Principles Of

Digital Design

Chapter 3

Boolean Algebra and Logic Design

- *Boolean Algebra*
- *Logic Gates*
- *Digital Design*
- *Implementation Technology*
  - *ASICs*
  - *Gate Arrays*

Basic Algebraic Properties

- A ***set*** is a collection of objects with a common property
  - If S is a set and x is a member of the set S, then $x \in S$
    - $A = \{1, 2, 3, 4\}$ denotes the set A, whose elements are $1, 2, 3, 4$

- A ***binary operator*** on a set S is a rule that assigns to each pair of elements in S another element that is in S

- ***Axioms*** are assumption that are valid without proof

Slides by Philip Pham, University of California, Irvine

Examples of Axioms

- **Closure**
  - A set S is closed with respect to a binary operator $\bullet$ iff for all $x, y \in S$, $(x \bullet y) \in S$
    - $Z^+ = \{1, 2, 3, \ldots\}$ is closed to addition, because positive numbers are in Z^+

- **Associativity**
  - A binary operator $\bullet$ defined on a set S is associative iff for all $x, y, z \in S$

$$(x \bullet y) \bullet z = x \bullet (y \bullet z)$$

- **Identity Element**
  - A set S has an identity element e for every $x \in S$

$$e \bullet x = x \bullet e = x$$

$$x + 0 = 0 + x = x$$

Slides by Philip Pham, University of California, Irvine

Examples of Axioms

- **Commutativity**
  - A binary operator • is commutative iff for all $x, y \in S$

$$x \bullet y = y \bullet x$$

- **Inverse Element**
  - A set S has an inverse iff for every $x \in S$, there exists an element $y \in S$ such that

$$x \bullet y = e$$

- **Distributivity**
  - If • and □ are two binary operators on a set S, • is said to be distributive over □ if, for all $x, y, z \in S$

$$x \bullet (y \square z) = (x \bullet y) \square (x \bullet z)$$

Axiomatic Definition of Boolean Algebra

Boolean algebra is a set of elements B with two binary operators, + and ·, which satisfies the following six axioms:

- **Axiom 1 (Closure Property): (a) B is closed with respect to the operator +; (b) B is also closed with respect to the operator ·**
- **Axiom 2 (Identity Element): (a) B has an identity element with respect to +, designated by 0; (b) B also has an identity element with respect · , designated by 1**
- **Axiom 3 (Commutativity Property): (a) B is commutative with respect to +; (b) B is also commutative with respect to ·**
- **Axiom 4 (Distributivity Property): (a) The operator · is distributive over +; (b) similarly, the operator + is distributive over ·**
- **Axiom 5 (Complement Element): For every $x \in B$, there exists an element $x' \in B$ such that (a) $x + x' = 1$ and (b) $x \cdot x' = 0$**
  This second element x', is called the complement of x
- **Axiom 6 (Lower Cardinality Bound): There are at least two elements $x, y \in B$ such that $x \neq y$**

Axiomatic Definition of Boolean Algebra

Differences between Boolean algebra and ordinary algebra

- In ordinary algebra, + is not distributive ·
- Boolean algebra does not have inverses with respect to + and · ; therefore, there are no subtraction or division operations in Boolean algebra
- Complements are available in Boolean algebra, but not in ordinary algebra
- Boolean algebra applies to a finite set of elements, whereas ordinary algebra would apply to the infinite sets of real numbers
- The definition above for Boolean algebra does not include associativity, since it can be derived from the other axioms

Two-valued Boolean Algebra

- **Set** B **has two elements:** 0 **and** 1
- **Algebra has two operators:** AND **and** OR

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

AND Operator

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

OR Operator

Two-valued Boolean Algebra

Two-valued Boolean algebra satisfies Huntington axioms

- **Axiom 1 (Closure Property): Closure is evident in the AND/OR tables, since the result of each operation is an element of *B*.**

- **Axiom 2 (Identity Element): The identity elements in this algebra are 0 for the operator + and 1 for the operator ·. From the AND/OR tables, we see that:**
  - $0 + 0 = 0$, and $0 + 1 = 1 + 0 = 1$
  - $1 \cdot 1 = 1$, and $1 \cdot 0 = 0 \cdot 1 = 0$

- **Axiom 3 (Commutativity Property): The commutativity laws follow from the symmetry of the operator tables.**

Two-valued Boolean Algebra

- **Axiom 4 (Distributivity): The distributivity of this algebra can be demonstrated by checking both sides of the equation.**

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

x	y	z	$y+z$	$x \cdot (y+z)$	xy	xz	$(xy)+(xz)$
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

Proof of distributivity of ·

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

x	y	z	yz	$x+(yz)$	$x+y$	$x+z$	$(x+y)(x+z)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

Proof of distributivity of +

Two-valued Boolean Algebra

- **Axiom 5 (Complement):** 0 **and** 1 **are complements of each other, since** $0 + 0' = 0 + 1 = 1$ **and** $1 + 1' = 1 + 0 = 1$; furthermore, $0 \cdot 0' = 0 \cdot 1 = 0$ **and** $1 \cdot 1' = 1 \cdot 0 = 0$.

x	x'
0	1
1	0

NOT Operator

- **Axiom 6 (Cardinality): The cardinality axiom is satisfied, since this two-valued Boolean algebra has two distinct elements,** 1 **and** 0**, and** $1 \neq 0$.

Boolean Operator Procedure

- **Boolean operators are applied in the following order.**

  - Parentheses ( )
  - NOT ′
  - AND ·
  - OR +

Example: Evaluate expression $(x + xy)'$ for $x = 1$ and $y = 0$:

$$(1 + 1\cdot0)' = (1 + 0)' = (1)' = 0$$

Duality Principle

- **Any algebraic expression derived from axioms stays valid when**
  - OR and AND
  - 0 and 1

are interchanged.

Example:

If

$$X + 1 = 1$$

then

$$X \cdot 0 = 0$$

by the duality principle

Theorem of Boolean Algebra

Theorem 1	**(a)**	$x + x$	$=$	x
(Idempotency)	**(b)**	xx	$=$	x
Theorem 2	**(a)**	$x + 1$	$=$	1
	(b)	$x \cdot 0$	$=$	1
Theorem 3	**(a)**	$yx + x$	$=$	x
(Absorption)	**(b)**	$(y + x)x$	$=$	x
Theorem 4		$(x')'$	$=$	x
(Involution)				
Theorem 5	**(a)**	$(x + y) + z$	$=$	$x + (y + z)$
(Associativity)	**(b)**	$x(yz)$	$=$	$(xy)z$
Theorem 6	**(a)**	$(x + y)'$	$=$	$x'y'$
(De Morgan's Law)	**(b)**	$(xy)'$	$=$	$x' + y'$

Basic Theorems of Boolean Algebra

Theorem Proofs in Boolean Algebra

- **Theorems can be proved by transformations based on axioms and theorems**

  Example:

  Theorem 1(a) Idempotency: $x + x = x$.

  Proof:

$x + x$	$=$	$(x + x) \cdot 1$	by identity (Ax. 2b)
	$=$	$(x + x)(x + x')$	by complement (Ax. 5a)
	$=$	$x + xx'$	by distributivity (Ax. 4b)
	$=$	$x + 0$	by complement (Ax. 5b)
	$=$	x	by identity (Ax. 2a)

- **Duality**

  Example:

  Theorem 1(b) Idempotency: $x \cdot x = x$.

