples**

- $x y' + z \rightarrow$ is $x \cdot y'$ not $(xy)'$

- $(x y + z)'$ $\begin{matrix} = (xy)'z' \\ = (x' + y')z' \end{matrix}$

- **The operations are implemented with logic gates**

真值表





Name	Graphic symbol	Algebraic function	Truth table															
AND		$F = xy$	<table><tr><th>x</th><th>y</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>	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = x + y$	<table><tr><th>x</th><th>y</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>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Inverter NOT		$F = x'$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	F	0	1	1	0									
x	F																	
0	1																	
1	0																	
Buffer		$F = x$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	x	F	0	0	1	1									
x	F																	
0	0																	
1	1																	

Figure 2.5 Digital logic gates (continued)


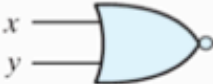
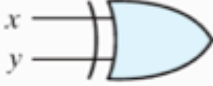
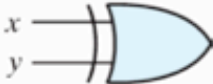
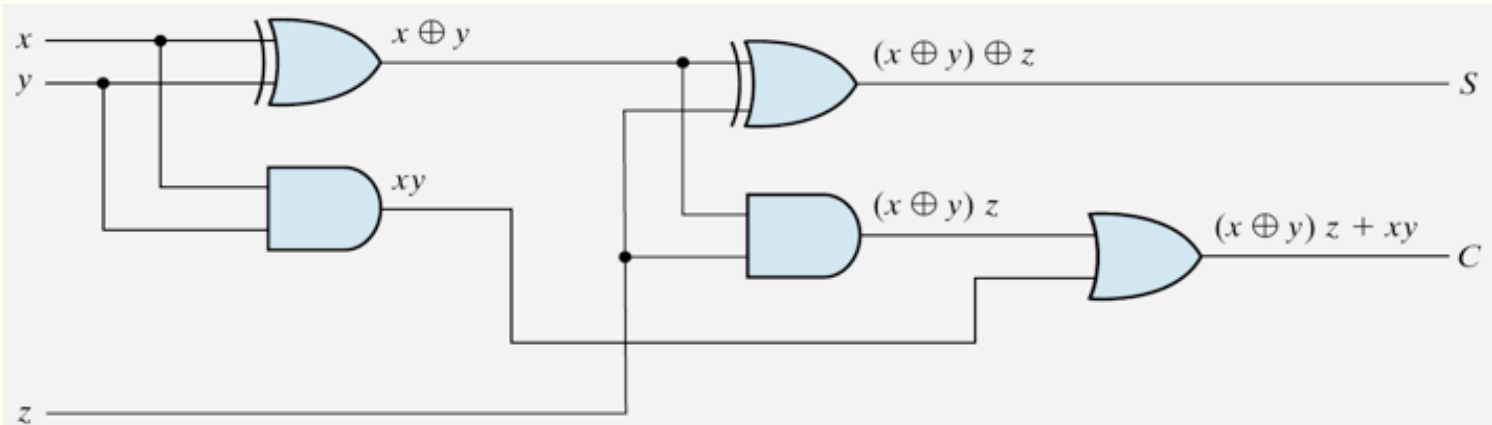
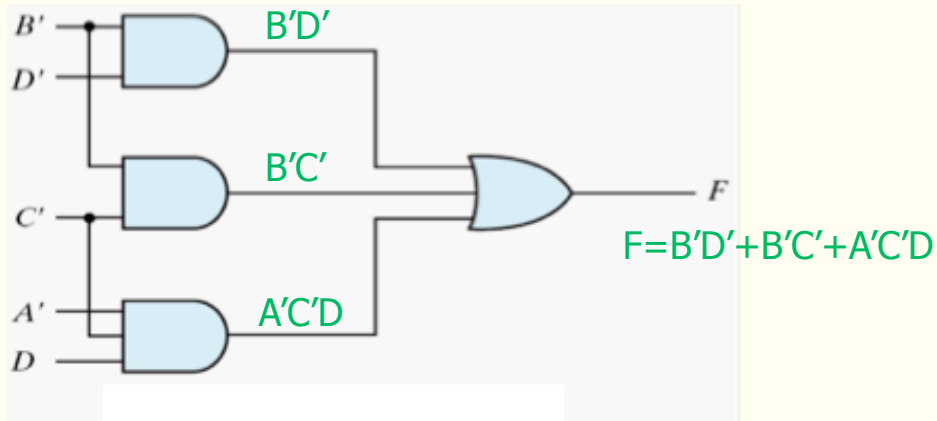
NAND		$F = (xy)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = (x + y)'$	<table><tr><th>x</th><th>y</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>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
Exclusive-OR (XOR) 互斥或閘		$F = xy' + x'y$ $= x \oplus y$	<table><tr><th>x</th><th>y</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>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	<table><tr><th>x</th><th>y</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>1</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

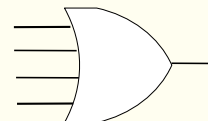
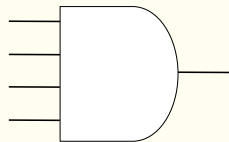
Figure 2.5 Digital logic gates

