



Chapter 3 Boolean Algebra and Logic Gates

Algebraic Properties

- ☞ A **set** is a collection of objects with a common property.
- ☞ A **binary operator** on set S is a **rule** that assigns to each pair of elements in S another unique element in S .
- ☞ The **axioms (postulates)** of an **algebra** are the basic assumptions from which all theorems of the algebra can be proved.
- ☞ It is assumed that there is an **equivalence relation** ($=$), which satisfies the **principle of substitution**.
 - ⇒ It is **reflexive**, **symmetric**, and **transitive**.
- ☞ Most common axioms used to formulate an algebra structure (e.g., field):
 - ☆ **Closure**: a set S is closed with respect to \bullet if and only if $\forall x, y \in S, (x \bullet y) \in S$.
 - ☆ **Associativity**: a binary operator \bullet on S is associative if and only if $\forall x, y, z \in S, (x \bullet y) \bullet z = x \bullet (y \bullet z)$.
 - ☆ **Identity element**: a set S has an identity element with respect to \bullet if and only if $\exists e \in S$ such that $\forall x \in S, e \bullet x = x \bullet e = x$.
 - ☆ **Commutativity**: a binary operator \bullet defined on S is commutative if and only if $\forall x, y \in S, x \bullet y = y \bullet x$.
 - ☆ **Inverse element**: a set S having the identity element e with respect to \bullet has an inverse if and only if $\forall x \in S, \exists y \in S$ such that $x \bullet y = e$.
 - ☆ **Distributivity**: if \bullet and \square are binary operators on S , \bullet is distributive over \square if and only if $\forall x, y, z \in S, x \bullet (y \square z) = (x \bullet y) \square (x \bullet z)$.

Axiomatic Definition of Boolean Algebra

- ☞ **Boolean algebra**: an algebraic system of logic introduced by George Boole in 1854.
 - An Investigation of Laws of Thought on which are Founded the Mathematical Theories of Logic and Probabilities, Macmillan (Dover, 1958)
- ☞ **Switching algebra**: a 2-valued Boolean algebra introduced by Claude Shannon in 1938.
 - A Symbolic Analysis of Relay and Switching Circuits, MS Thesis, MIT
- ☞ **Huntington postulates** for Boolean algebra: defined on a set B with binary operators $+$ & \cdot , and the equivalence relation $=$ (Edward Huntington, 1904):
 - ① (a) Closure with respect to $+$. (b) Closure with respect to \cdot .
 - ② (a) Identity element 0 with respect to $+$. (b) Identity element 1 with respect to \cdot .
 - ③ (a) Commutative with respect to $+$. (b) Commutative with respect to \cdot .
 - ④ (a) \cdot is distributive over $+$. (b) $+$ is distributive over \cdot .
 - ⑤ $\forall x \in B, \exists x' \in B$ (called the **complement** of x) such that (a) $x + x' = 1$ and (b) $x \cdot x' = 0$.
 - ⑥ There are *at least* 2 distinct elements in B .
- ☞ The axioms are *independent*; none can be proved from the others.
- ☞ *Associativity* is not included, since it can be derived (for both operators) from the given axioms.
- ☞ In ordinary algebra, $+$ is not distributive over \cdot .
- ☞ No additive or multiplicative inverses; no subtraction or division operations.
- ☞ Complement is not available in ordinary algebra.
- ☞ B is as yet undefined. It is to be defined as the set $\{0,1\}$ (**two-valued Boolean algebra**).
- ☞ In ordinary algebra, the set S can contain an infinite number of elements (e.g., all integers).

Exercise 1

Prove the associative law: $a + (b + c) = (a + b) + c$ and $a(bc) = (ab)c$. \square

★ Boolean algebra


- ☆ Set B of at least 2 elements (*not* variables).
- ☆ Rules of operation for the 2 binary operators (+ & \cdot).
- ☆ Huntington postulates satisfied by the elements of B and the operators.

★ Two-valued Boolean algebra (switching algebra)

- ☆ $B \equiv \{0, 1\}$.
- ☆ The binary operations are defined as the logical AND (\cdot) and OR (+). For convenience, a unary operation NOT (complement) is also included when we define the basic operations.
- ☆ The Huntington postulates can be shown valid (read pp. 66-68).
- ☆ Unless otherwise noted, we will use the term *Boolean algebra* for 2-valued Boolean algebra.

Basic Theorems of Boolean Algebra**Theorem 1 (Idempotency)**

(a) $x + x = x$; (b) $x \cdot x = x$.

 Thm. 1(b) is the **dual** of Thm. 1(a), and vice versa.

