CS204: 數位系統設計

Boolean Algebra and Logic Gates

Outline

2.1 Introduction (p.54)

- A set is collection of elements having the same property
  - S: set, x and y: element or event
  - For example: $S = \{1, 2, 3, 4\}$
    - If $x = 2$, then $x \in S$.
    - If $y = 5$, then $y \notin S$.
- A binary operator defines a rule which assigns a pair of elements from S to a unique element from S
  - For example: given a set S, consider $a * b = c$ and $*$ is a binary operator.
  - If (a, b) through $*$ get c and $a, b, c \in S$, then $*$ is a binary operator of S.
  - On the other hand, if $*$ is not a binary operator of S and $a, b \in S$, then $c \notin S$.

Common Algebraic Postulates (p.55)

- A set of elements *S* and two binary operators + and *

1. Closure property: a set *S* is closed with respect to a binary operator if, for every pair of elements of *S*, the binary operator specifies a rule for obtaining a unique element of *S*.
   - $x, y \in S, \ni x + y \in S$ (such that)
     - $a + b = c$, for any $a, b, c \in N$ (Natural numbers). ("+" binary operator plus has closure)
     - But operator – is not closed for *N*, because $2 - 3 = -1$ and $2, 3 \in N$, but $(-1) \notin N$.
2. Associative law: a binary operator * on a set *S* is said to be associative whenever
   - $(x * y) * z = x * (y * z)$ for all $x, y, z \in S$
     - $(x + y) + z = x + (y + z)$
3. Commutative law: a binary operator * on a set *S* is said to be commutative whenever
   - $x * y = y * x$ for all $x, y \in S$
     - $x + y = y + x$

Common Algebraic Postulates (p.55)

4. **Identity element**: a set S is said to have an identity element e with respect to a binary operator * on S if there exists an element $e \in S$ with the property that
   - $e * x = x * e = x$ for every $x \in S$
     - » $0 + x = x + 0 = x$ for every $x \in I$. $I = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$.
     - » $1 * x = x * 1 = x$ for every $x \in I$. $I = \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots\}$.
5. **Inverse**: a set S having the identity element e with respect to a binary operator * is said to have an inverse whenever, for every $x \in S$, there exists an element $y \in S$ such that
   - $x * y = e$
     - » In I and the operator +, with $e = 0$, the inverse of an element a is $(-a)$, since $a + (-a) = 0$.
6. **Distributive law**: if * and + are two binary operators on a set S, * is said to be distributive over + whenever
   - $x * (y + z) = (x * y) + (x * z)$

2.3 Six Postulates of Huntington for Boolean Algebra (p.56)

- A set of elements *B* and two binary operators + and ·
  - Closure: (a) + and (b) · are closed
  - + have identity 0, · have identity 1
    - (a) $x + 0 = 0 + x = x$
    - (b) $x \cdot 1 = 1 \cdot x = x$
  - + and · are commutative
    - (a) $x + y = y + x$
    - (b) $x \cdot y = y \cdot x$
  - Distributive
    - (a) · is distributive over +
      - $x \cdot (y + z) = xy + xz$
    - (b) + is distributive over ·
      - $x + (y \cdot z) = (x + y) \cdot (x + z)$
  - Each $x \in B$, $\ni x' \in B$ (complement) s.t.
    - (a) $x + x' = 1$
    - (b) $x \cdot x' = 0$
  - At least exist two elements $x, y \in B$ and $x \neq y$

Note

- The associative law can be derived
- No additive and multiplicative inverses
- Complement

歷史: 1854年 George Boole 發展布林代數; 1904 E. V. Huntingtion 提出布林代數的公理; 1938年 C. E. Shannon 介紹一種二值布林代數，稱為交換代數 (switching algebra)，說明了雙穩態的電子交換電路，可以用交換代數來表示。

Postulates of Two-Valued Boolean Algebra (1/3) (p.57)

- B = {0, 1} and two binary operators, + and ·
- The rules of operations: AND、OR and NOT.

AND

x	y	$x \cdot y$
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

1. Closure (+ and·)
2. The identity elements

   (1) ＋: 0 → 0 + 0 = 0, 0 + 1 = 1 + 0 = 1

   (2) · : 1 → 1 · 1 = 1, 1 · 0 = 0 · 1 = 0

Postulates of Two-Valued Boolean Algebra (2/3) (p.58)

3. The commutative laws
4. The distributive laws

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

Postulates of Two-Valued Boolean Algebra (3/3) (p.58)

5. Complement
   - $x + x' = 1 \rightarrow 0 + 0' = 0 + 1 = 1;\ 1 + 1' = 1 + 0 = 1$
   - $x \cdot x' = 0 \rightarrow 0 \cdot 0' = 0 \cdot 1 = 0;\ 1 \cdot 1' = 1 \cdot 0 = 0$
6. Has two distinct elements 1 and 0, with $0 \neq 1$

- Note
  - A set of two elements
  - $+$: OR operation; $\cdot$: AND operation
  - A complement operator: NOT operation
  - Binary logic (defined in Chap 1.9) is a two-valued Boolean algebra

2.4 Basic Theorems and Properties (p.59)

- **Property: duality principle**
  - Every algebraic expression deducible from the postulates of Boolean algebra remains valid if
    - » The binary operators are interchanged; AND (·) ⇔ OR (+), and
    - » the identity elements are interchanged; 1 ⇔ 0
- **Basic Theorems**

Table 2.1
Postulates and Theorems of Boolean Algebra

Postulate 2	(a)	$x + 0 = x$	(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)	$x(x + y) = x$

Basic Theorems (1/3) (p.60)

- Theorem 1(a): $x + x = x$

◆ $x + x = (x + x) \cdot 1$	by postulate: 2(b)
$= (x + x) \cdot (x + x')$	5(a)
$= x + xx'$	4(b)
$= x + 0$	5(b)
$= x$	2(a)

