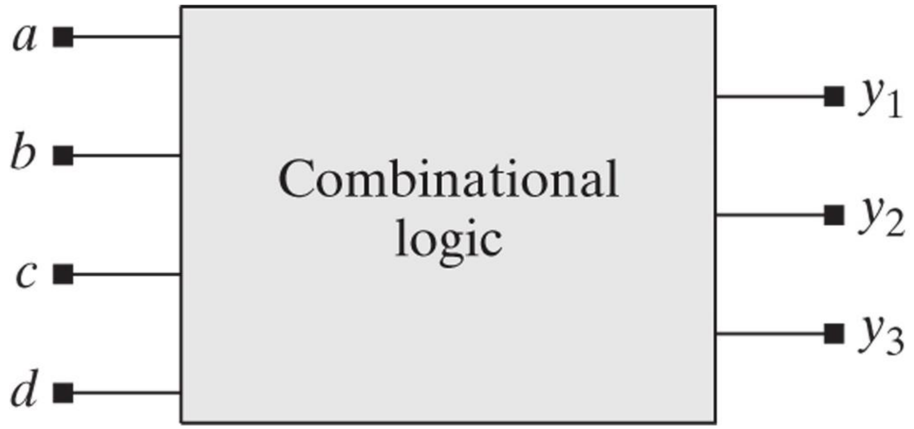
Digital Design with Verilog

Lecture 2: Review of Combinational Logic Design

Combinational Logic

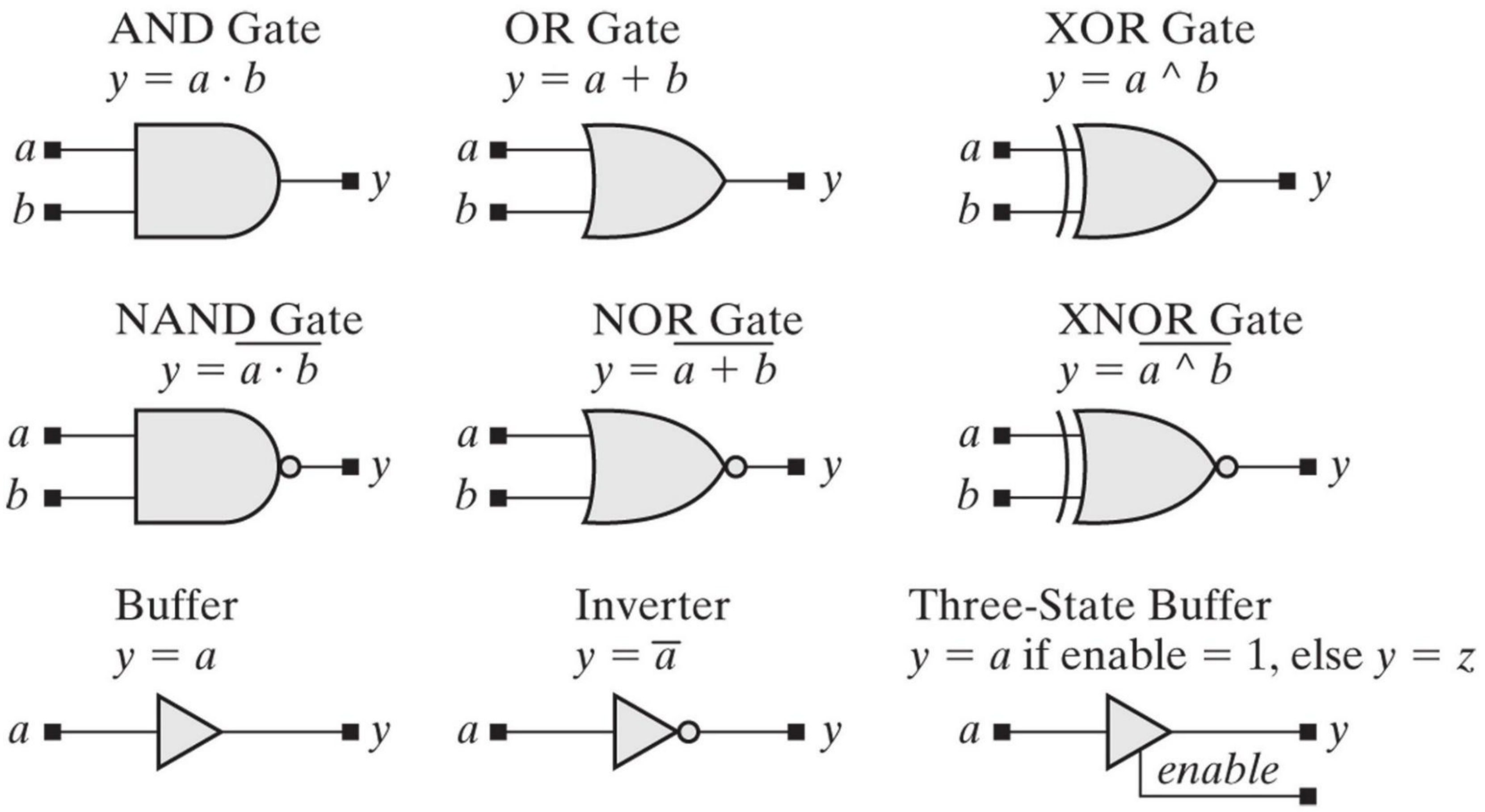
- Combinational logic forms its outputs as Boolean functions of its input variables on an instantaneous basis.