  Proof:

$x + x$	$=$	x	by Theorem 1(a)
$x \cdot x$	$=$	x	by Duality principle

Theorem Proofs in Boolean Algebra

- **Checking theorems for every combinations of variable value**

  Example:

  Theorem 6(a) DeMorgan's Law: $(x + y)' = x'y'$

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

Proof of Demorgan's First Theorem

Slides by Philip Pham, University of California, Irvine

Boolean Functions

- **Algebraic expression, which are formed from binary variables and Boolean operators AND, OR and NOT.**

Example:

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

This function would be equal to 1 if $x = 1$ and $y = 1$, or
if $x = 1$ and $y = 0$ and $z = 1$, or
if $x = 0$ and $y = 1$ and $z = 1$;
otherwise, $F_1 = 0$.

Note 1: When we evaluate Boolean expressions, we must follow a specific order of operations, namely, (1) parentheses, (2) NOT, (3) AND, (4) OR.

Note 2: A primed or unprimed variable is usually called a literal.

Boolean Functions

- **Truth tables which list the functional value for all combinations of variable values.**

  Example:

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

Row Numbers	Variable Values			Function Values
	x	y	z	F_1
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

Complement of a Function

- **Complement of function F is function F', where F' can be obtained by:**
  - **Interchanging 0 and 1 in the truth table.**

  Example:

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

Row Numbers	Variable Values			Function Values	
	x	y	z	F_1	F_1'
0	0	0	0	0	1
1	0	0	1	0	1
2	0	1	0	0	1
3	0	1	1	1	0
4	1	0	0	0	1
5	1	0	1	1	0
6	1	1	0	1	0
7	1	1	1	1	0

Complement of a Function

- Complement of function F is function F', where F' can be obtained by:
  - **Repeatedly applying DeMorgan's theorems.**

Example:

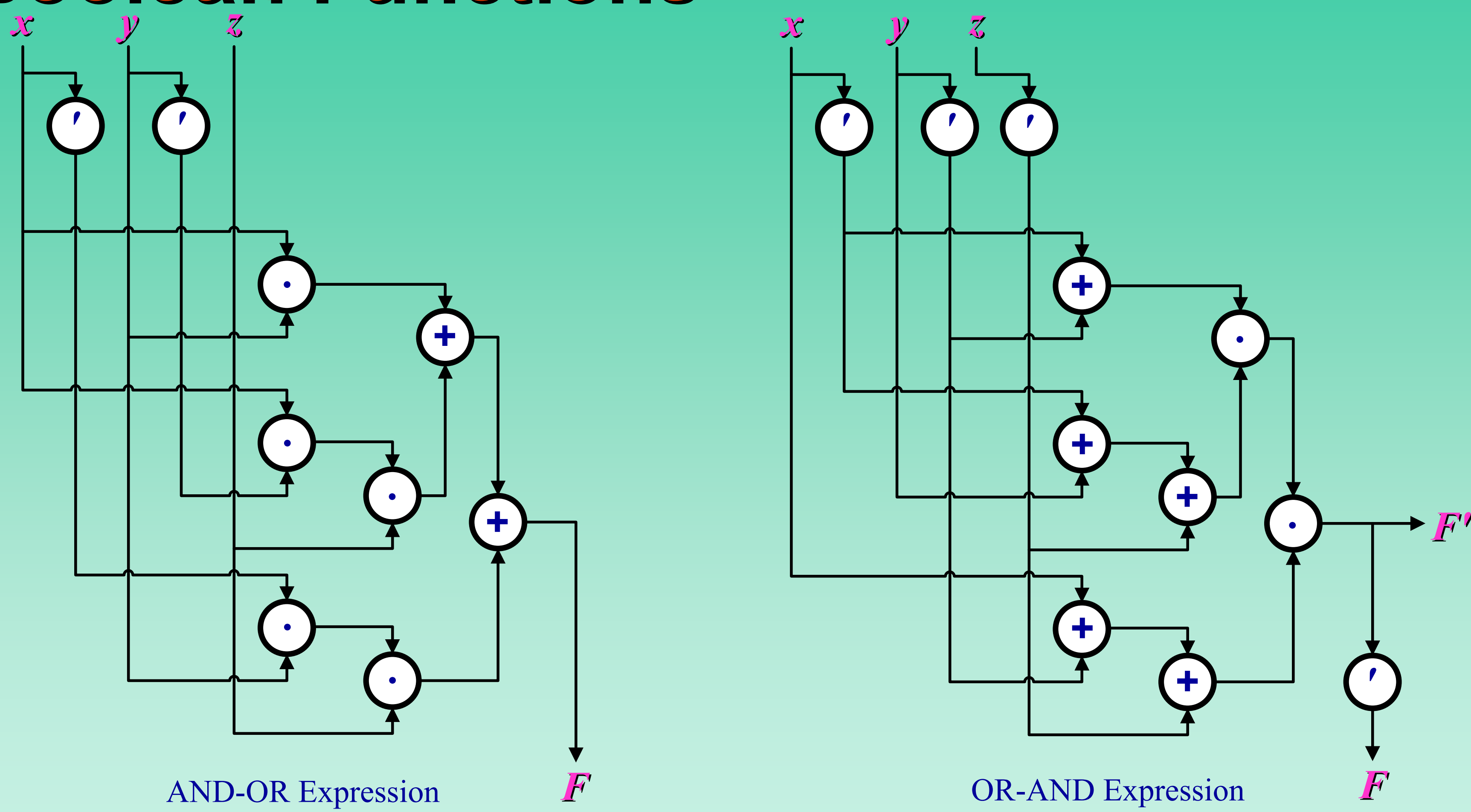
$$F_1' = (xy + xy'z + x'yz)' \quad \text{by definition of } F$$
$$= (xy)'(xy'z)'(x'yz)' \quad \text{by DeMorgan's Th.}$$
$$= (x' + y')(x' + y + z')(x + y' + z') \quad \text{by DeMorgan's Th.}$$

  - **Duality Principle**

Example:

$$F_1 = (x \cdot y) + (x \cdot y' \cdot z) + (x' \cdot y \cdot z)$$
$$\downarrow$$
$$F_1' = (x' + y') \cdot (x' + y + z') \cdot (x + y' + z')$$

Graphic Representation of Boolean Functions



AND-OR Expression

F size = 5 ANDs
2 ORs
2 NOTs

OR-AND Expression

F size = 2 ANDs
5 ORs
4 NOTs

Two different expressions have different sizes

Slides by Philip Pham, University of California, Irvine

Expression Equivalence

- **We can prove expression equivalence by algebraic manipulation in which each transformation uses an axiom or a theorem of Boolean algebra.**

Example:

$$F_1 = xy + xy'z + x'yz$$
$$= xy + xz + yz$$

Proof:

$xy + xy'z + x'yz$	$= xy + xyz + xy'z + x'yz$	by absorption
	$= xy + x(y + y')z + x'yz$	by distributivity
	$= xy + x1z + x'yz$	by complement
	$= xy + xz + x'yz$	by identity
	$= xy + xyz + xz + x'yz$	by absorption
	$= xy + xz + (x + x')yz$	by distributivity
	$= xy + xz + 1yz$	by complement
	$= xy + xz + yz$	by identity

$xy + xy'z + x'yz$	requires	5 ANDs	2 ORs	2 NOTs
$xy + xz + yz$	requires	3 ANDs	2 ORs	
Difference:		2 ANDs	and	2 NOTs