- Theorem 1(b): $x \cdot x = x$

◆ $x \cdot x = xx + 0$	by postulate: 2(a)
$= xx + xx'$	5(b)
$= x \cdot (x + x')$	4(a)
$= x \cdot 1$	5(a)
$= x$	2(b)

Basic Theorems (2/3) (p.60)

- Theorem 2(a): $x + 1 = 1$

$x + 1 = 1 \cdot (x + 1)$	postulate 2(b)
$=(x + x')(x + 1)$	5(a)
$= x + x' 1$	4(b)
$= x + x'$	2(b)
$= 1$	5(a)

- Theorem 2(b): $x \cdot 0 = 0$ by duality
- Theorem 3: $(x')' = x$
  - Postulate 5 defines the complement of x, $x + x' = 1$ and $x x' = 0$
  - The complement of x' is $(x')' = x$

Basic Theorems (3/3) (p.61)

- Theorem 6(a): $x + xy = x$
  - $x + xy = x \cdot 1 + xy$ postulate2(b)
  - $= x(1 + y)$ 4(a)
  - $= x(y + 1)$ 3(a)
  - $= x \cdot 1$ T.2(a)
  - $= x$ 2(b)
- Theorem 6(b): $x(x + y) = x$ by duality
- By means of truth table (another way to proof)

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

DeMorgan's Theorems (p.61)

- **DeMorgan's Theorems**
  - $(x + y)' = x' y'$
  - $(x y)' = x' + y'$

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	xy	$(xy)'$	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 (p.61)

- The operator precedence for evaluating Boolean Expression is
  - Parentheses (括弧)
  - NOT
  - AND
  - OR
- Examples
  - $x\,y' + z$
  - $(x\,y + z)'$

2.5 Boolean Functions (p.62)

- A Boolean function (described by an algebraic expression)
  - Binary variables
  - Binary operators OR and AND
  - Unary operator NOT
  - Parentheses
- Examples
  - $F_1 = x\,y\,z'$
  - $F_2 = x + y'z$
  - $F_3 = x'\,y'\,z + x'\,y\,z + x\,y'$
  - $F_4 = x\,y' + x'\,z$

Boolean Functions (p.62)

- The truth table of 2^n entries

$F_1 = x\,y\,z'$

$F_2 = x + y'z$

$F_3 = x'\,y'\,z + x'\,y\,z + x\,y'$

$F_4 = x\,y' + x'\,z$

x	y	z	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

- Two Boolean expressions may specify the same function
  - $F_3 = F_4$

Boolean Functions (p.63)

- Implementation with logic gates

$F_1 = x\,y\,z'$

$F_2 = x + y'z$

Boolean Functions (p.63)

- $F_3 = F_4$
- F_4 is more economical

$$F_3 = x'y'z + x'yz + xy'$$

$$F_4 = xy' + x'z$$

Algebraic Manipulation (p.64)

- Minimize Boolean expressions
  - Literal: a primed or unprimed variable (an input to a gate)
  - Term: an implementation with a gate
  - The minimization of the number of literals and the number of terms → a circuit with less hardware resource
  - It is a hard problem (no specific rules to follow)
- Example 2.1
  1. $x(x' + y)$
  2. $x + x'y$
  3. $(x + y)(x + y')$
  4. $xy + x'z + yz$
  5. $(x + y)(x' + z)(y + z)$

Complement of a Function (p.65)

- An interchange of 0's for 1's and 1's for 0's in the value of *F*
  - By DeMorgan's theorem
  - $(A+B+C)' = (A+X)'$ let $B+C = X$

$(A+B+C)' = (A+X)'$	let $B+C = X$
$= A'X'$	by theorem 5(a) (DeMorgan's)
$= A'(B+C)'$	substitute $B+C = X$
$= A'(B'C')$	by theorem 5(a) (DeMorgan's)
$= A'B'C'$	by theorem 4(b) (associative)

- *Generalizations*: a function is obtained by interchanging AND and OR operators and complementing each literal
  - $(A+B+C+D+ \ldots +F)' = A'B'C'D' \ldots F'$
  - $(ABCD \ldots F)' = A'+B'+C'+D' \ldots +F'$

Examples (p.66)

- **Example 2.2**
  - $F_1' = (x'yz' + x'y'z)'$
  - $F_2' = [x(y'z'+yz)]'$

- **Example 2.3: a simpler procedure**
  - Take the dual of the function and complement each literal

1. $F_1 = x'yz' + x'y'z$

2. $F_2 = x(y'z' + yz)$

2.6 Canonical and Standard Forms (p.67)

Minterms and Maxterms

- A minterm (standard product): 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 maxterm (standard sums): an OR term
  - It is also call a standard sum
  - 2^n maxterms

Minterms and Maxterms (1/3) (p.67)

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

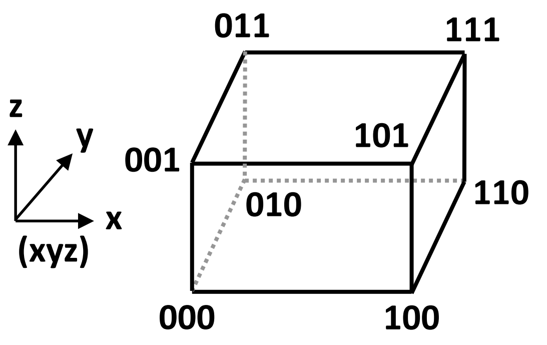


Table 2.3
Minterms and Maxterms for Three Binary Variables

			Minterms		Maxterms	
x	*y*	*z*	**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

Minterms and Maxterms (2/3) (p.68)

- An Boolean function can be expressed by
  - A truth table
  - Sum of minterms
  - $f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$
  - $f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$