A Simple Circuit



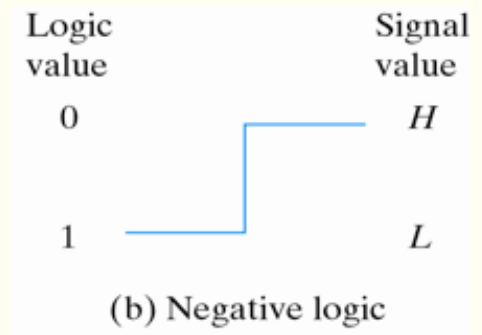
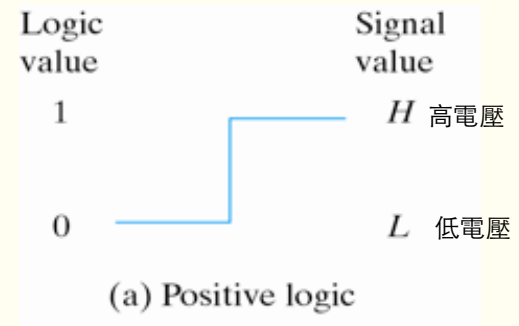
Extension to Multiple Inputs

- Extension to multiple inputs
 - A gate can be extended to multiple inputs
 - if its binary operation is commutative and associative
 - AND and OR are commutative and associative
 - $(x+y)+z = x+(y+z) = x+y+z$
 - $(x \ y)z = x(y \ z) = x \ y \ z$



Positive and Negative Logic

- Positive and Negative Logic
 - two signal values \Leftrightarrow two logic values
 - positive logic: $H=1$; $L=0$ 1會動
 - negative logic: $H=0$; $L=1$ 0會動
- the positive logic is used in this book



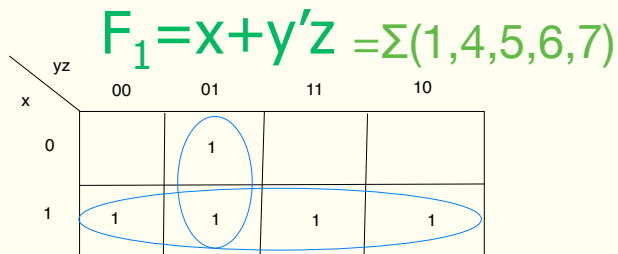
- ➔ A device is active high or active low
- ➔ Some devices are active high and some are active low.

Boolean Functions

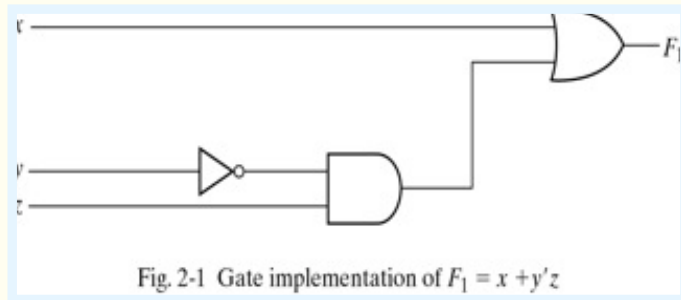
- 3 kinds of representations for Boolean functions
 - Boolean Algebra
 - binary variables
 - constants 0,1
 - logic operation symbols $+$, \bullet , $'$
 - Truth table
 - Circuit diagram

Three representations

1. Boolean Algebra



3. Circuit Diagram



2. Truth Table

真值表

n input variables $\rightarrow 2^n$ combinations

Inputs $F_1 = x'y'z + xy'z' + xy'z + xyz' + xyz$

x	y	z	y'	y'z	F_1
0	0	0	1	0	0
0	0	1	1	1	1 m_1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	0	1 m_4
1	0	1	1	1	1 m_5
1	1	0	0	0	1 m_6
1	1	1	0	0	1 m_7

Canonical and Standard Forms

- Minterms and Maxterms

- A ^{全及項} **minterm**: an **AND** term consists of all literals in their normal form or in their complement form
- For example, two binary variables x and y
 - $xy, xy', x'y, x'y'$
- It is also called a standard product
- n variables can be combined to form 2^n minterms
- A ^{全或項} **maxterm**: an **OR** term Maxterm為minterm的補數
- It is also called a standard sum
- 2^n maxterms

Minterms and Maxterms

- each maxterm is the complement of its corresponding minterm, and vice versa

Table 2.3

Minterms and Maxterms for Three Binary Variables

			Minterms		Maxterms	
<i>x</i>	<i>y</i>	<i>z</i>	Term	Designation	Term	Designation
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

Canonical Forms

- Canonical Forms 正規型式
 - Sum (ORing) of minterms
 - $F = x'y'z + xyz + xyz'$ in canonical form of 3 variables
 - $F = x'y'z + xy$ not a canonical form
 - Product (ANDing) of maxterms
 - $F = (x'+y'+z)(x+y+z)(x+y+z')$ in canonical form of 3 variables
 - $F = (x'+y'+z)(x+y)$ not a canonical form
- Properties of Boolean Algebra 用minterms推回Boolean function
 - A Boolean function can be expressed as a sum of minterms
 - A Boolean function can be expressed as a product of maxterms