Minterms

- **Minterm definition**

  If $i = b_{n-1} \ldots b_0$ is a binary number between 0 and $2^n - 1$, then a minterm of n variables $x_{n-1}, x_{n-2} \ldots, x_0$, could be represented as:

$$m_i(x_{n-1}, x_{n-2} \ldots, x_0) = y_{n-1} \ldots y_0$$

  where for all k such that $0 \le k \le n-1$,

$$y_k = \begin{cases} x_k & \text{if } b_k = 1 \\ x_k' & \text{if } b_k = 0 \end{cases}$$

x	y	z	Minterms	Designation
0	0	0	$x'y'z'$	m_0
0	0	1	$x'y'z$	m_1
0	1	0	$x'yz'$	m_2
0	1	1	$x'yz$	m_3
1	0	0	$xy'z'$	m_4
1	0	1	$xy'z$	m_5
1	1	0	xyz'	m_6
1	1	1	xyz	m_7

Minterms for Three Binary Variables

Sum-of-Minterms

- **Any Boolean function can be expressed as a sum (OR) of its 1-minterms:**

$$F(\text{list of variables}) = \Sigma(\text{list of 1-minterm indices})$$

Example:

Row Numbers	Variable Values			Function Values	
	x	y	z	F_1	F_1'
0	0	0	0	0	1
1	0	0	1	0	1
2	0	1	0	0	1
3	0	1	1	1	0
4	1	0	0	0	1
5	1	0	1	1	0
6	1	1	0	1	0
7	1	1	1	1	0

Equation Table

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

$$F_1' = (x' + y')(x' + y + z')(x + y' + z')$$

$$\begin{aligned} F_1(x, y, z) &= \Sigma(3, 5, 6, 7) \\ &= m_3 + m_5 + m_6 + m_7 \\ &= x'yz + xy'z + xyz' + xyz \end{aligned}$$

$$\begin{aligned} F_1'(x, y, z) &= \Sigma(0, 1, 2, 4) \\ &= m_0 + m_1 + m_2 + m_4 \\ &= x'y'z' + x'y'z + x'yz' + xy'z' \end{aligned}$$

Expansion to Sum-of-Minterms

- **Any Boolean function can be expanded into a sum-of-minterms form be expanding each term with $(x + x')$ for each missing variable x.**

Example:

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

After removing duplicates and rearranging the minterms in ascending order:

$$\begin{aligned} F &= x'yz + xy'z' + xy'z + xyz' + xyz \\ &= m_3 + m_4 + m_5 + m_6 + m_7 \\ &= \Sigma(3, 4, 5, 6, 7) \end{aligned}$$

Conversion to Sum-of-Minterms

- **Each Boolean function can be converted into a sum-of-minterms form by generating the truth table and identifying 1-minterms.**

Example: $F = x + yz$

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

$$F = m_3 + m_4 + m_5 + m_6 + m_7$$

Maxterms

- **Maxterms can be defined as the complement of minterms:**

$$M_i = m_i' \text{ and } M_i' = m_i$$

x	y	z	Maxterms	Designation
0	0	0	$x + y + z$	M_0
0	0	1	$x + y + z'$	M_1
0	1	0	$x + y' + z$	M_2
0	1	1	$x + y' + z'$	M_3
1	0	0	$x' + y + z$	M_4
1	0	1	$x' + y + z'$	M_5
1	1	0	$x' + y' + z$	M_6
1	1	1	$x' + y' + z'$	M_7

Maxterms for Three Binary Variables

Product-of-Maxterms

- **Any Boolean function can be expressed as a product (AND) of its 0-maxterms:**

$$F(\text{list of variables}) = \Pi(\text{list of 0-maxterm indices})$$

Example:

Row Numbers	Variable Values x	y	z	Function Values F_1	F_1'
0	0	0	0	0	1
1	0	0	1	0	1
2	0	1	0	0	1
3	0	1	1	1	0
4	1	0	0	0	1
5	1	0	1	1	0
6	1	1	0	1	0
7	1	1	1	1	0

Equation Table

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

$$F_1' = (x' + y')(x' + y + z')(x + y' + z')$$

$$\begin{aligned} F_1(x, y, z) &= \Pi(0, 1, 2, 4) \\ &= M_0 M_1 M_2 M_4 \\ &= (x + y + z)(x + y + z')(x + y' + z)(x' + y' + z) \end{aligned}$$

$$\begin{aligned} F_1'(x, y, z) &= \Pi(3, 5, 6, 7) \\ &= M_3 M_5 M_6 M_7 \\ &= (x + y' + z')(x' + y + z')(x' + y' + z)(x' + y' + z') \end{aligned}$$

Product-of-maxterms can also be obtained by complementing the sum-of-minterms

$$\begin{aligned} (F_1)' &= (x'yz + xy'z + xyz' + xyz)' \\ &= (x + y' + z')(x' + y + z')(x' + y' + z)(x' + y' + z') \\ &= M_3 M_5 M_6 M_7 \end{aligned}$$

$$\begin{aligned} F_1 &= (F_1')' \\ &= (x'y'z' + x'y'z + x'yz' + xyz')' \\ &= (x + y + z)(x + y + z')(x + y' + z)(x' + y' + z) \\ &= M_0 M_1 M_2 M_4 \end{aligned}$$

Expansion to Product-of-Maxterms

- **Any Boolean function can be expanded into a product-of-maxterms form be expanding each term with xx' for each missing variable x.**

Example:

Convert

$$
\begin{aligned}
F &= x'y' + xz \\
&= (x'y' + x)(x'y' + z) \\
&= (x' + x)(y' + x)(x' + z)(y' + z) \\
&= (x + y')(x' + z)(y' + z)
\end{aligned}
$$

missing x_i missing y_i missing c_i

Expand

$$
\begin{aligned}
x + y' &= x + y' + zz' &&= (x + y' + z)(x + y' + z') \\
x' + z &= x' + z + yy' &&= (x' + y + z)(x' + y' + z) \\
y' + z &= y' + z + xx' &&= (x + y' + z)(x' + y' + z)
\end{aligned}
$$

Combine

$$
\begin{aligned}
F &= (x + y' + z)(x + y' + z')(x' + y + z)(x' + y' + z) \\
&= M_2 M_3 M_4 M_6 = \prod(2, 3, 4, 6)
\end{aligned}
$$

Conversion to Product-of-Maxterms

- **Any Boolean expression can be converted into a sum-of-maxterms by generating the truth table and listing all the 0-maxterms.**
  - **Example:** $F = x'y' + xz$

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

$$F(x, y, z) = \Sigma(0, 1, 5, 7)$$

$$F(x, y, z) = \Pi(2, 3, 4, 6)$$

Canonical Forms

- Two canonical forms:
  - Sum-of-minterms
  - Product-of-maxterms
- Canonical forms are unique.

- Conversion between canonical forms is achieved by:
  - Exchanging Σ and Π
  - Listing all the missing indices

Standard Forms

- Two standard forms
  - Sum-of-products
  - Product-of-sums
- Standard forms are not unique.