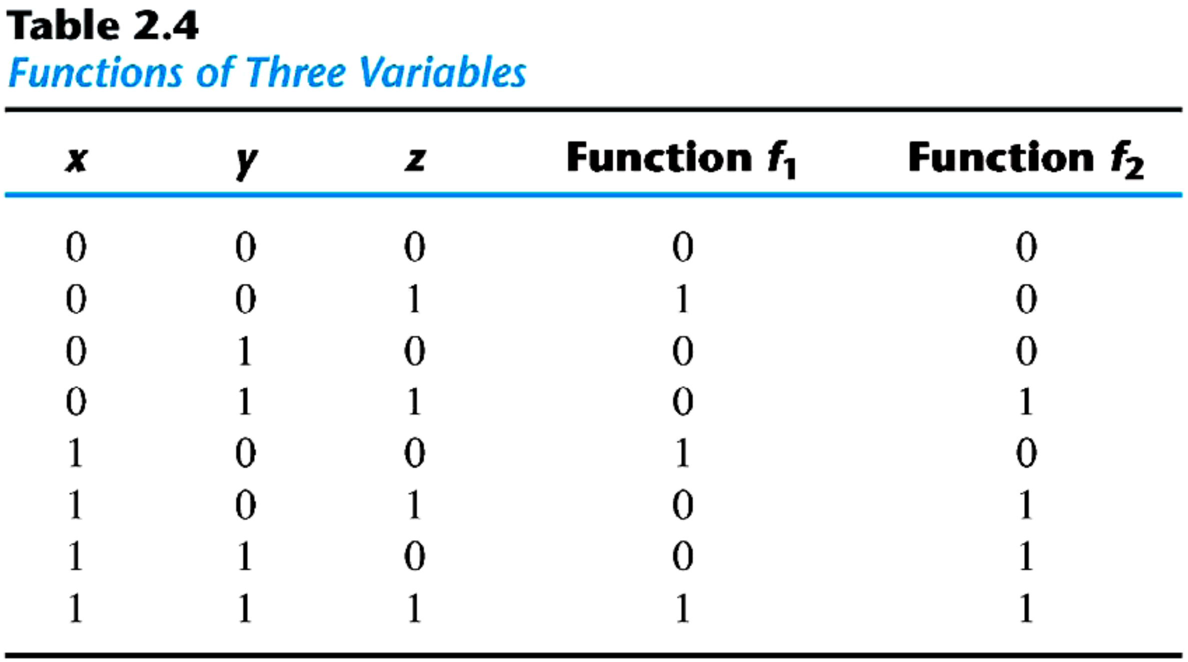


Table 2.4
Functions of Three Variables

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

Minterms and Maxterms (3/3) (p.68)

- The complement of a Boolean function
  - The minterms that produce a 0
  - $f_1' = m_0 + m_2 + m_3 + m_5 + m_6 = x'y'z' + x'yz' + x'yz + xy'z + xyz'$
  - $f_1 = (f_1')' = (x+y+z)(x+y'+z)(x+y'+z')(x'+y+z')(x'+y'+z) = M_0 M_2 M_3 M_5 M_6$
  - $f_2 = (x+y+z)(x+y+z')(x+y'+z)(x'+y+z) = M_0 M_1 M_2 M_4$
- Any Boolean function can be expressed as
  - A sum of minterms (“sum” meaning the ORing of terms)
    - » That produce a 1
  - A product of maxterms (“product” meaning the ANDing of terms)
    - » That produce a 0
  - Both Boolean functions are said to be in Canonical form

Sum of Minterms (p.68)

- Example 2.4: express $F = A+B'C$ as a sum of minterms
  - $F = A + B'C$

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

Product of Maxterms (p.70)

- Product of maxterms: using distributive law to expand.
  - $x + yz$
- Example 2.5: express $F = xy + x'z$ as a product of maxterms.

Conversion between Canonical Forms (p.71)

- The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function
  - $F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$
  - Then, $F'(A, B, C) = \Sigma(0, 2, 3)$
  - By DeMorgan's theorem

    $F(A, B, C) = \Pi(0, 2, 3)$

    $F'(A, B, C) = \Pi(1, 4, 5, 6, 7)$
  - $m_j' = M_j$
  - Sum of minterms $\leftrightarrow$ product of maxterms
  - Interchange the symbols Σ and Π and list those numbers missing from the original form
    - Σ of 1's
    - Π of 0's

Conversion Example

- Example
  - $F = xy + x'z$
  - $F(x, y, z) = \Sigma(1, 3, 6, 7)$
  - $F(x, y, z) = \Pi(0, 2, 4, 5)$

Table 2.6
Truth Table for $F = xy + x'z$

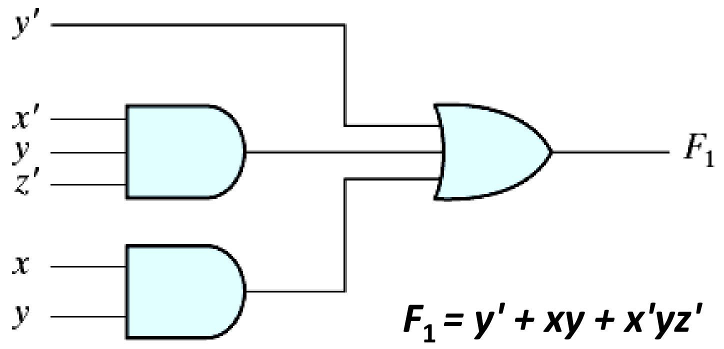
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Standard Forms (P.72)

- Canonical forms are very seldom with the least number of literals
- Standard forms: the terms that form the function may obtain one, two, or any number of literals
  - Sum of products: $F_1 = y' + xy + x'yz'$
  - Product of sums: $F_2 = x(y' + z)(x' + y + z')$
  - Nonstandard form: $F_3 = AB + C(D + E)$