Sum of Minterms (全及項的和)

Table 2.4
Functions of Three Variables

	x	y	z		Function f_1	Function f_2
m_0	0	0	0		0	0
m_1	0	0	1	←	1	0
m_2	0	1	0		0	0
m_3	0	1	1		0	1
m_4	1	0	0	←	1	0
m_5	1	0	1		0	1
m_6	1	1	0		0	1
m_7	1	1	1	←	1	1

- An Boolean function can be expressed by a Sum of minterms (全及項的和)
 - $f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7 = \Sigma(1,4,7)$
 - $f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$

Product of Maxterms (全或項的積)

$$f_1 = x'y'z + xy'z' + xyz$$

$$= m_1 + m_4 + m_7$$

求 Maxterm : 先算出 f_1 (minterm) ,
再求 f_1' , 即為 minterm 相反項之積

Table 2.4
Functions of Three Variables

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- An Boolean function can also be expressed by a Product of maxterms (全或項的積)

$$\square f_1' = x'y'z' + x'yz' + x'yz + xy'z + xyz' \text{ (所有 } f_1=0 \text{ 的全及項的和)}$$

$$= m_0 + m_2 + m_3 + m_5 + m_6$$

$$\rightarrow f_1 = (x'y'z')'(x'yz')'(x'yz)'(xy'z)'(xyz)'$$

$$= (x+y+z)(x+y'+z)(x+y'+z')(x'+y+z')(x'+y'+z) = M_0 M_2 M_3 M_5 M_6$$

Conversion between Canonical Forms

■ $F(A,B,C) = \Sigma (1, 4, 5, 6, 7)$

→ $F'(A,B,C) = \Sigma(0, 2, 3) = m_0 + m_2 + m_3$

→ $F(A,B,C) = (m_0 + m_2 + m_3)' = m'_0 m'_2 m'_3$
 $= M_0 M_2 M_3 = \Pi(0, 2, 3)$

→ $F(A,B,C) = \Sigma (1, 4, 5, 6, 7) = \Pi(0, 2, 3)$

$F(A,B,C) = \Sigma(1, 3, 6, 7) = \Pi(0, 2, 4, 5)$

Table 2.5

Truth Table for $F = A + B'C$

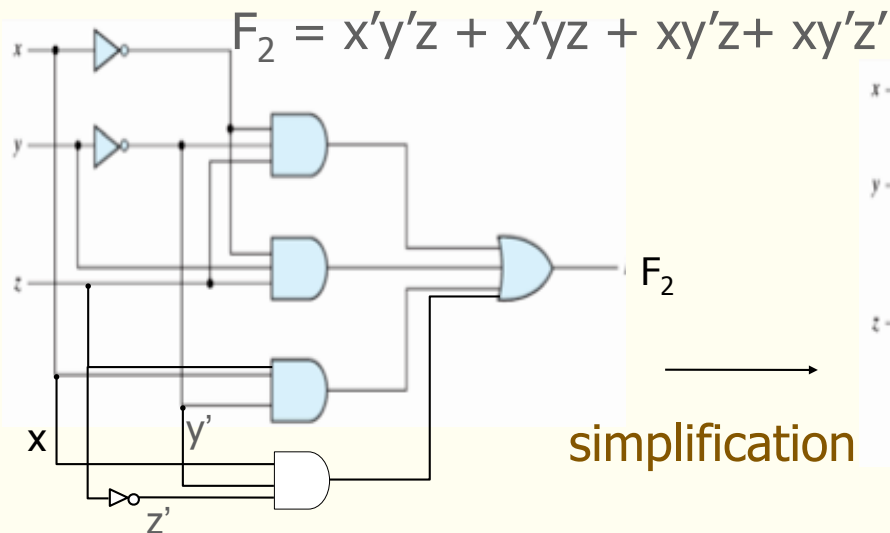
A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Standard Forms

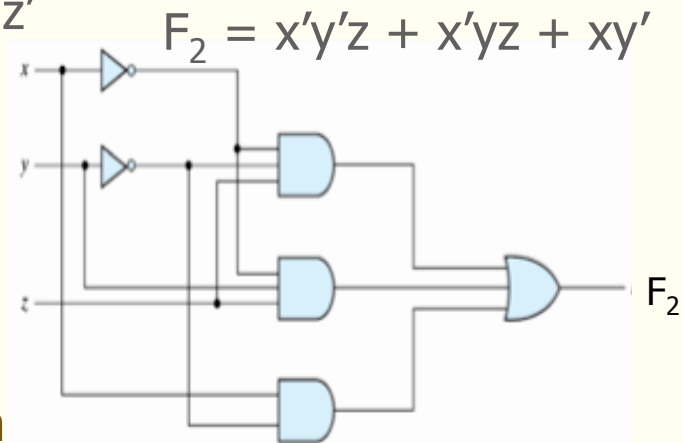
- Disadvantage of the canonical forms 不夠精簡
 - It requires each term to contain all variables (not the least number of literals)
sum of minterms or product of maxterms
 - are seldom used
- Another is Standard Forms
 1. sum of products
 2. product of sums
- Characteristics of Standard Forms
 - with the *least number of literals*