- Sum-of-products is an OR expression with product terms that may have less literals than minterms

  Example:

$$F = xy + x'yz + xy'z$$

Standard Forms

- **Product-of-sums is an AND expression with sum terms that may have less literals than maxterms**

  Example:

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

- **Standard forms have fewer operators (literals) than canonical forms**

- **Standard forms can be derived from canonical forms by combining terms that differ in one variable (this is, terms at distance 1)**

  Example:

$$\begin{aligned} F_1 &= xyz + xyz' + xy'z + x'yz \\ &= xyz + xyz' + xyz + xy'z + xyz + x'yz \\ &= xy(z + z') + x(y + y')z + (x + x')yz \\ &= xy + xz + yz \end{aligned}$$

Slides by Philip Pham, University of California, Irvine

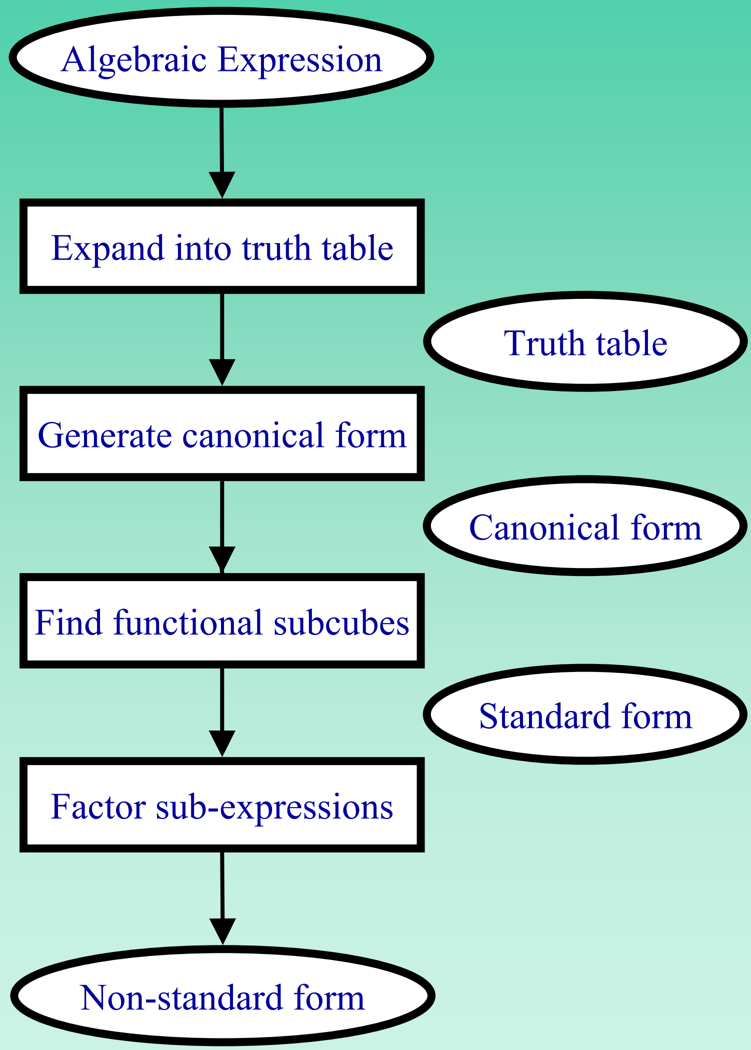
Non-standard Forms

- **Non-standard forms have fewer operators (literals) than standard forms.**
- **They are obtained by factoring variables.**

Example:

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

Strategy for Operator (Literal) Reduction in Boolean Expressions



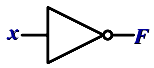
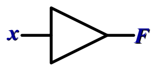
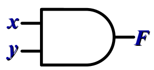
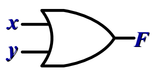
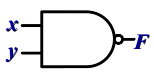
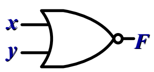
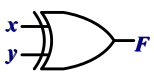
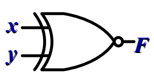
Binary Logic Operations

- There are 2^{2^n} Boolean functions for n binary variables
- Therefore, 16 Boolean functions for $n = 2$. They are

Name	Operator Symbol	Functional Values for x,y = 00	01	10	11	Algebraic Expression	Comment
Zero		0	0	0	0	$F_0 = 0$	Binary constant 0
AND	$x \cdot y$	0	0	0	1	$F_1 = xy$	x and y
Inhibition	x / y	0	0	1	0	$F_2 = xy'$	x but not y
Transfer		0	0	1	1	$F_3 = x$	x
Inhibition	y / x	0	1	0	0	$F_4 = x'y$	y but not x
Transfer		0	1	0	1	$F_5 = y$	y
XOR	$x \oplus y$	0	1	1	0	$F_6 = xy' + x'y$	x or y but not both
OR	$x + y$	0	1	1	1	$F_7 = x + y$	x or y
NOR	$x \downarrow y$	1	0	0	0	$F_8 = (x + y)'$	Not-OR
Equivalence	$x \odot y$	1	0	0	1	$F_9 = xy + x'y'$	x equals y
Complement	y'	1	0	1	0	$F_{10} = y'$	Not y
Implication	$x \subset y$	1	0	1	1	$F_{11} = x + y'$	If y, then x
Complement	x'	1	1	0	0	$F_{12} = x'$	Not x
Implication	$x \supset y$	1	1	0	1	$F_{13} = x' + y$	If x, then y
NAND	$x \uparrow y$	1	1	1	0	$F_{14} = (xy)'$	Not-AND
One		1	1	1	1	$F_{15} = 1$	Binary constant 1

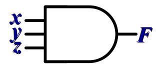
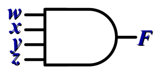
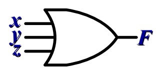
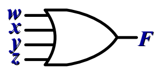
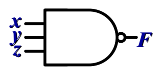
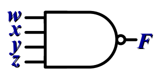
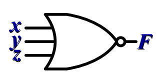
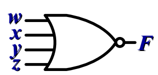
- There are two functions that generate constants: *Zero* and *One*. For every combination of variable values, the *Zero* function will return to 0, whereas the *One* function will return to 1.
- There are four functions of one variable, which indicate *Complement* and *Transfer* operations. Specifically, the *Complement* function will produce the complement of one of the binary variables. The *Transfer* functions by contrast will reproduce one of the binary variables at the output.
- There are ten functions that define eight specific binary operations: *AND*, Inhibition, *XOR*, *OR*, *NOR*, Equivalence, Implication, and *NAND*.

Digital Logic Gates

Name	Graphic Symbol	Functional Expression	Number of transistors	Delay in *ns*
Inverter		$F = x'$	**2**	**1**
Driver		$F = x$	**4**	**2**
AND		$F = xy$	**6**	**2.4**
OR		$F = x + y$	**6**	**2.4**
NAND		$F = (xy)'$	**4**	**1.4**
NOR		$F = (x + y)'$	**4**	**1.4**
XOR		$F = x \oplus y$	**14**	**4.2**
XNOR		$F = x \odot y$	**12**	**3.2**

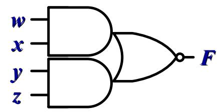
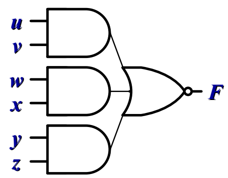
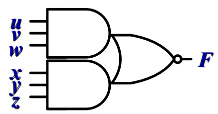
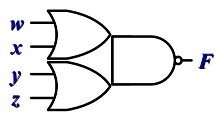
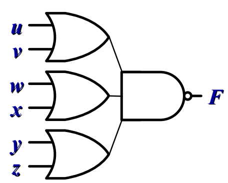
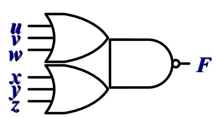
Basic Logic Library
(CMOS Technology Implementations)

Multiple-Input Gates

Name	Graphic Symbol	Functional Expression	Number of transistors	Delay in *ns*
3–input AND		$F = xyz$	**8**	**2.8**
4–input AND		$F = xyzw$	**10**	**3.2**
3–input OR		$F = x + y + z$	**8**	**2.8**
4–input OR		$F = x + y + z + w$	**10**	**3.2**
3–input NAND		$F = (xyz)'$	**6**	**1.8**
4–input NAND		$F = (xyzw)'$	**8**	**2.2**
3–input NOR		$F = (x + y + z)'$	**6**	**1.8**
4–input NOR		$F = (x + y + z + w)'$	**8**	**2.2**