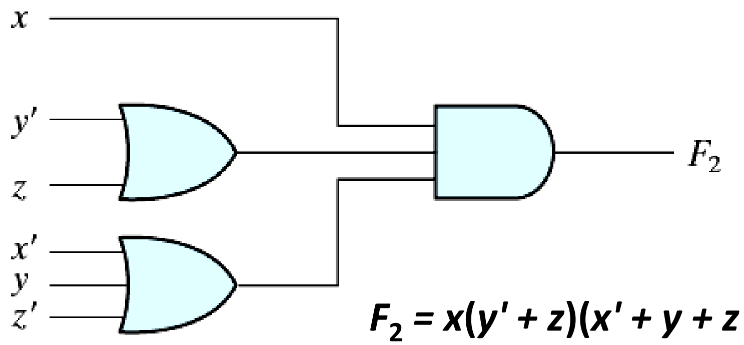
Implementation (p.73)

Two-level implementation



$F_1 = y' + xy + x'yz'$

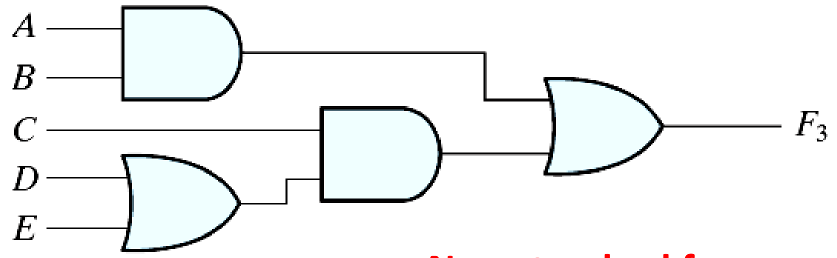
(a) Sum of Products



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

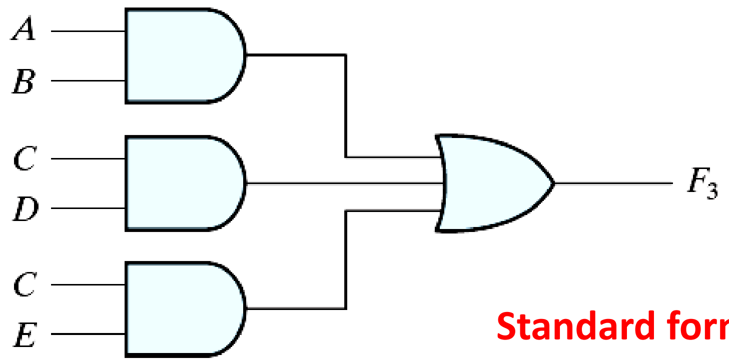
(b) Product of Sums

Three-/multi-level implementation



Non-standard form

(a) $AB + C(D + E)$



Standard form

(b) $AB + CD + CE$

2.7 Other Logic Operations (p.74)

- 2^n rows in the truth table of n binary variables
- 2^{2^n} functions for n binary variables

- 16 functions of two binary variables

Table 2.7
Truth Tables for the 16 Functions of Two Binary Variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Boolean Expressions (p.75)

Table 2.8
Boolean Expressions for the 16 Functions of Two Variables

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y, then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x, then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

All the new symbols except for the exclusive-OR symbol are not commonly used by digital designers

2.8 Digital Logic Gates (p.76)

- Boolean expression: AND, OR, and NOT operations
- Considerations for constructing gates of other logic operations
  - The feasibility and economy
  - The possibility of extending gate's inputs
  - The basic properties of the binary operations (e.g., commutative and associative)
  - The ability of the gate to implement Boolean functions alone or in conjunction with other gates

Standard Gates (p.76)

- Consider the 16 functions in Table 2.8
  - Two are equal to a constant (F_0 and F_{15})
  - Four are repeated twice (F_4, F_5, F_{12} and F_{13})
  - Inhibition (F_2) and implication (F_{11}) are not commutative or associative
  - The other eight: complement (F_{10}), transfer (F_3), AND (F_1), OR (F_7), NAND (F_{14}), NOR (F_8), XOR (F_6), and equivalence (XNOR) (F_9) are used as standard gates
  - Complement: inverter
  - Transfer: buffer/repeater (increasing drive strength)
  - Equivalence: XNOR

Summary of Logic Gates (p.77)

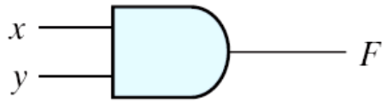
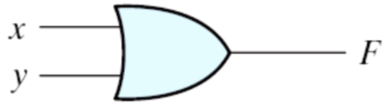
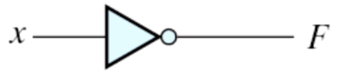
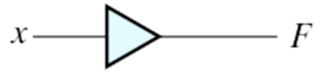
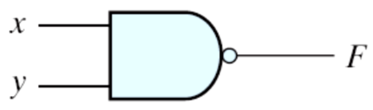
Name	Graphic symbol	Algebraic function	Truth table
AND		$F = xy$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$F = x'$	x | F 0 | 1 1 | 0
Buffer		$F = x$	x | F 0 | 0 1 | 1

Figure 2.5 Digital logic gates

Summary of Logic Gates (p.77)

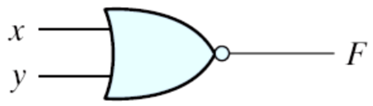
NAND



$F = (xy)'$

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

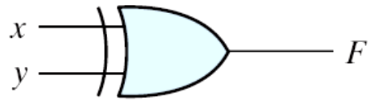
NOR



$F = (x + y)'$

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

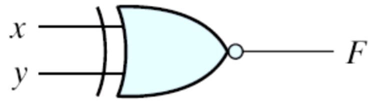
Exclusive-OR (XOR)