Three Equivalent Circuits

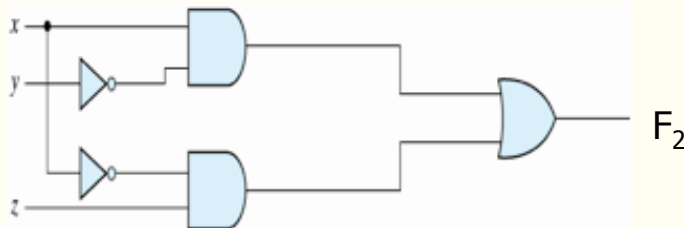
An Boolean function can be expressed by a Sum of minterms



simplification



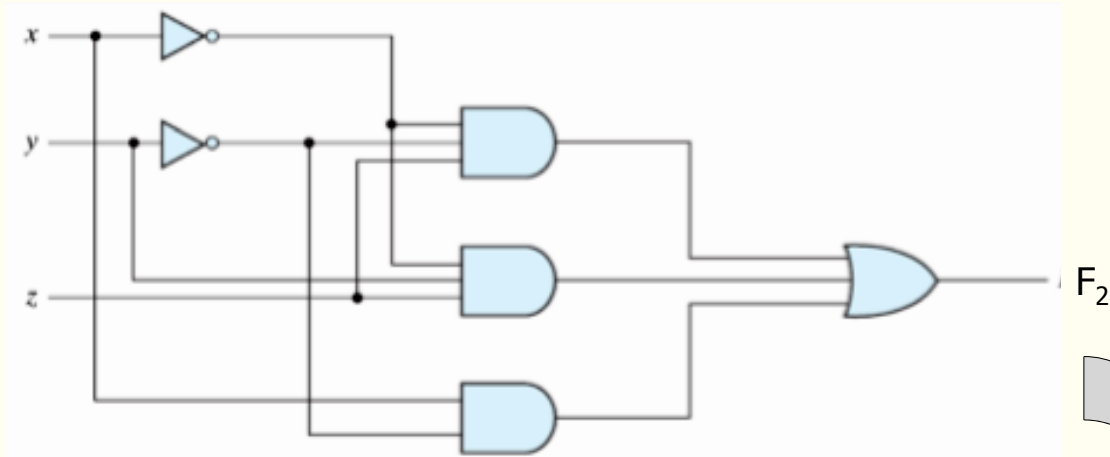
$$F_2 = x'y'z + x'yz + xy' = x'z(y' + y) + xy' = x'z + xy'$$



They are equivalent

simplification

Simplification with Boolean Algebra (1/2)



$$F_2 = x'y'z + x'yz + xy'z = x'z(y' + y) + xy'z = x'z + xy'z$$



They are
equivalent

Simplification with Boolean Algebra (2/2)

DeMorgan's theorem

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

$$(ABC)' = A'+B'+C'$$

Generalizations

$$(A+B+C+ \dots +F)' = A'B'C' \dots F'$$

$$(ABC \dots F)' = A'+ B'+C'+ \dots +F'$$

布林代式精簡化

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

$$[x(y'z'+yz)]' = x' + (y'z'+yz)' = x' +$$

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

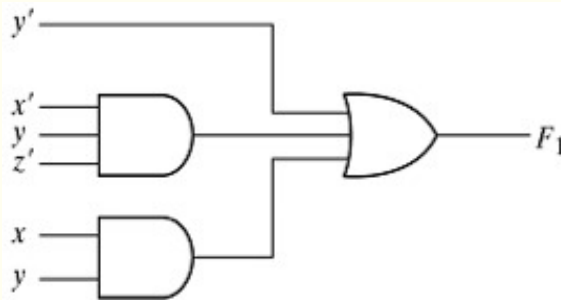
$$+ yz' + zy'$$

We skip the discussion of simplification with Boolean algebra here.
In fact, software tools can do a better job than human.