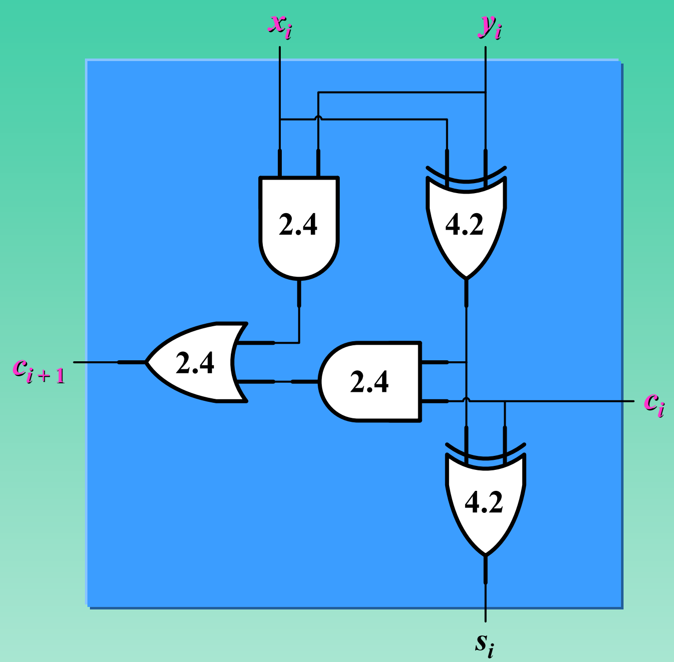
Multiple-Input Standard Logic Gates

Multiple-Operator (Complex) Gates

Name	Graphic Symbol	Functional Expression	Number of transistors	Delay in *ns*
2–wide, 2–input AOI		$F = (wx + yz)'$	**8**	**2.0**
3–wide, 2–input AOI		$F = (uv + wx + yz)'$	**12**	**2.4**
2–wide, 3–input AOI		$F = (uvw + xyz)'$	**12**	**2.2**
2–wide, 2–input OAI		$F = ((w + x)(y + z))'$	**8**	**2.0**
3–wide, 2–input OAI		$F = ((u + v)(w + x)(y + z))'$	**12**	**2.2**
2–wide, 3–input OAI		$F = ((u + v + w)(x + y + z))'$	**12**	**2.4**

Multiple-Operator Standard Logic Gates

Full-adder Design Using XOR Gates



Logic Schematic (46 Transistors)

x_i	y_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Truth table

Input/Output Path	Delay (ns)
c_i to c_{i+1}	4.8 ns
c_i to s_i	4.2 ns
x_i, y_i to c_{i+1}	9.0 ns
x_i, y_i to s_i	8.4 ns

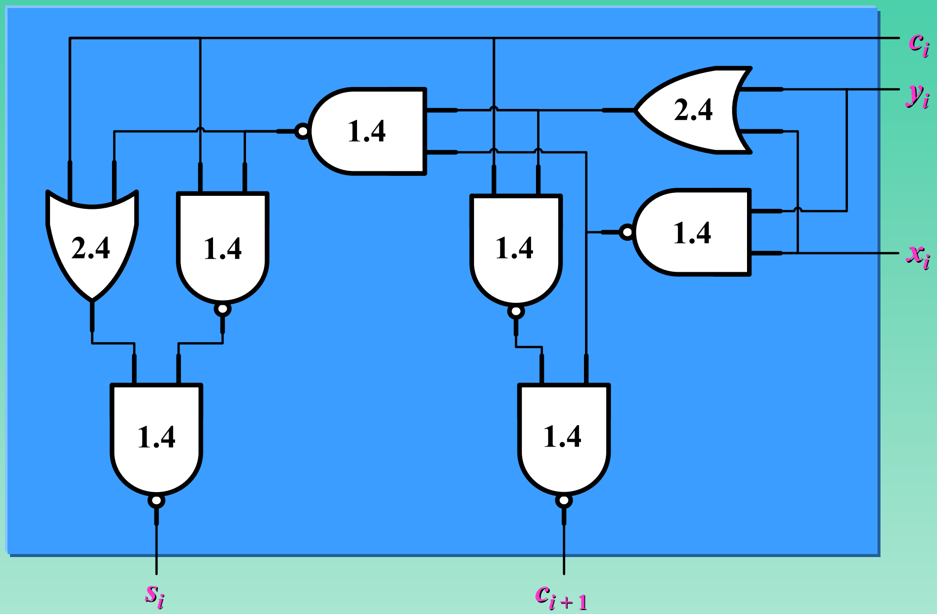
Full–adder delays

$$
\begin{aligned}
s_i &= x_i'y_i'c_i + x_i'y_ic_i' + x_iy_i'c_i' + x_iy_ic_i \\
&= (x_i'y_i + x_iy_i')c_i' + (x_i'y_i' + x_iy_i)c_i \\
&= (x_i \oplus y_i)c_i' + (x_i \odot y_i)c_i \\
&= (x_i \oplus y_i)c_i' + (x_i \oplus y_i)'c_i \\
&= (x_i \oplus y_i) \oplus c_i
\end{aligned}
$$

$$
\begin{aligned}
c_{i+1} &= x_iy_ic_i' + x_iy_ic_i + x_i'y_ic_i + x_iy_i'c_i \\
&= x_iy_i(c_i' + c_i) + c_i(x_i'y_i + x_iy_i') \\
&= x_iy_i + c_i(x_i \oplus y_i)
\end{aligned}
$$

Full–adder equation

Full-adder Design Using Fast Gates



Logic Schematic (36 Transistors)

x_i	y_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Truth table

Input/Output Path	Delay (ns)
c_i to c_{i+1}	2.8 ns
c_i to s_i	3.8 ns
x_i, y_i to c_{i+1}	5.2 ns
x_i, y_i to s_i	7.6 ns

Full–adder delays

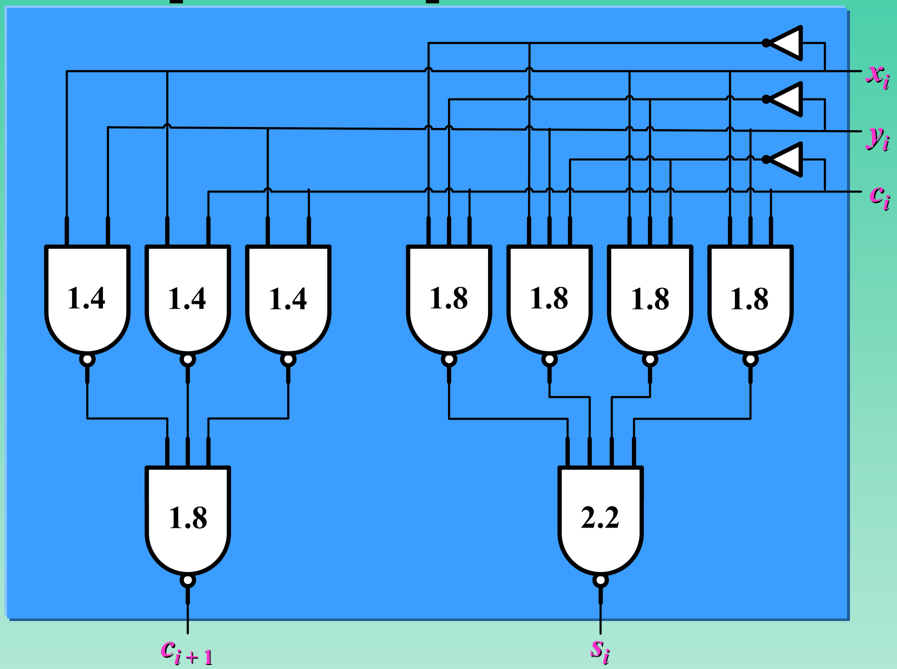
$$x_i \odot y_i = x_i y_i + x_i' y_i' = ((x_i y_i)'(x_i' y_i')')' = ((x_i y_i)'(x_i + y_i))'$$

$$s_i = (x_i \oplus y_i)c_i' + (x_i \odot y_i)c_i = (x_i \odot y_i)'c_i' + (x_i \odot y_i)c_i = (x_i \odot y_i) \odot c_i$$

$$c_{i+1} = x_i y_i + c_i(x_i + y_i) = ((x_i y_i)'(c_i(x_i + y_i))')'$$

Full–adder equation

Full-adder Design with Multiple-input Gates



Logic Schematic (56 Transistors)

x_i	y_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Truth table

Input/Output Path	Delay (ns)
c_i to c_{i+1}	3.2 ns
c_i to s_i	5.0 ns
x_i, y_i to c_{i+1}	4.2 ns
x_i, y_i to s_i	5.0 ns