$F = xy' + x'y$
$= x \oplus y$

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

Exclusive-NOR or equivalence



$F = xy + x'y'$
$= (x \oplus y)'$

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

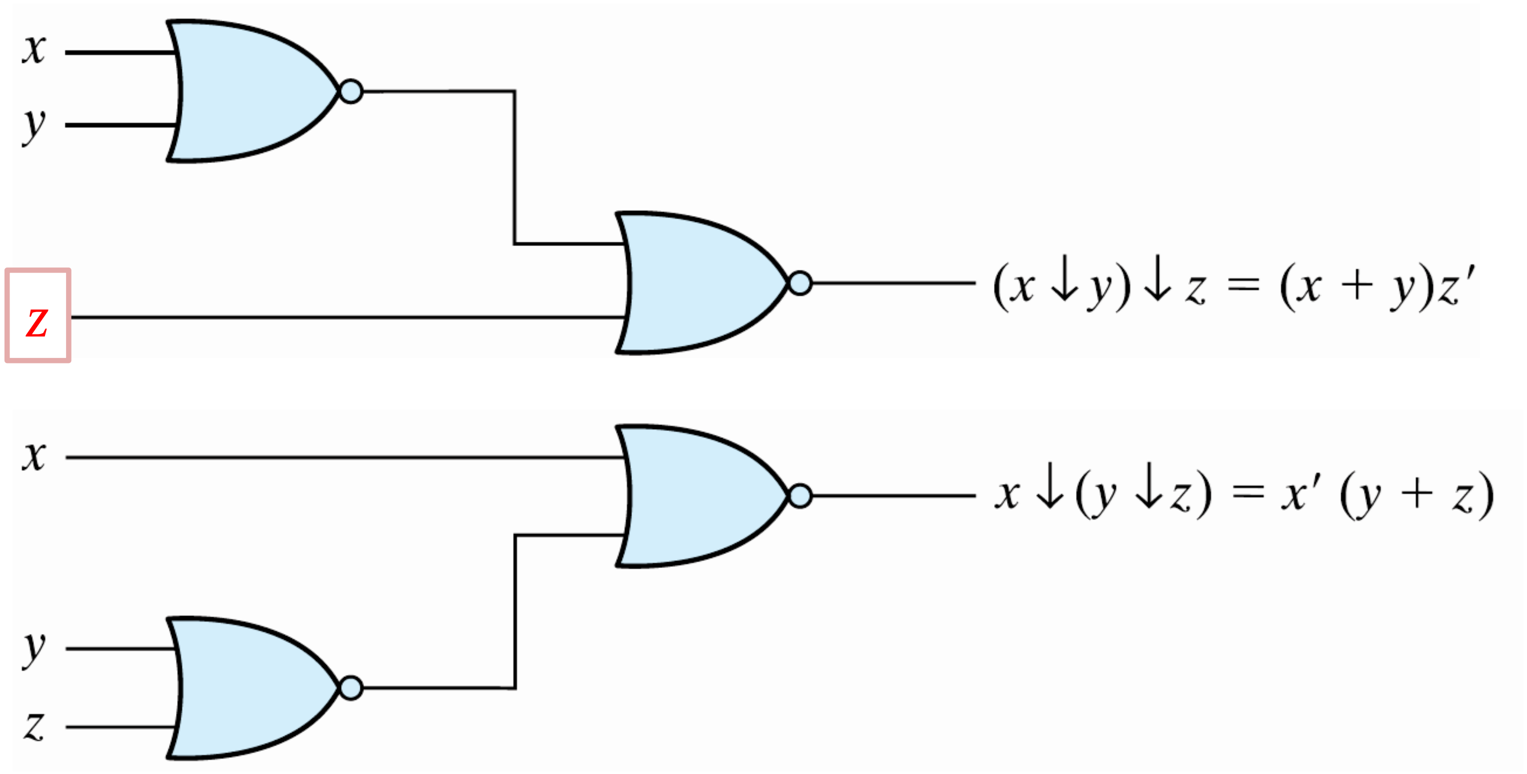
Figure 2.5 Digital logic gates

Multiple Inputs (p.78)

- 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
    - OR
      - $x + y = y + x$
      - $(x + y) + z = x + (y + z) = x + y + z$
    - AND
      - $x\,y = y\,x$
      - $(x\,y)\,z = x\,(y\,z) = x\,y\,z$

Multiple Inputs (p.79)

- **NAND and NOR are commutative but not associative → they are not extendable**

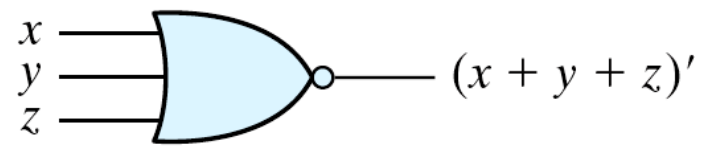


x
y
z
$(x \downarrow y) \downarrow z = (x + y)z'$
x
$x \downarrow (y \downarrow z) = x' (y + z)$
y
z

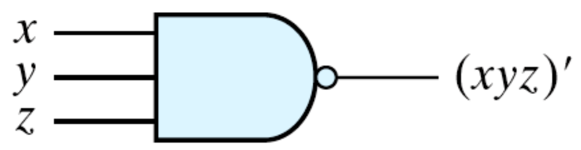
Figure 2.6 Demonstrating the nonassociativity of the NOR operator:
$(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$

Multiple Inputs (p.79)