Theorem 2

(a) $x + 1 = 1$; (b) $x \cdot 0 = 0$.

Theorem 3 (Absorption)

(a) $yx + x = x$; (b) $(y + x)x = x$.

Theorem 4 (Involution)

$(x')' = x$.

Theorem 5 (Associativity)

(a) $x + (y + z) = (x + y) + z$; (b) $x(yz) = (xy)z$.

Theorem 6 (De Morgan)

(a) $(x + y)' = x' \cdot y'$; (b) $(xy)' = x' + y'$.

☞ **Duality principle:** every algebraic expression deducible from the axioms of Boolean algebra remains valid if $+ \leftrightarrow \cdot$ and $1 \leftrightarrow 0$

Theorem 7

If $E(x_1, x_2, \dots, x_n)$ is a Boolean expression of n variables, and $E_d(x_1, x_2, \dots, x_n)$ is its dual expression, then

$$E'(x_1, x_2, \dots, x_n) = E_d(x'_1, x'_2, \dots, x'_n)$$

Theorem 8 (De Morgan (generalized))

$$(x_1 + x_2 + \dots + x_n)' = x'_1 x'_2 \dots x'_n$$

$$(x_1 x_2 \dots x_n)' = x'_1 + x'_2 + \dots + x'_n$$

☞ The theorems usually are proved algebraically (i.e., by transformations based on axioms and theorems) or by truth table.

Exercise 2

Show that the *set theory* is an example of (multi-element) Boolean algebra. □

Boolean Functions

A **boolean function** is an *algebraic expression* formed with boolean variables, the operators OR, AND, and NOT, parentheses, and an equal sign.

- ☞ When evaluating a boolean function, we must follow a specific order of computation: (1) parenthesis, (2) NOT, (3) AND, and (4) OR.
- ☞ Any boolean function can be represented by a truth table. The number of rows in the table is 2^n , where n is the number of variables in the function.
- ☞ There are infinitely many algebraic expressions that specify a given boolean function. It is important to find the simplest one.
- ☞ Any boolean function can be transformed in a straightforward manner from an algebraic expression into a logic diagram composed only of AND, OR, and NOT gates.

Example 1

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

Row number	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

□

☞ A **literal** is a **variable** or its complement in a boolean expression, e.g., F_1 has 8 literals, 1 OR term (**sum term**), and 3 AND terms (**product terms**).

☞ The complement of any function F is F' , which can be obtained by De Morgan's theorem: 1) take the dual of F , and 2) complement each literal in F .

Example 2

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

□

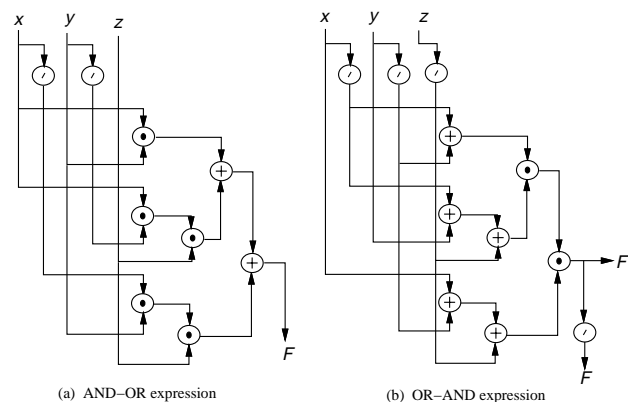


Figure 1: Graphic representation of two expressions for F_1 [Gajski].

Exercise 3

Given n variables, how many different boolean functions can we define? \square

Algebraic Manipulation

- \rightarrow Minimization of the number of literals and the number of terms usually results in a simpler circuit (less expensive).
- \rightarrow The number of literals can be minimized by algebraic manipulation. Unfortunately, there are no specific rules to follow that will guarantee the optimal result.
- \rightarrow CAD tools for logic minimization are commonly used today.

Some useful rules:

$$\textcircled{1} \quad x + x'y = x + y$$

$$\textcircled{2} \quad x(x' + y) = xy$$

$$\textcircled{3} \quad xy + yz + x'z = xy + x'z \quad (\text{the Consensus Theorem I})$$

$$\textcircled{4} \quad (x + y)(y + z)(x' + z) = (x + y)(x' + z) \quad (\text{the Consensus Theorem II})$$

Example 3

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

The literal count is reduced from 8 to 6. \square

Canonical Forms

☞ 2 variables \Rightarrow 4 combinations (x_1x_2 , $x_1x'_2$, x'_1x_2 , and $x'_1x'_2$).