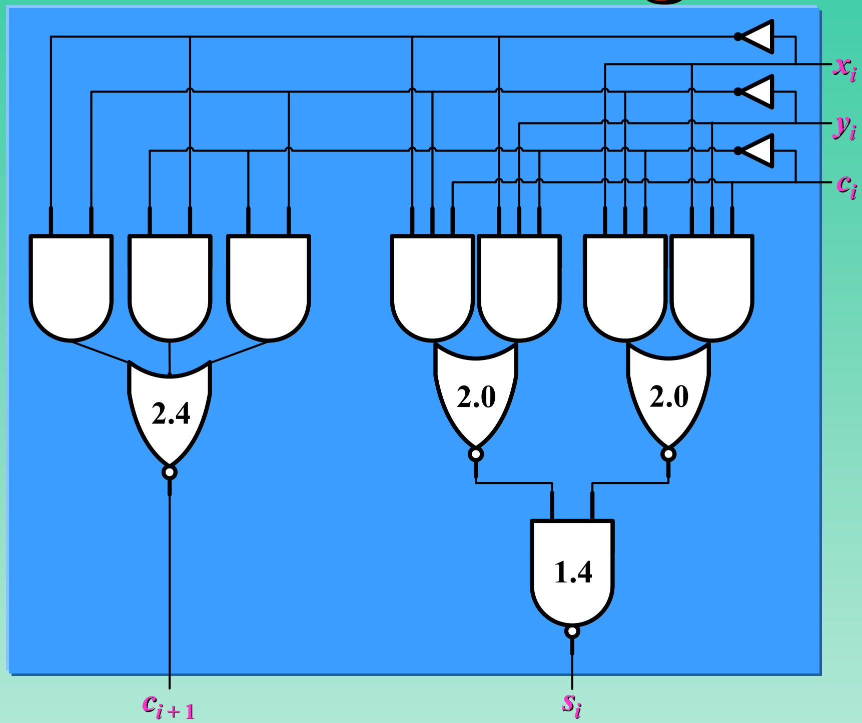
Full–adder delays

$$s_i = x_i'y_i'c_i + x_i'y_ic_i' + x_iy_i'c_i' + x_iy_ic_i$$
$$= ((x_i'y_i'c_i)'(x_i'y_ic_i')'(x_iy_i'c_i')'(x_iy_ic_i))'$$

$$c_{i+1} = x_iy_i + c_ix_i + c_iy_i$$
$$= ((x_iy_i)'(c_ix_i)'(c_iy_i)')'$$

Full–adder equation

Full-adder Design with Complex Gates



Logic Schematic (46 Transistors)

x_i	y_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Truth table

Input/Output Path	Delay (ns)
c_i to c_{i+1}	3.4 ns
c_i to s_i	4.4 ns
x_i, y_i to c_{i+1}	3.4 ns
x_i, y_i to s_i	4.4 ns

Full–adder delays

$$s_i = x_i'y_i'c_i + x_i'y_ic_i' + x_iy_i'c_i' + x_iy_ic_i$$
$$= ((x_i'y_i'c_i + x_i'y_ic_i)'(x_iy_i'c_i' + x_iy_ic_i)')'$$

$$c_{i+1} = x_iy_i + c_ix_i + c_iy_i$$
$$= ((x_iy_i)'(c_ix_i)'(c_iy_i)')'$$
$$= ((x_i'+y_i')(c_i'+x_i')(c_i'+y_i'))'$$
$$= (x_i'y_i' + c_i'x_i' + c_i'y_i')'$$

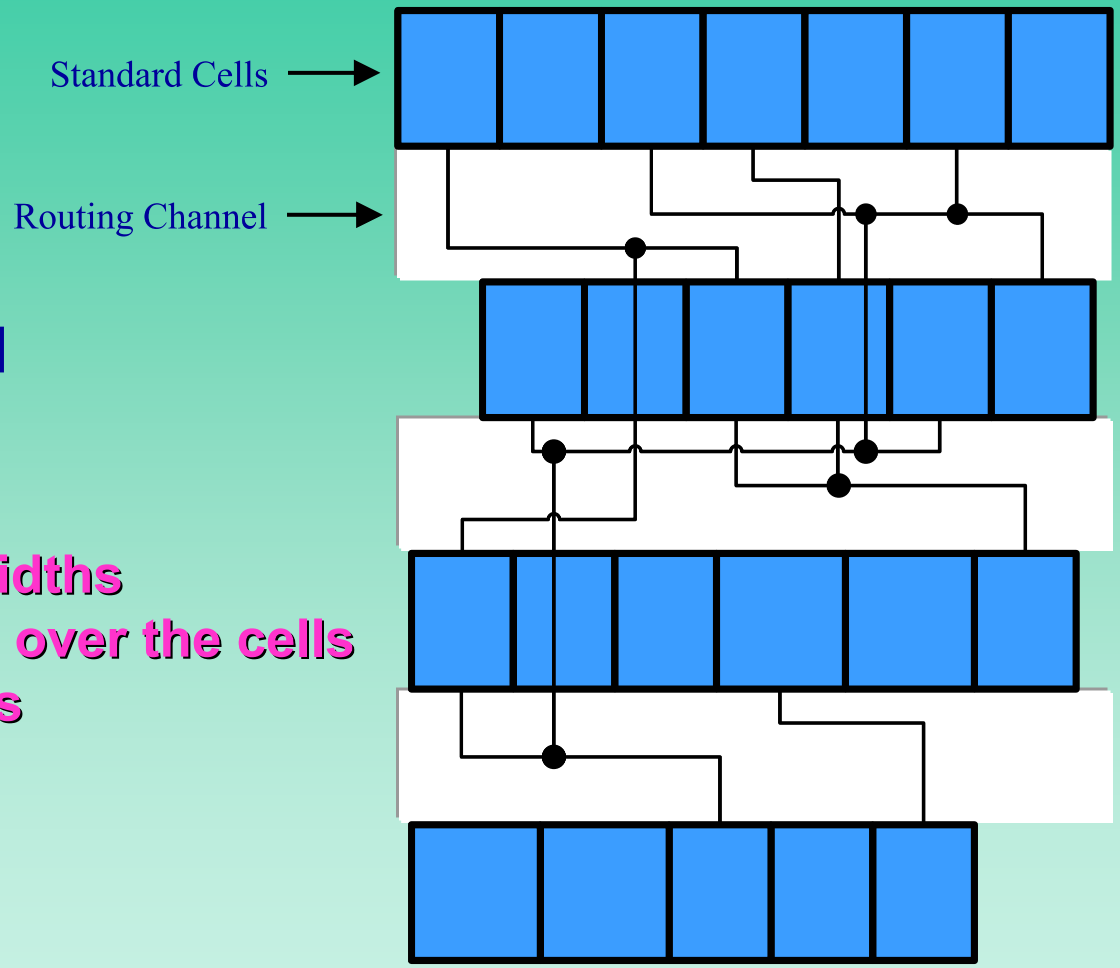
Full–adder equation

VLSI Technology

- **Small-scale integration (SSI)**
  - **10 gates/package**
- **Medium-scale integration (MSI)**
  - **10 – 100 gates/package (2 – 4 bit slices)**
- **Large-scale integration (LSI)**
  - **100 – 1000 gates/package (controllers, datapaths, bit slices)**
- **Very-large-scale integration (VLSI)**
  - **1000+ gates/package (systems on a chip)**
    - ***Custom designs (Standard cells)***
    - ***Gate arrays (GAs)***
    - ***Field-programmable (FPGAs)***