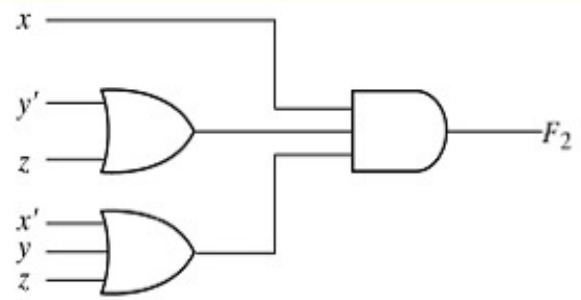
Standard Forms - Two-level Gating

Ex. $F_1 = y' + xy + x'yz'$
(sum of products)

$F_2 = x(y' + z)(x' + y + z')$
(product of sums)



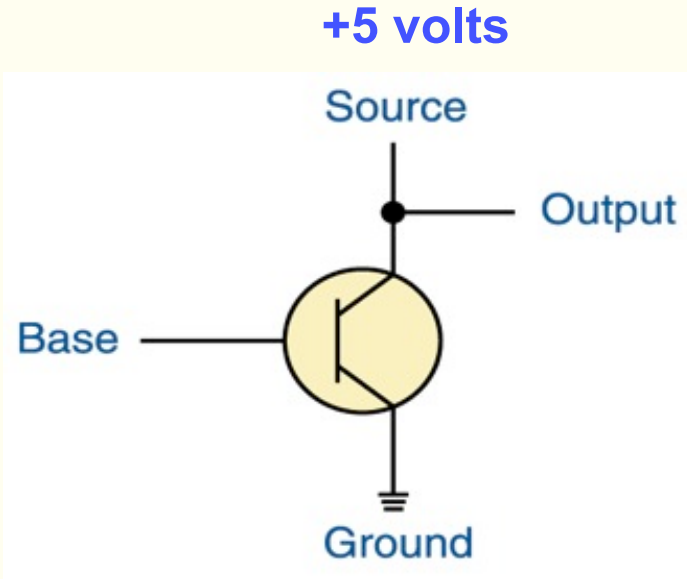
(a) Sum of Products



(b) Product of Sums

Fig. 2-3 Two-level implementation

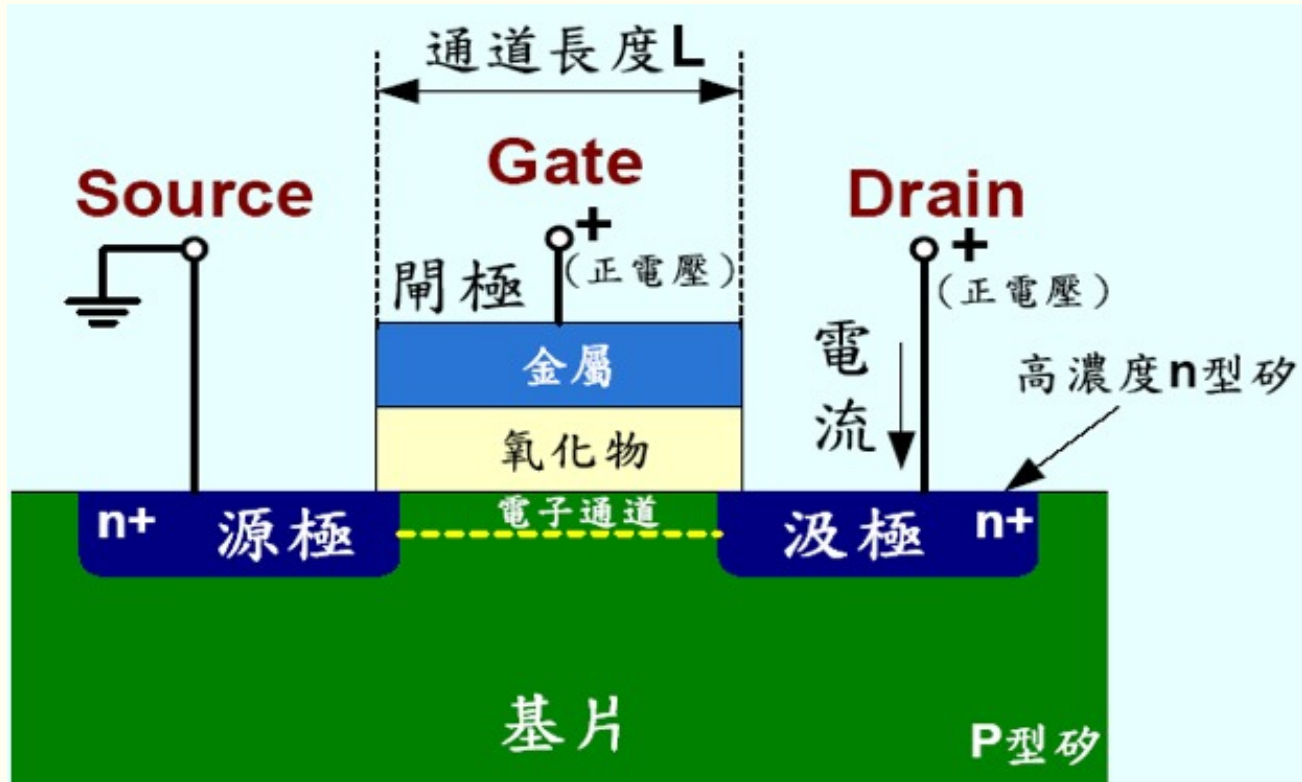
Transistor (電晶體)