Block diagram symbol for combinational logic

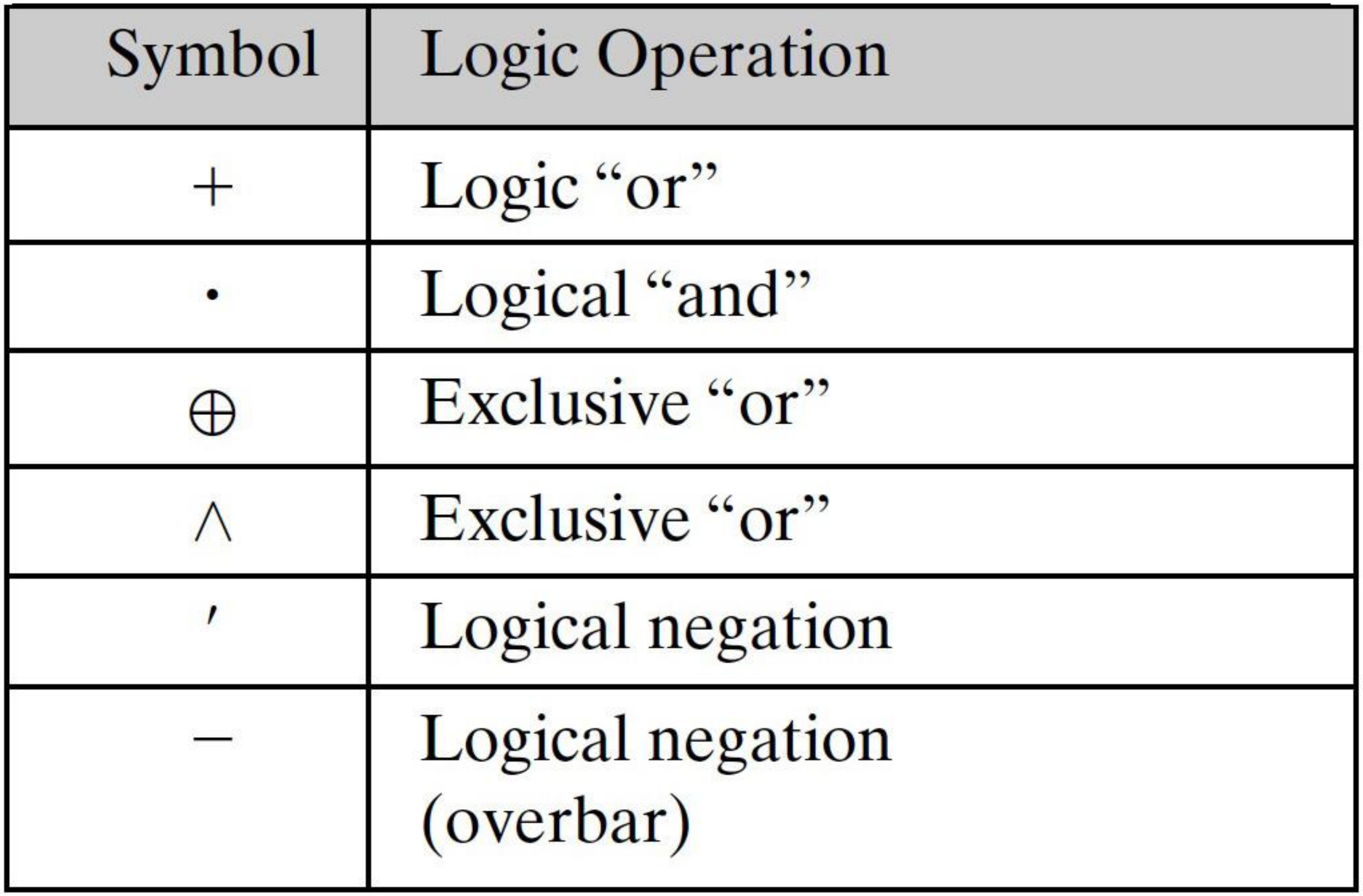
- At any time *t* the outputs y_1, y_2, and y_3 depend on only the values of *a, b,* c, and *d* at time *t.*