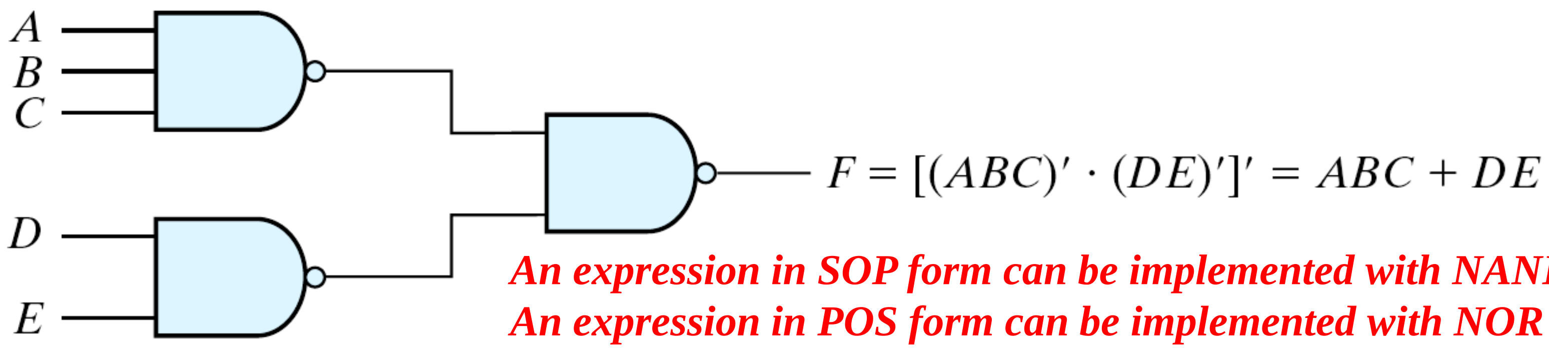
- Multiple NOR = a complement of OR gate
- Multiple NAND = a complement of AND
- The cascaded NAND operations = sum of products
- The cascaded NOR operations = product of sums



(a) 3-input NOR gate



(b) 3-input NAND gate



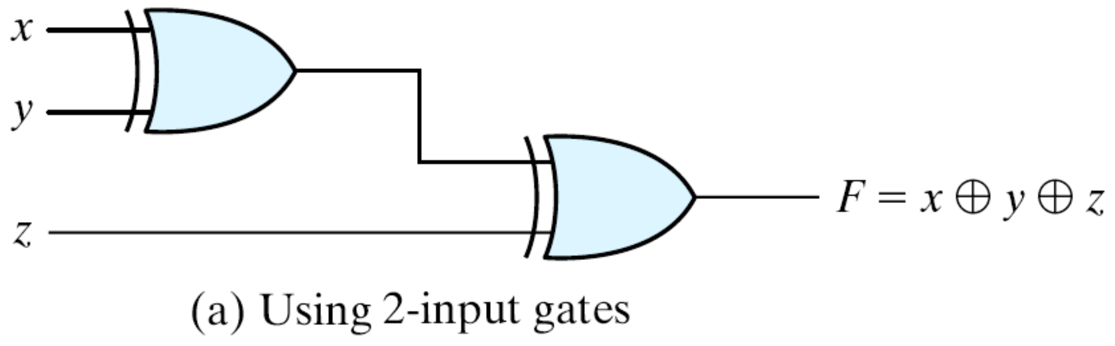
An expression in SOP form can be implemented with NAND gates
An expression in POS form can be implemented with NOR gates

(c) Cascaded NAND gates

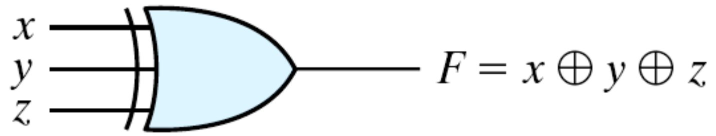
Figure 2.7 Multiple-input and cascaded NOR and NAND gates

Multiple Inputs (p.79)

- The XOR and XNOR gates are commutative and associative
- Multiple-input XOR gates are uncommon
- XOR is an odd function: 1 if the input variables have odd number of 1's



(a) Using 2-input gates



(b) 3-input gate

x	y	z	F
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

(c) Truth table

Figure 2.8 3-input XOR gate

Positive and Negative Logic (p.80)

- Positive and Negative Logic System
  - Two signal values <=> two logic values
  - Positive logic system: H=1; L=0
  - Negative logic system: H=0; L=1
- Consider a gate (next page)
  - A positive logic AND gate
  - A negative logic OR gate
  - The positive logic is used in this book

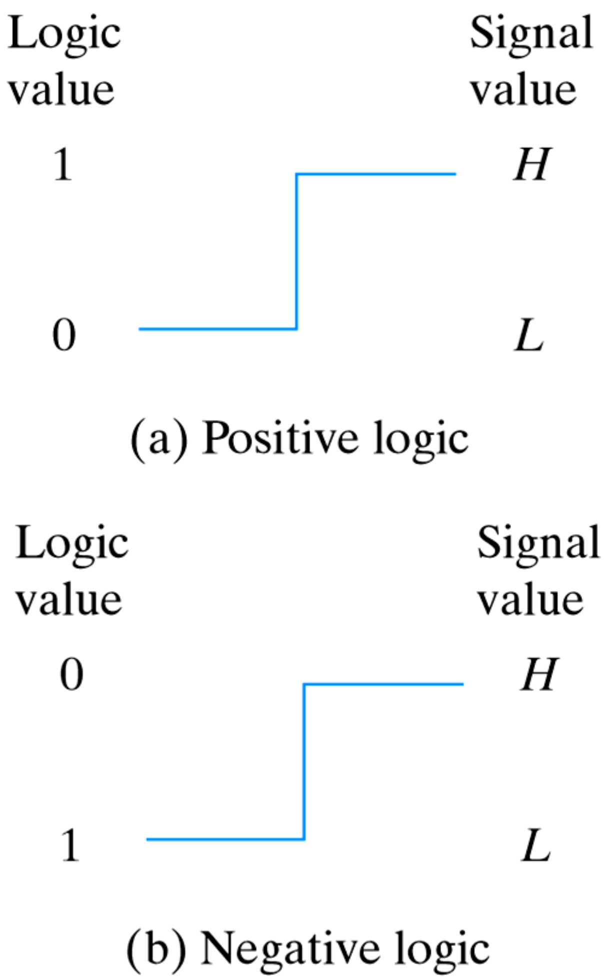
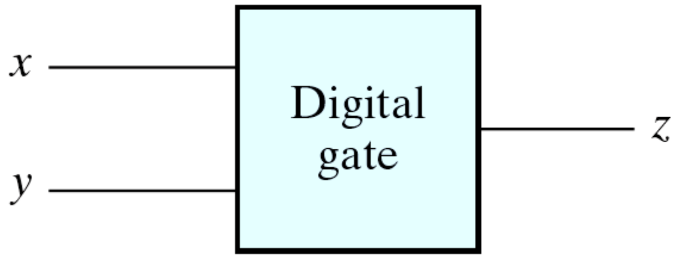