- A transistor has three terminals
 - A source (feed with 5 volts)
 - A base
 - An emitter, typically connected to a ground wire
- If the base signal is high (close to +5 volts), the source signal is grounded and the output signal is low (0). If the base signal is low (close to 0 volts), the source signal stays high and the output signal is high (1)

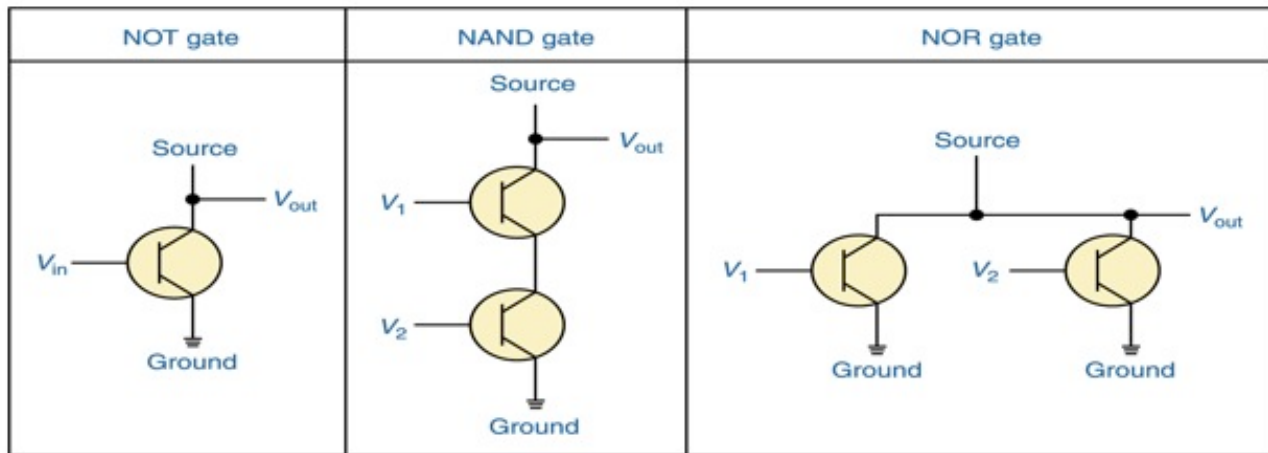
N-channel MOS Transistor

Transistor (電晶體) – **Semiconductor** (半導體)



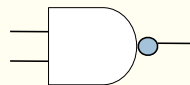
Constructing Gates (semiconductor)

- It turns out that, because the way a transistor works, the easiest gates to create are the NOT, NAND, and NOR gates

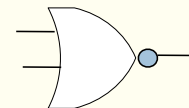


V_{in}	V_{out}
0	1
1	0

V_1	V_2	V_{out}
0	0	1
0	1	1
1	0	1
1	1	0



V_1	V_2	V_{out}
0	0	1
0	1	0
1	0	0
1	1	0



Circuits

- **Gate (1 gate \approx 2~14 transistors)**

A combination of interacting transistors

- **Circuit**

A combination of interacting gates designed to accomplish a specific logical function → **Integrated Circuit (IC)**

- **System → PCB (printed circuit board)**

- As with gates, we can describe the operations of entire circuits using three notations

- Boolean expressions
- logic diagrams
- truth tables



Integrated Circuits (IC)

- Chip
 - A silicon semiconductor crystal that contains the electronic components for constructing digital gates.
- Levels of Integration - Categories of ICs by their complexity
 - SSI - Small-scale Integration devices (~10 gates)
 - MSI - Medium-scale Integration devices (~1000 gates)
 - LSI - Large-scale Integration devices (>1000 gates)
 - VLSI - Very Large-scale integration
 - hundreds of thousands of gates
 - millions of transistors
 - SoC (system on a chip)



Digital Logic Families (by circuit technology)

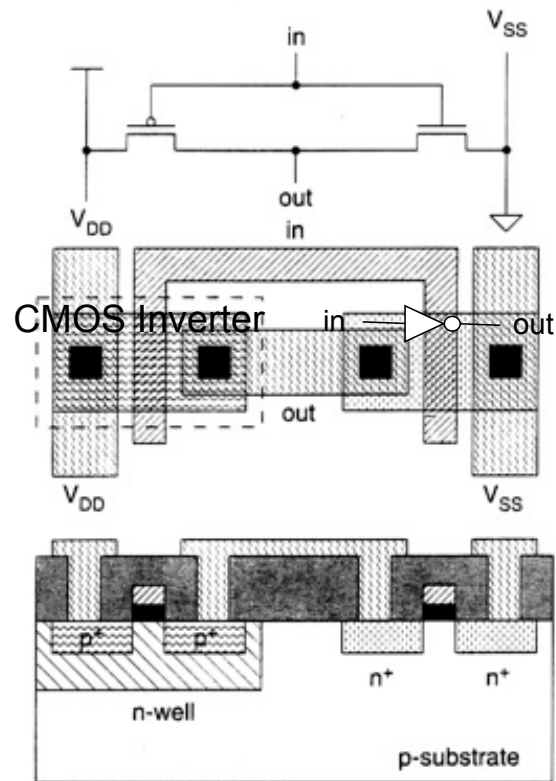
- Digital Logic Families
 - Each family has its own basic electronic circuit
 - named by its electronic components employed in the construction of the basic circuit
- Most popular digital logic families
 - TTL : Transistor-transistor logic (standard)
 - ECL : emitter-coupled logic (high-speed)
 - MOS : metal-oxide semiconductor (high component density)
 - CMOS : complementary metal-oxide semiconductor (low power consumption)