☞ n variables $\Rightarrow 2^n$ combinations, each is called a **minterm** (or a **standard product**), denoted as m_i , $0 \leq i \leq 2^n - 1$. Their complements are called the **maxterms** (or **standard sums**), denoted as M_i , $0 \leq i \leq 2^n - 1$.

☞ **Canonical forms:**

☆ **Sum-of-minterms** (som)

☆ **Product-of-maxterms** (pom)

☞ Each maxterm is the complement of its corresponding minterm: $m'_i = M_i$.

	x	y	z	Minterms	Notation	Maxterms	Notation
0	0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
1	0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
2	0	1	0	$x'y'z'$	m_2	$x + y' + z$	M_2
3	0	1	1	$x'y'z$	m_3	$x + y' + z'$	M_3
4	1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
5	1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
6	1	1	0	xyz'	m_6	$x' + y' + z$	M_6
7	1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Example 4

Consider the two functions F_2 and F_1 as shown in the following table.

	x	y	z	F_2	F'_2	F_1	F'_1
0	0	0	0	0	1	0	1
1	0	0	1	1	0	0	1
2	0	1	0	0	1	0	1
3	0	1	1	0	1	1	0
4	1	0	0	1	0	0	1
5	1	0	1	0	1	1	0
6	1	1	0	0	1	1	0
7	1	1	1	1	0	1	0

Sum-of-minterms:

$$F_2 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7 \equiv \sum(1, 4, 7)$$

$$F_1 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7 \equiv \sum(3, 5, 6, 7)$$

Product-of-maxterms:

$$\begin{aligned} F_2 &= (x + y + z)(x + y' + z)(x + y' + z')(x' + y + z')(x' + y' + z) \\ &= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6 \equiv \prod(0, 2, 3, 5, 6) \end{aligned}$$

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

Any function can be represented by either of these 2 canonical forms. □

- ☞ To convert from one canonical form to another, interchange \sum and \prod , and list the numbers that were excluded from the original form.
- ☞ $F_1 = \sum(3, 5, 6, 7)$ is the sum of **1-minterms** for F_1
- ☞ $F_1' = \sum(0, 1, 2, 4)$ is the sum of **0-minterms** for F_1 .
- ☞ How do we convert, e.g., $F = x + yz$, into the canonical forms?
 - ⇒ By truth table.
 - ⇒ By expanding the missing variables in each term, using $1 = x + x'$ and $0 = x \cdot x'$.

Exercise 4

Convert $F = x'y' + xz$ to its canonical (som & pom) forms. □

Standard Forms

- ☞ **Standard forms:**
 - ☆ **Sum-of-products (sop)**
 - ☆ **Product-of-sums (pos)**
- ☞ **Product terms** (or **implicants**) are the **AND terms**, which can have fewer literals than the minterms.

- ☞ **Sum terms** are the **OR terms**, which can have fewer literals than the max-terms.
- ☞ Standard forms are not unique!
- ☞ Standard forms can be derived from canonical forms by combining terms that differ in one variable, i.e., terms with distance 1.
- ☞ Each product term in the **reduced** sop form (i.e., no more reduction is possible) is called a **prime implicant** (PI).
 - Each prime implicant represents 1 or more 1-minterms.
 - Each 1-minterm can be included in several prime implicants.
 - If there is a 1-minterm included in only 1 prime implicant, then the prime implicant is an **essential prime implicant** (EPI).
- ☞ Each sum term in the **reduced** pos form is called a **prime implicate**.

Example 5

(a) $F_1 = xy + xy'z + x'y'z$ is in sop form, and $F'_1 = (x' + y')(x' + y + z')(x + y' + z')$ is in pos form.

(b) Consider the boolean function $F = (wx + yz)(w'x' + y'z')$ in nonstandard form.

★ Sop form: $F = w'x'yz + wxy'z'$.

★ Pos form: $F = (w + x')(w' + y')(y + z')(z + x)$. □

- ☞ Nonstandard forms can have fewer literals than standard forms.

Example 6

(a) $xy + xy'z + xy'w = x(y + y'z + y'w) = x(y + y'(z + w))$

(b) $F_1 = xy + xz + yz = xy + (x + y)z = x(y + z) + yz = xz + y(x + z)$ □

Other Logic Operations

- ☞ There are 2^{2^n} different boolean functions for n binary variables.
- ☞ There are 16 different boolean functions if $n = 2$.

Table 1: Boolean functions of two variables.