Figure 2.9 Signal assignment and logic polarity

Positive and Negative Logic (p.81)

x	y	z
L	L	L
L	H	L
H	L	L
H	H	H

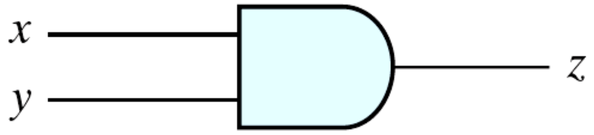
(a) Truth table with H and L



(b) Gate block diagram

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

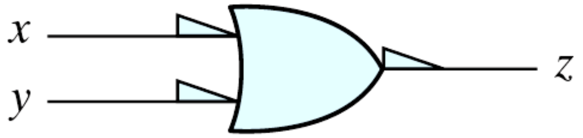
(c) Truth table for positive logic



(d) Positive logic AND gate

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

(e) Truth table for negative logic



(f) Negative logic OR gate

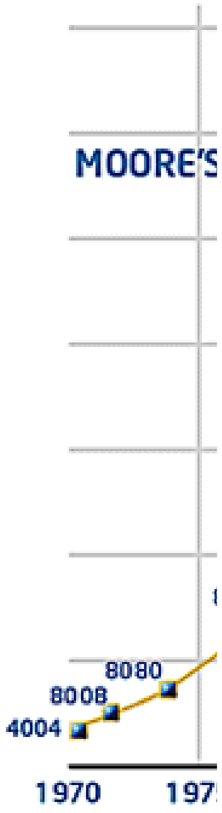
Figure 2.10 Demonstration of positive and negative logic

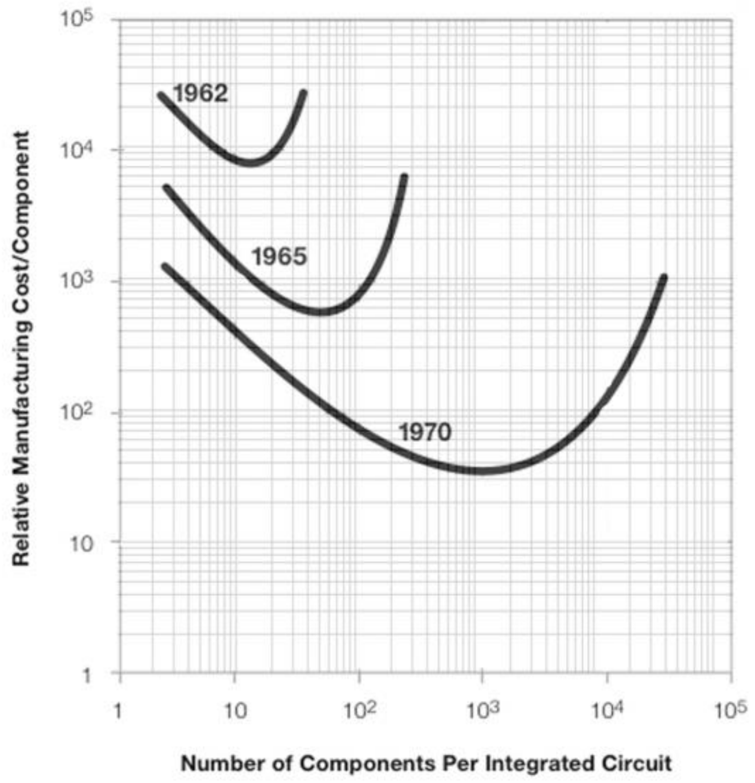
2.9 Integrated Circuits (p.82)

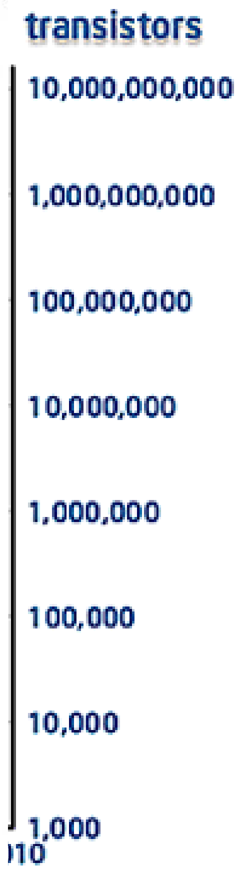
- **An IC (a chip) is fabricated on a *die* of a silicon semiconductor crystal, containing the electronic components for constructing digital gates**
- **Level of Integration:**
  - **Small-scale Integration (SSI): < 10 gates**
  - **Medium-scale Integration (MSI): 10 ~ 100 gates**
  - **Large-scale Integration (LSI): 100 ~ *n* k gates**
  - **Very Large-scale Integration (VLSI): > *n* k gates**
- **VLSI**
  - **Small size (compact size)**
  - **Low cost**
  - **Low power consumption**
  - **High reliability**
  - **High speed**

Moore's Law

- Transistors on lead microprocessors double every 2 years (transistors on electronic component double every 18 months)