most
popular

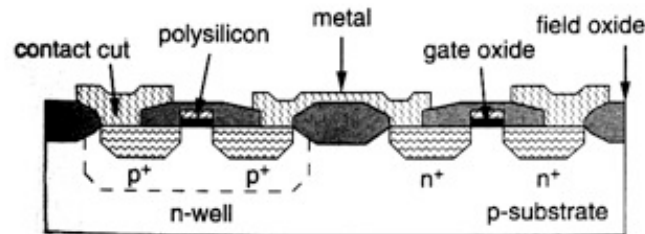
IC Design (with CMOS)

One npn transistor
and one pnp transistor
are used to construct
one inverter.



done by
chip designer

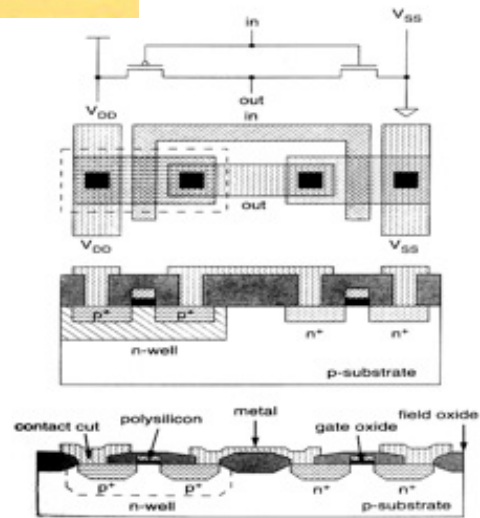
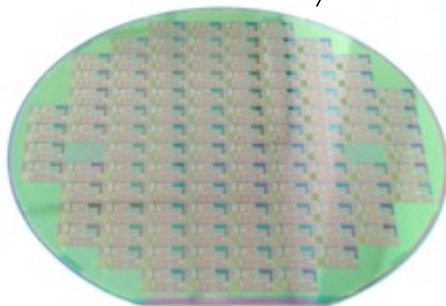
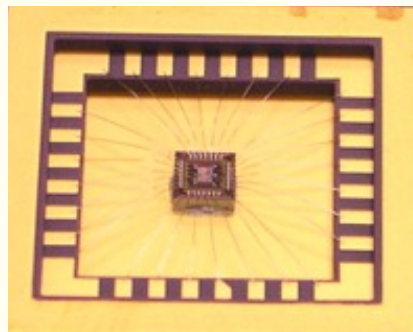
masking



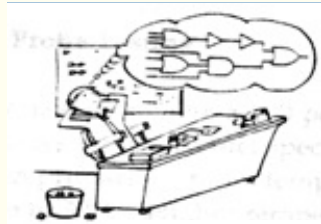
done by
TSMC, UMC

Packing, Testing

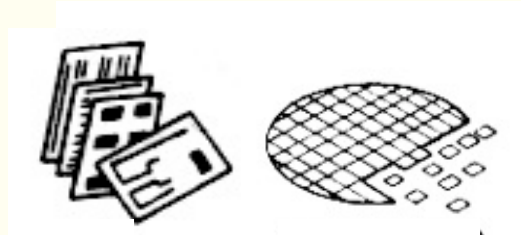
Chip/Circuit Everywhere!