Name	Operator	Values xy				Algebraic	Comment
	Symbol	00	01	10	11	Expression	
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


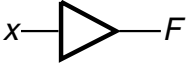






- ☞ There are 2 functions that generate constants, i.e., the *Zero* and *One* functions.
- ☞ There are 4 functions of one variable which indicate *Complement* and *Transfer* operations.
- ☞ There are 10 functions that define 8 specific binary operations: AND, Inhibition, XOR (exclusive-OR), OR, NOR, Equivalence, Implication and NAND.
- ☞ Apart from AND, OR, and NOT (complement), the following functions also are frequently used: NAND, NOR, XOR, XNOR (exclusive-NOR, or equivalence), & Transfer.





Exercise 5

Show that Inhibition and Implication are neither commutative nor associative; and NAND and NOR are commutative but not associative. Are XOR and XNOR commutative? Are they associative? \square

Digital Logic Gates

Table 2: Basic logic library in CMOS technology [Gajski].

Name	Graphic symbol	Function	No. transistors	Gate delay (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

-  You have to memorize the graphic symbols.
-  The number of transistors represent the hardware cost. The numbers in this table are based on the typical complementary metal-oxide semiconductor (CMOS) implementation.
-  The gate delay represents the performance. The numbers in this table are also based on the typical CMOS implementation.
-  We would like to maximize the performance and minimize the cost.

Example 7

Consider the **full adder** as defined in the following truth table.

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

We first derive expressions for the two output functions that contain a minimum number of operators.

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

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

Note that the carry function c_{i+1} could be reduced to $x_i y_i + c_i (x_i + y_i)$.

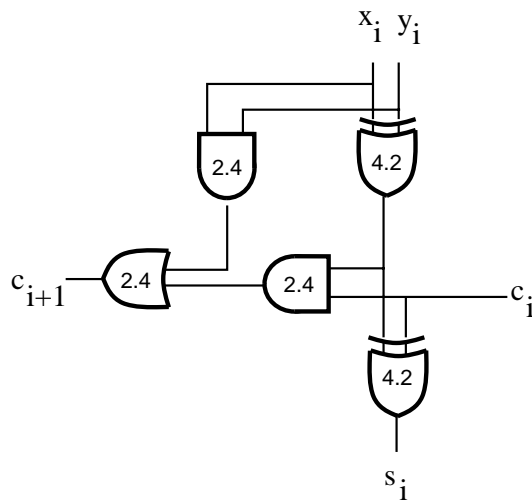
$$\begin{aligned}
 c_{i+1} &= x_i y_i + c_i (x_i + y_i) \\
 &= ((x_i y_i)' (c_i (x_i + y_i)))'
 \end{aligned}$$

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

We can implement $x_i \odot y_i$ with NAND and NOR gates.

$$\begin{aligned}
 x_i \odot y_i &= x_i y_i + x_i' y_i' \\
 &= ((x_i y_i)' (x_i + y_i))'
 \end{aligned}$$

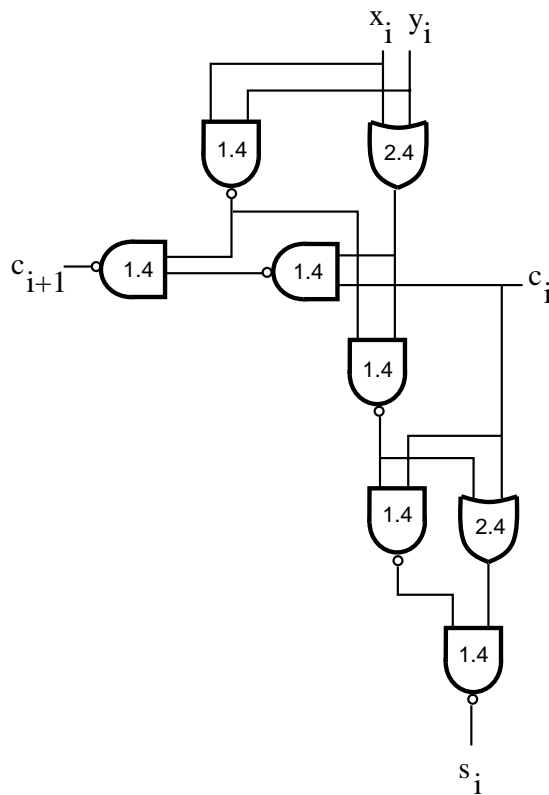
The gate-level implementations for the full adder are shown in Fig. 2. □



(a) Design with minimum number of operators

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

(b) Input-output delays for the design in (a)


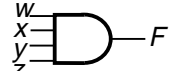

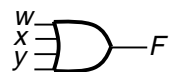
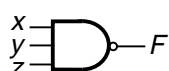