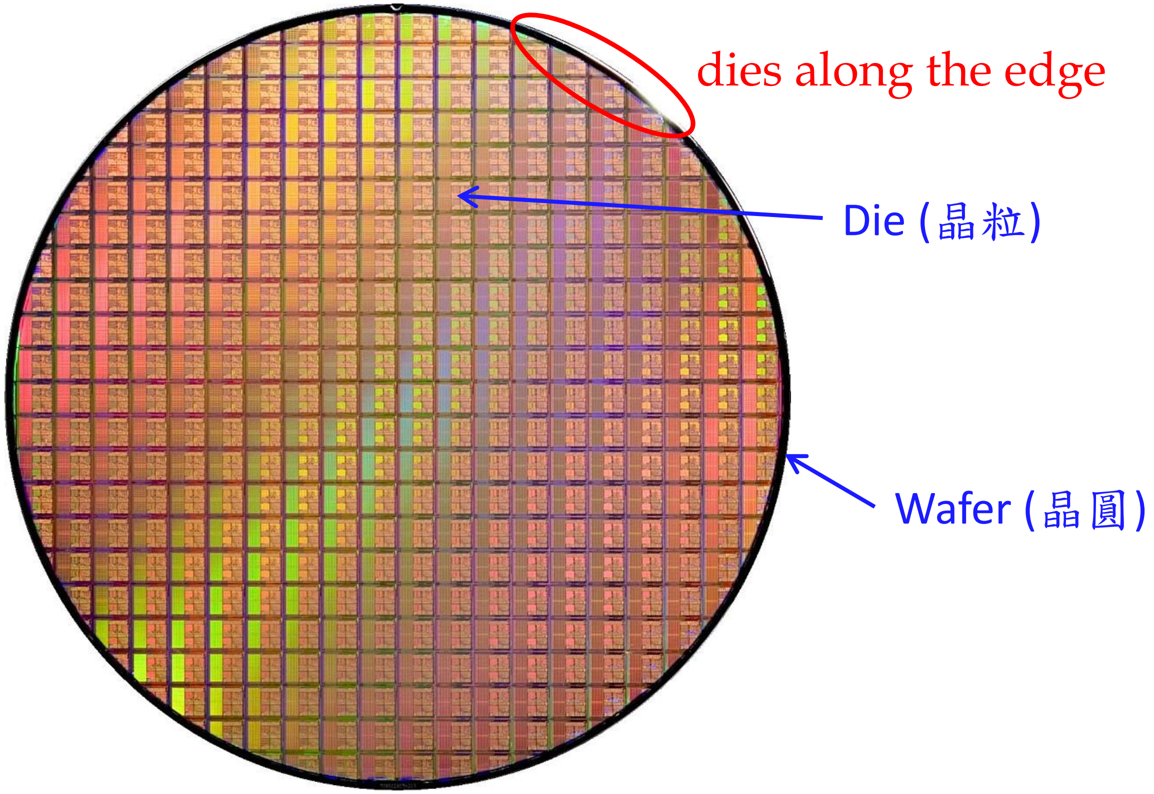




source: http://www.intel.com

Silicon Wafer and Dies

- **Exponential cost decrease – technology basically the same:**
  - **A *wafer* is tested and chopped into *dies* that are packaged**



AMD K8, source: http://www.amd.com

Cost of an Integrated Circuit (IC)

$$\text{Cost of IC} = \frac{\text{Cost of die} + \text{Cost of testing die} + \text{Cost of packaging and final test}}{\text{Final test yield}}$$

(A greater portion of the cost that varies between machines)

$$\text{Cost of die} = \frac{\text{Cost of wafer}}{\#\text{Dies per wafer} \times \text{Die yield}}$$

$$\#\text{Dies per wafer} = \frac{\pi \times (\text{Wafer radius})^2}{\text{Die area}} - \frac{\pi \times \text{Wafer diameter}}{\sqrt{2 \times \text{Die area}}}$$

(sensitive to die size)

(# of dies along the edge)

$$\text{Die yield} = \text{Wafer yield} \times \left(1 + \frac{\text{Defect density} \times \text{Die area}}{\alpha}\right)^{-\alpha}$$

Today's technology: $\alpha \approx 4.0$, defect density 0.4 ~ 0.8 per cm^2

Examples of Cost of an IC

- **Example 1: Find the number of dies per 30-cm wafer for a die that is 0.7 cm on a side.**
  - **The total die area is 0.49 cm^2. Thus**

$$\#\text{Dies per wafer} = \frac{\pi \times (\text{Wafer radius})^2}{\text{Die area}} - \frac{\pi \times \text{Wafer diameter}}{\sqrt{2 \times \text{Die area}}}$$

$$= \frac{\pi \times (30/2)^2}{0.49} - \frac{\pi \times 30}{\sqrt{2 \times 0.49}} = \frac{706.5}{0.49} - \frac{94.2}{0.99} = 1{,}347$$

- **Example 2: Find the die yield for dies that are 1 cm on a side and 0.7 cm on a side, assuming a defect density of 0.6 per cm^2.**
  - **The total die areas are 1 cm^2 and 0.49 cm^2. For the large die the yield is**

$$\text{Die yield} = \text{Wafer yield} \times \left(1 + \frac{\text{Detect desity} \times \text{Die area}}{\alpha}\right)^{-\alpha}$$

$$= \left(1 + \frac{0.6 \times 1}{4.0}\right)^{-4} = 0.35$$

For the small die, it is

$$\text{Die yield} = \left(1 + \frac{0.6 \times 0.49}{4.0}\right)^{-4} = 0.58$$

Digital Logic Families (p.82)

- Digital logic families: circuit technology
  - TTL: transistor-transistor logic
  - ECL: emitter-coupled logic (high speed, high power consumption)
  - MOS: metal-oxide semiconductor (NMOS, high density)
  - CMOS: complementary MOS (low power)
  - BiCMOS: high speed, high density

Digital Logic Families (p.83)

- The characteristics of digital logic families
  - Fan-out: the number of standard loads that the output of a typical gate can drive
  - Fan-in: the number of inputs available in a gate
  - Power dissipation: the power consumption during circuit switching
  - Propagation delay: the average transition delay time for the signal to propagate from input to output
  - Noise margin: the maximum external noise voltage added to an input signal that does not cause an undesirable change in the circuit output

CAD (p.83)