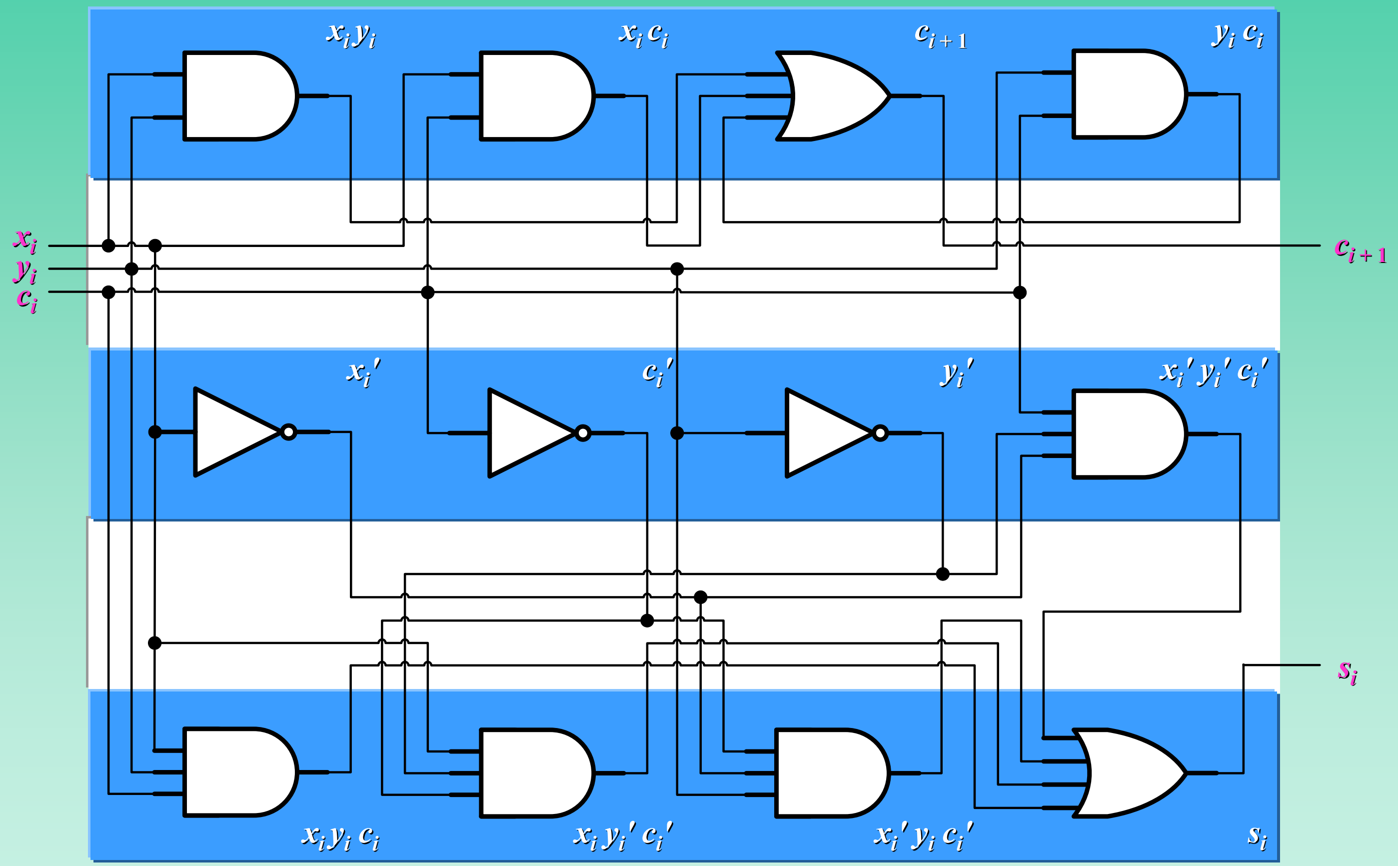
Custom Design

- **Each designed by hand**
- **Standard cells**
  - Same height, different widths
  - Routing in channels and over the cells
  - Two or more metal layers