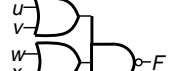


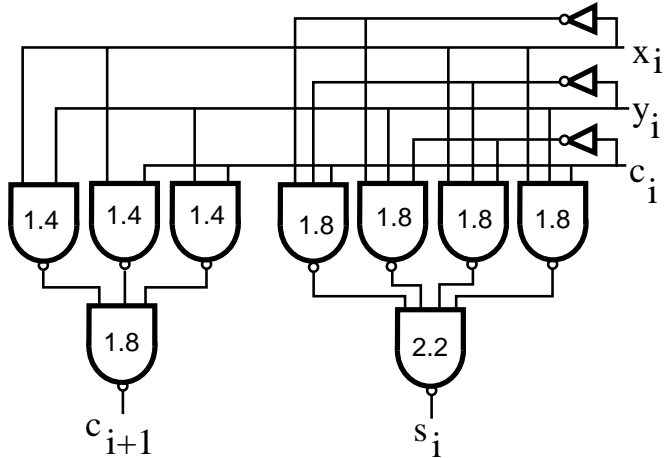
(c) Design with NANDs and ORs

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	7.2 ns

(d) Input-output delays for the design in (c)

Figure 2: Full adder design [Gajski].

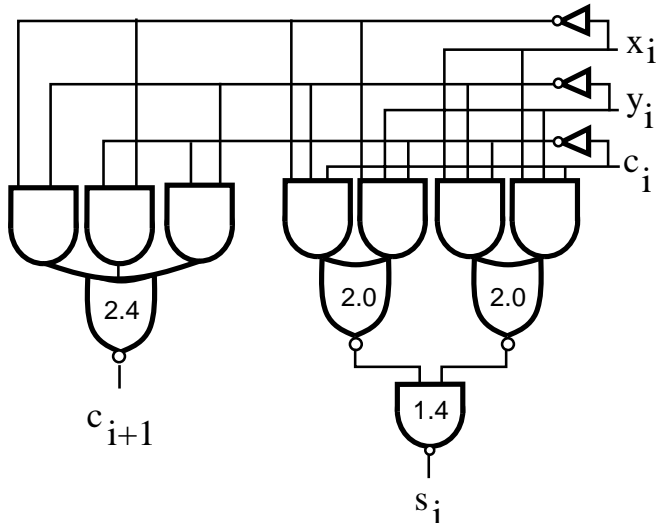
Name	Graphic symbol	Function	No. trs	Delay (ns)
3-input AND		$F = xyz$	8	2.8
4-input AND		$F = xyzw$	10	3.2
3-input OR		$F = x + y + z$	8	2.8
4-input OR		$F = x + y + z + w$	10	3.2
3-input NAND		$F = (xyz)'$	8	1.8
4-input NAND		$F = (xyzw)'$	10	2.2
3-input NOR		$F = (x + y + z)'$	8	1.8
4-input NOR		$F = (x + y + z + w)'$	10	2.2
2-wide,2-inAOI		$F = (wx + yz)'$	8	2.0
3-wide,2-inAOI		$F = (uv + wx + yz)'$	12	2.4
2-wide,3-inAOI		$F = (uvw + xyz)'$	12	2.2
2-wide,2-inOAI		$F = ((w + x)(y + z))'$	8	2.0
3-wide,2-inOAI		$F = ((u + v)(w + x)(y + z))'$	12	2.2
2-wide,3-inOAI		$F = ((u + v + w)(x + y + z))'$	12	2.4



(a) Design with multiple-input gates

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}	3.2 ns
x_i, y_i to s_i	5.0 ns

(b) Input-output delays for the design in (a)



(a) Design with multiple-operator gates

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

(b) Input-output delays for the design in (a)

Figure 3: Full adder design using multiple-input and multiple-operator gates [Gajski].

Gate Implementations

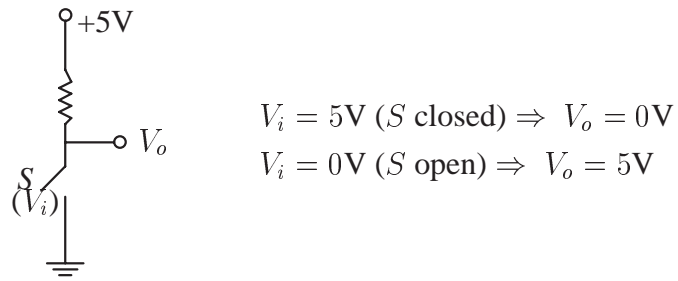


Figure 4: Inverter and binary logic.