IC Industry in Taiwan

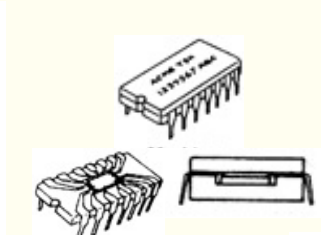


邏輯設計



光罩設計

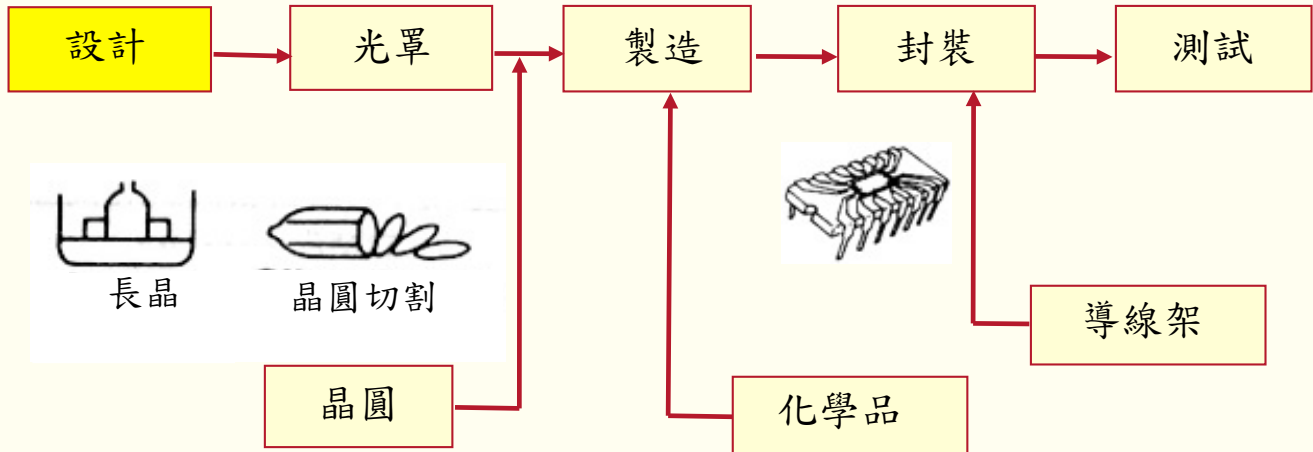
晶粒測試及切割



封裝



成品測試



Parameters for Digital Logic Families (1/2)

Parameters to be evaluated and compared among digital logic families

- **Fan-out** 可推動不影響正常行為的負載數
 - # of standard loads that the output of a typical gate can drive without impairing its normal operation =» 會影響電路設計
 - Standard load: the amount of current needed by an input of another gate
- **Fan-in** the number of inputs available in a gate
- **Power dissipation (power consumption)**
 - ➔ battery life and cooling system

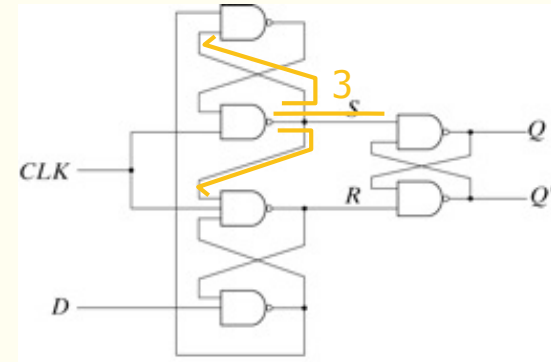
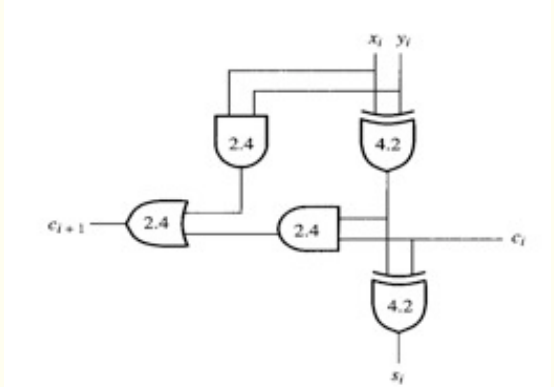


Fig. 5-10 D-Type Positive-Edge-Triggered Flip-Flop

Fan out of the gate must > 3

Parameters for Digital Logic Families (2/2)

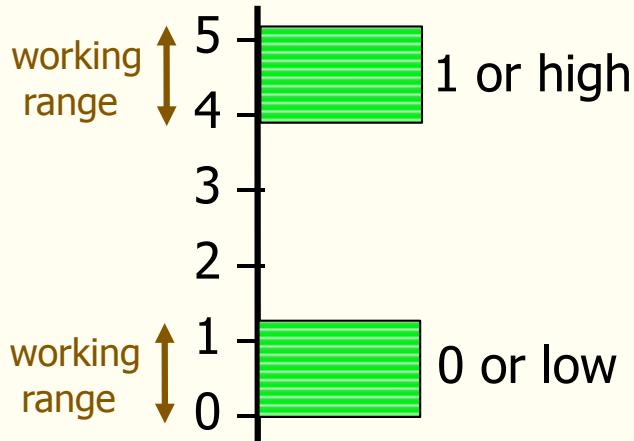


■ Propagation delay

ns = nano seconds

INPUT/OUTPUT PATH	DELAY (ns)
c_i to c_{i+1}	4.8
c_i to s_i	4.2
x_i, y_i to c_{i+1}	<u>9.0</u>
x_i, y_i to s_i	8.4

執行一次最少所需的時間 ← x_i, y_i to c_{i+1}



■ Noise margin

max external noise voltage to an input signal that dose not cause an undesirable change in the circuit output

Computer Aided Design (CAD)

電腦輔助設計，設計自動化的tool

- CAD tools is necessary (> millions of transistors)
- EDA (electronic design automation) is used specially for IC design
 - Schematic editor
 - HDL (Hardware Description Language) 硬體描述語言
 - Logic synthesis (automates the design) 會影響邏輯閘數目，儘量精簡以降成本
- Physical realization of a digital circuit
 - ASIC – application-specific integrated circuit
 - FPGA – Field-programmable gate array
 - CPLD – Complex programmable logic device