Standard Cell Approach

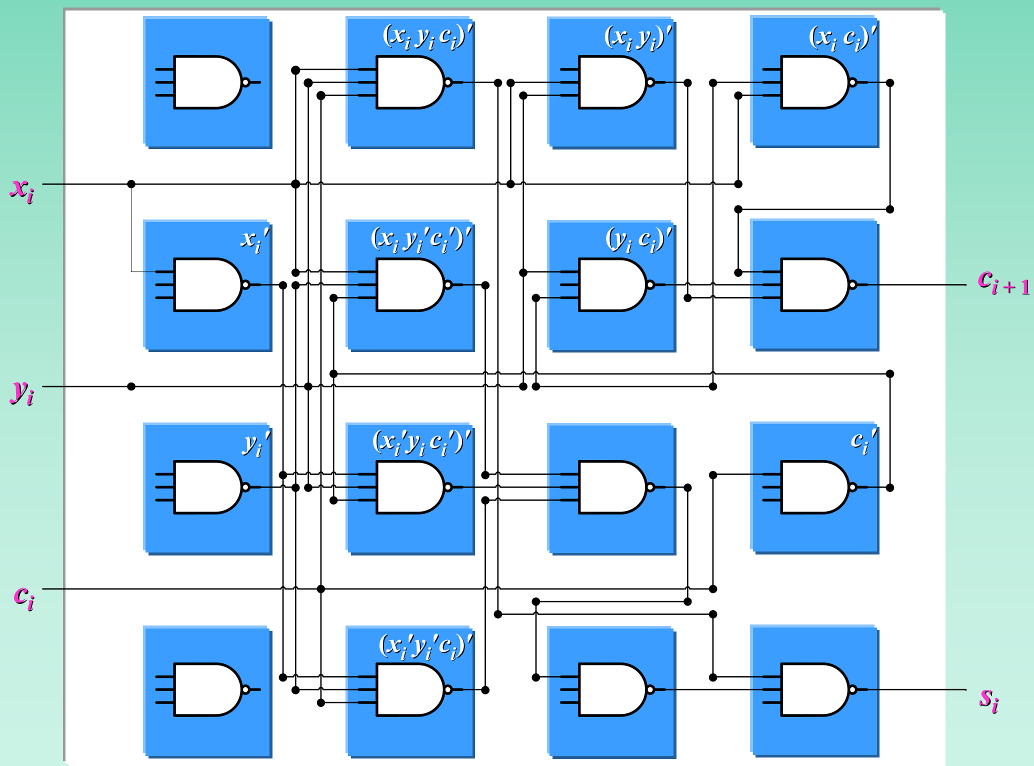
Example of Custom Design



Full-Adder Implementation with Standard Cells

Semi-Custom Approach with Gate Arrays

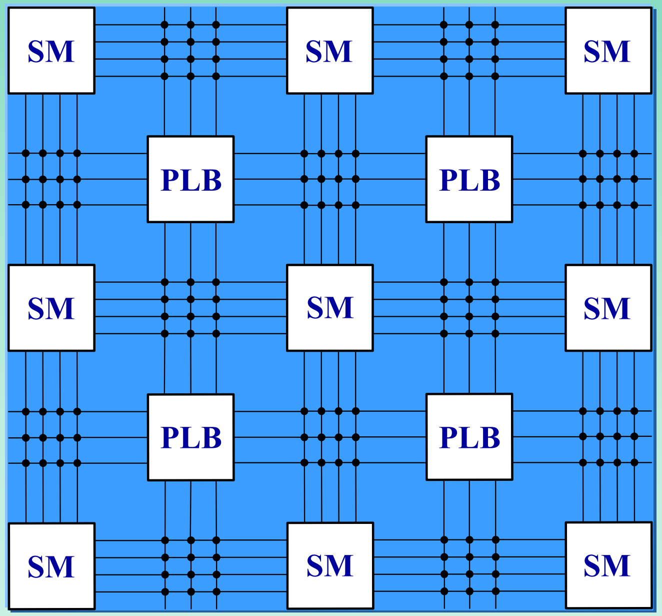
- Gate arrays are prefabricated arrays of interconnected gates
- All gates are the same type (3–input NAND, for example)
- Two or more metal layers used to connect gates



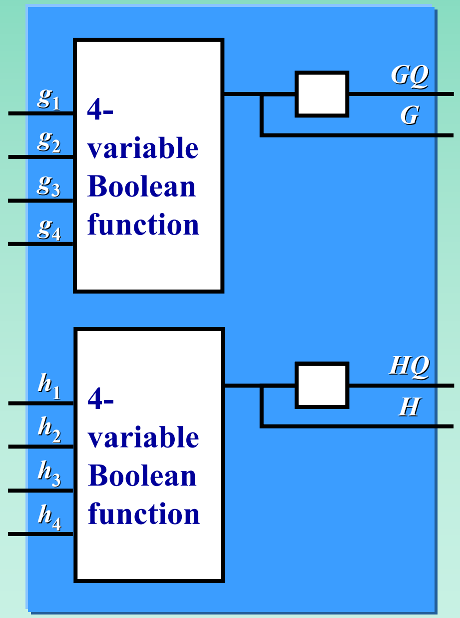
Full-Adder Implementation in a Gate Array

Field–Programmable Approach with FPGA

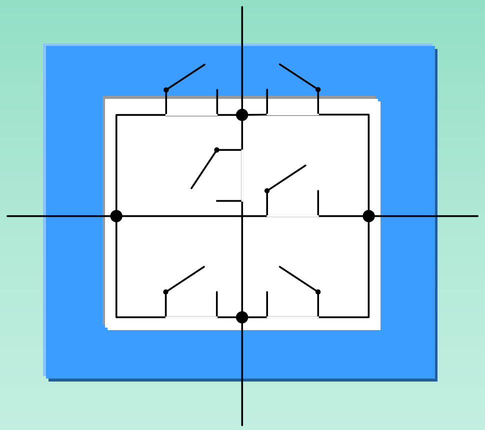
- FPGAs are programmed by loading data into internal memory
- Excellent for rapid prototyping
- Low density and low speed



Array Structure



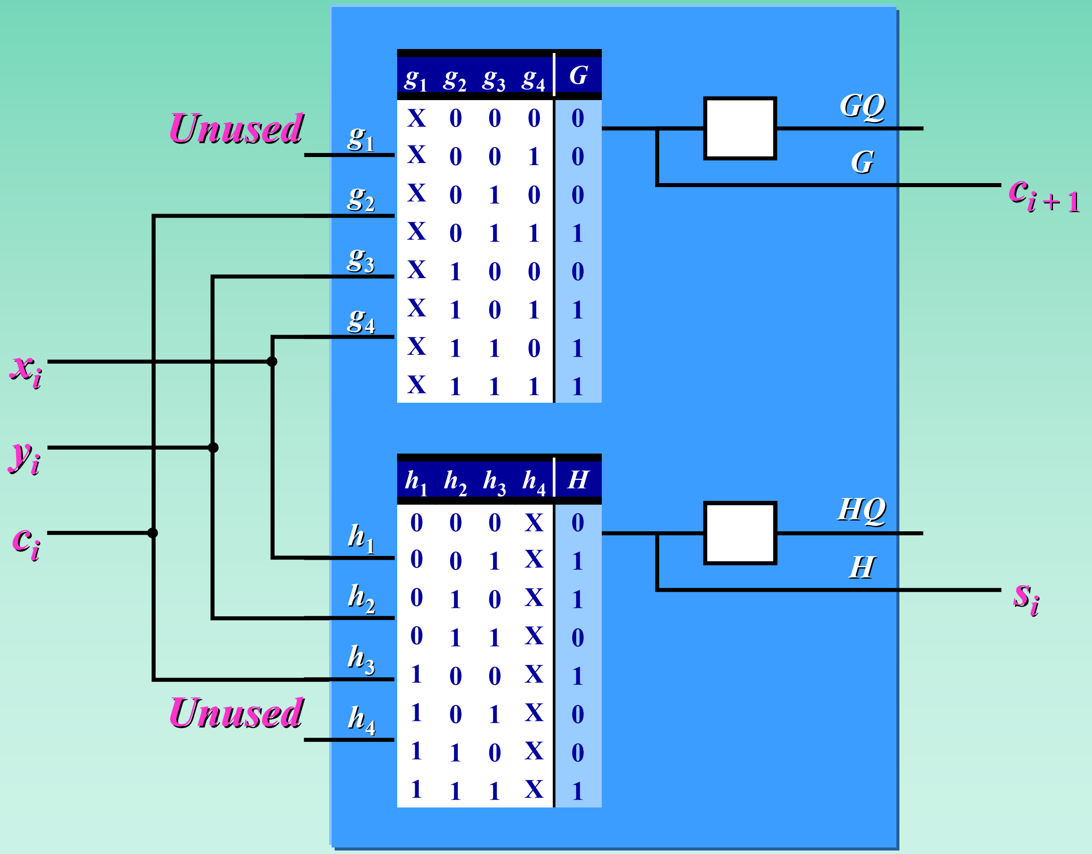
Programmable Logic Blocks (PLB)



Interconnect Point (IP)

Full-adder Implemented with FPGA

- 1 programmable logic block for each full-adder
- 3 out of 4 inputs are used for each Boolean function



g_1	g_2	g_3	g_4	G
X	0	0	0	0
X	0	0	1	0
X	0	1	0	0
X	0	1	1	1
X	1	0	0	0
X	1	0	1	1
X	1	1	0	1
X	1	1	1	1

h_1	h_2	h_3	h_4	H
0	0	0	X	0
0	0	1	X	1
0	1	0	X	1
0	1	1	X	0
1	0	0	X	1
1	0	1	X	0
1	1	0	X	0
1	1	1	X	1

Slides by Philip Pham, University of California, Irvine

Chapter Summary

- Boolean Algebra
  - Axioms
  - Basic theorems
- Boolean Functions
- Specification of Boolean Functions
  - Truth tables
  - Algebraic expressions
    - *Canonical forms*
    - *Standard forms*
    - *Non-standard forms*
- Algebraic Manipulation of Boolean Expressions
- Logic Gates
  - Simple gates
  - Multiple-input gates
  - Complex gates
- Implementation Technology
  - SSI (Small-scale integration)
  - MSI (Medium-scale integration)
  - LSI (Large-scale integration)
  - VLSI (Very-large-scale integration)
    - *Custom designs*
    - *Semi-custom designs*
    - *Field-programmable*