Positive and negative logic:

Logic type	High (3.3V)	Low (0V)
Positive logic	1	0
Negative logic	0	1

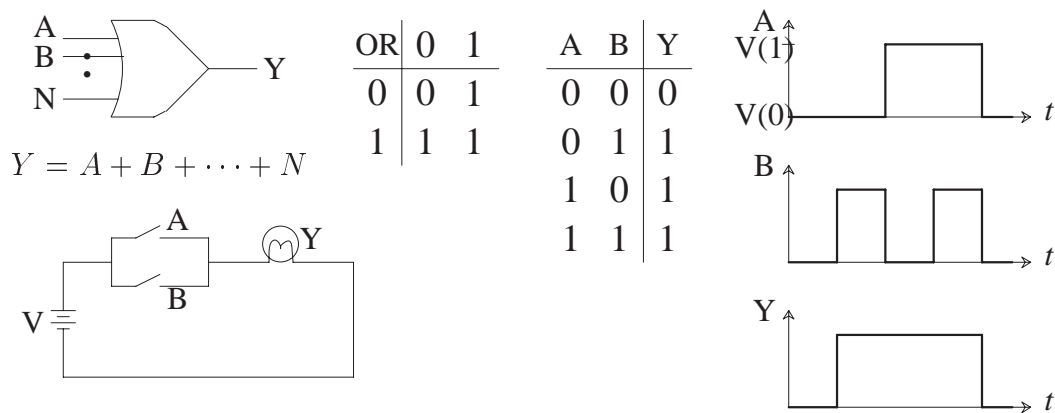


Figure 5: OR gate.

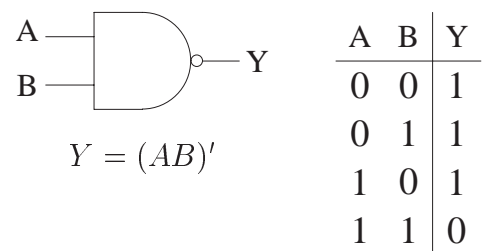
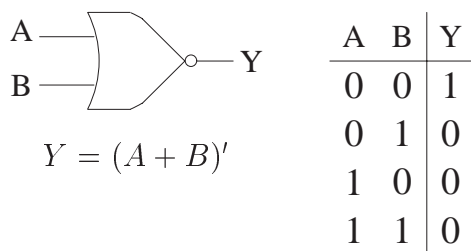
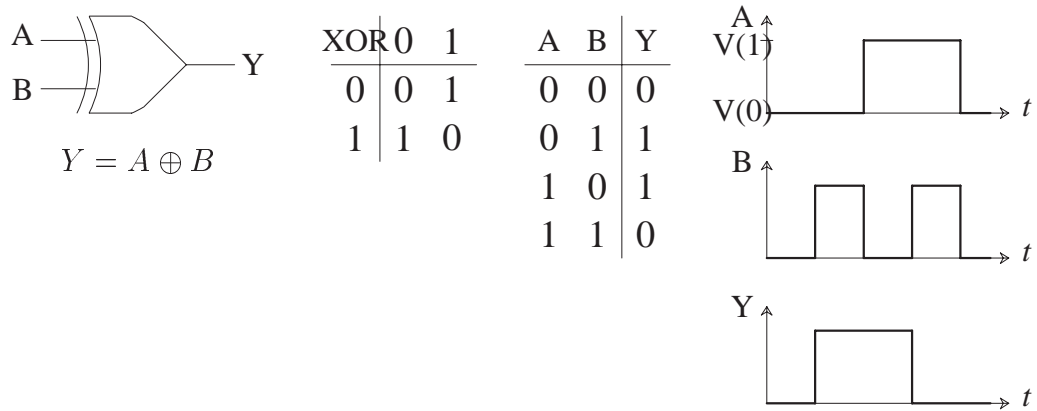
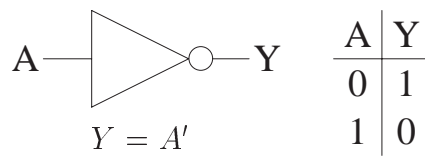
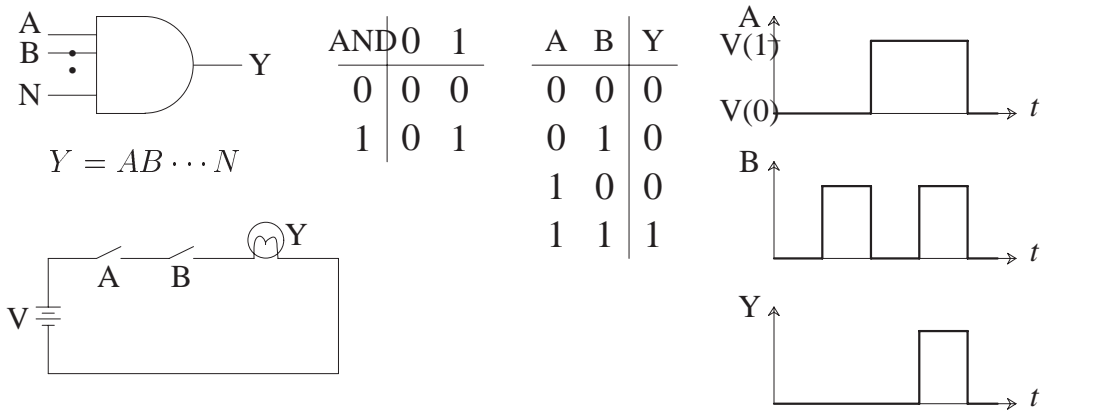


Figure 6: (a) AND gate; (b) NOT gate (Inverter); (c) XOR gate; (d) NOR and NAND gates.

👉 Negative-logic OR \equiv positive-logic AND.

👉 Negative-logic AND \equiv positive-logic OR.

Table 3: Examples of positive and negative logic symbols [Gajski].

Positive logic	Negative logic

★ **Noise:** undesirable voltage variations that are superimposed on the normal voltage levels.

★ **Noise margin:** the maximum noise voltage level that can be tolerated by the gate.

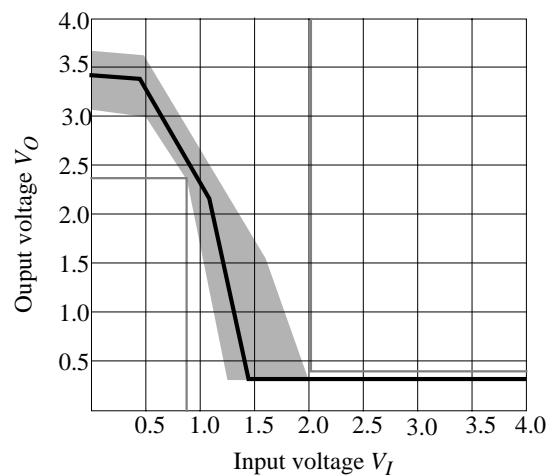


Figure 7: Typical I/O characteristics for an inverter [Gajski].

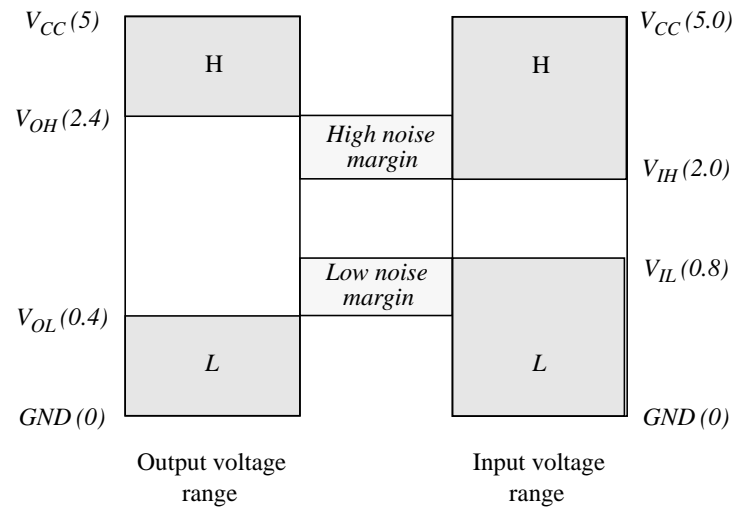


Figure 8: High and low noise margins [Gajski].

- ★ **Fan-out** (standard load): the number of gates that each gate can drive, while providing voltage levels in the guaranteed range.
- ★ The fan-out depends on the amount of current a gate can source or sink while driving other gates.

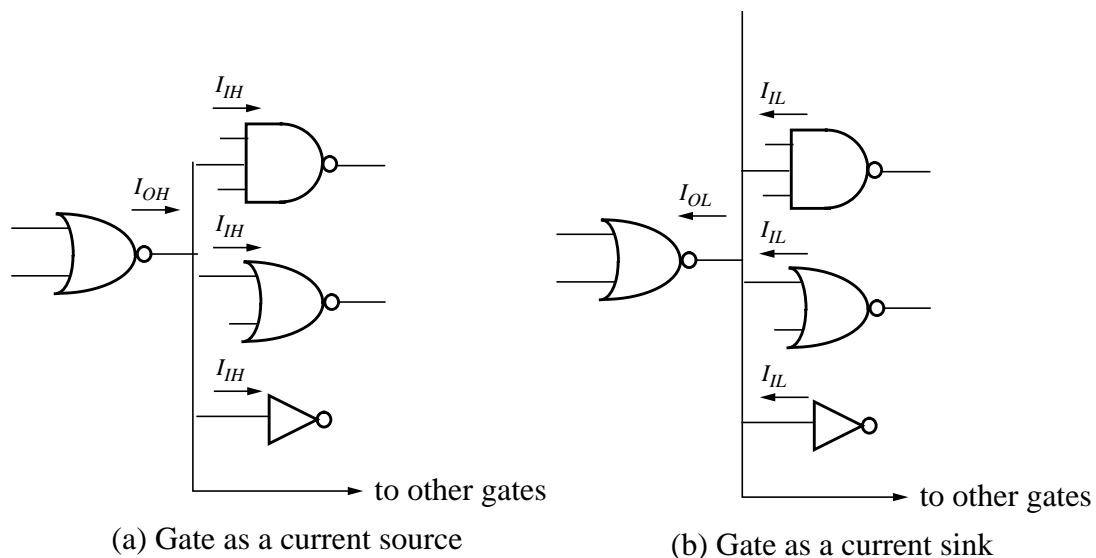


Figure 9: Current flow in a typical logic circuit [Gajski].

*Power Dissipation and Propagation Delay

- ☞ The average **power dissipation** is the product of average current and power supply voltage ($P = IV$).
 - ⇒ In standard CMOS technology, the power dissipation increases linearly with respect to the input transition rate.
 - ⇒ Heat removal is the major issue for high power dissipation.
- ☞ **Rise time**: delay for a signal to switch from 10% to 90% of its nominal value.
- ☞ **Fall time**: delay for a signal to switch from 90% to 10% of its nominal value.
 - ⇒ Rise time and fall time may not be equal.
- ☞ t_{PHL} (t_{PLH}): delay for the output signal to reach 50% of its nominal value on the H-to-L (L-to-H) transition after the input signal reached 50% of its nominal value.
- ☞ **Propagation delay**: $t_P = (t_{PHL} + t_{PLH})/2$.

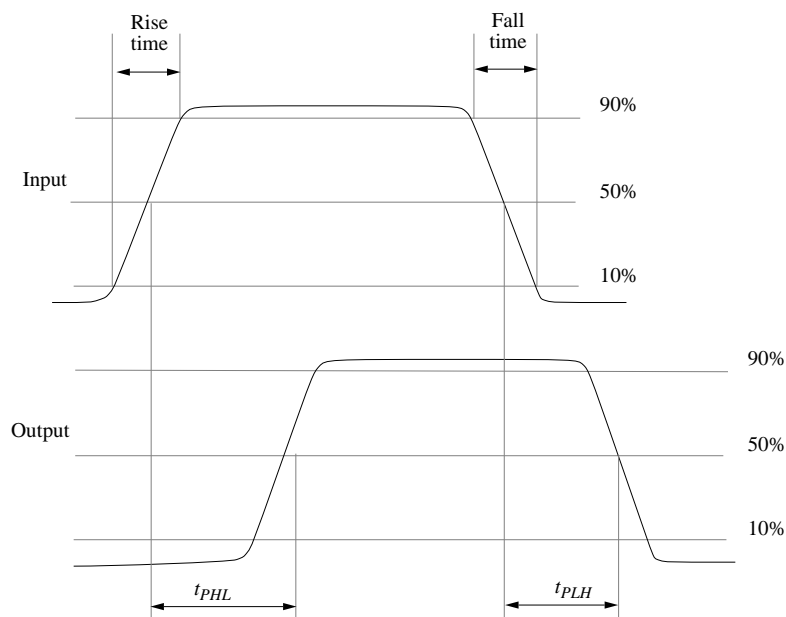


Figure 10: Propagation delay [Gajski].

Integrated Circuits and VLSI Technology

☞ What is an **integrated circuit** (IC)?

Levels of integration	Acronym	Number of gates
Small-scale integration	SSI	≤ 10
Medium-scale integration	MSI	10–100
Large-scale integration	LSI	Hundreds–Thousands
Very large-scale integration	VLSI	\geq Tens of Thousands
System-on-Chip	SOC	\geq Millions

Exercise 6

What are the popular digital logic families commonly used today?

□