Common Logic Gates



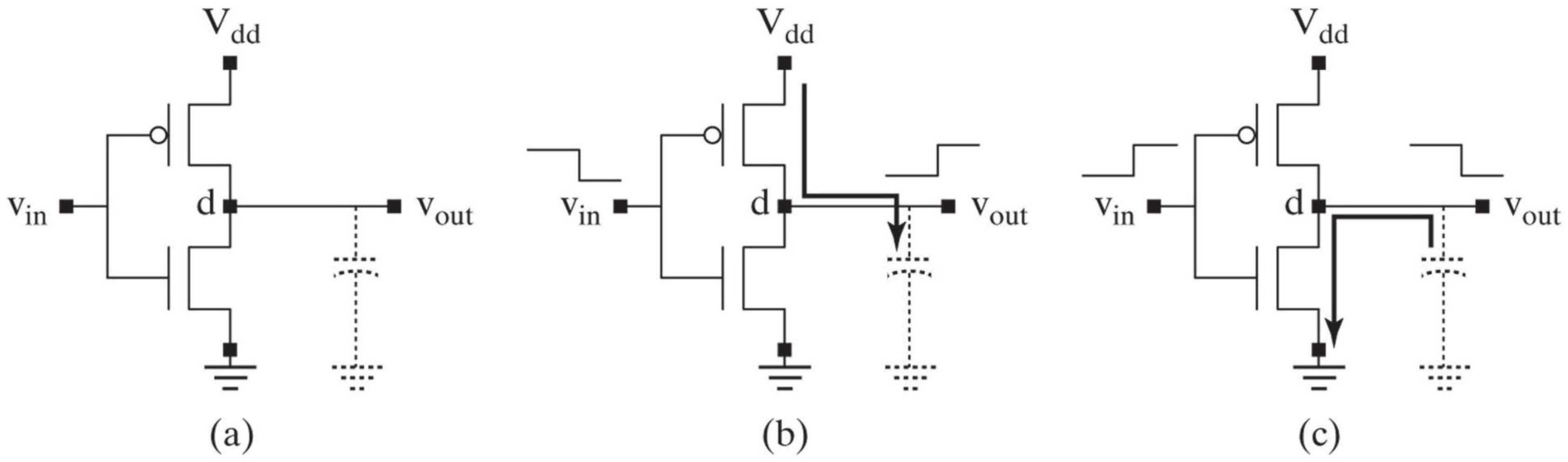
Schematic symbols and Boolean relationships for some common logic gates.

Boolean Logic Symbols and Operations



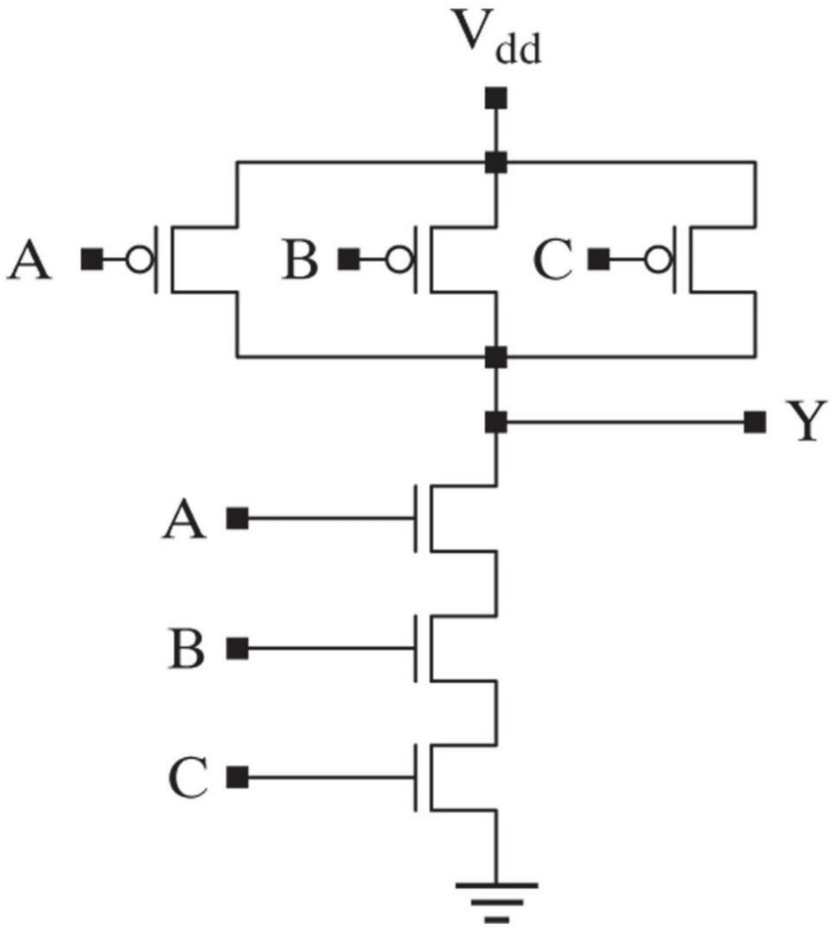
Symbol	Logic Operation
$+$	Logic “or”
$\cdot$	Logical “and”
$\oplus$	Exclusive “or”
$\wedge$	Exclusive “or”
$'$	Logical negation
$-$	Logical negation (overbar)

ASIC Library Cells

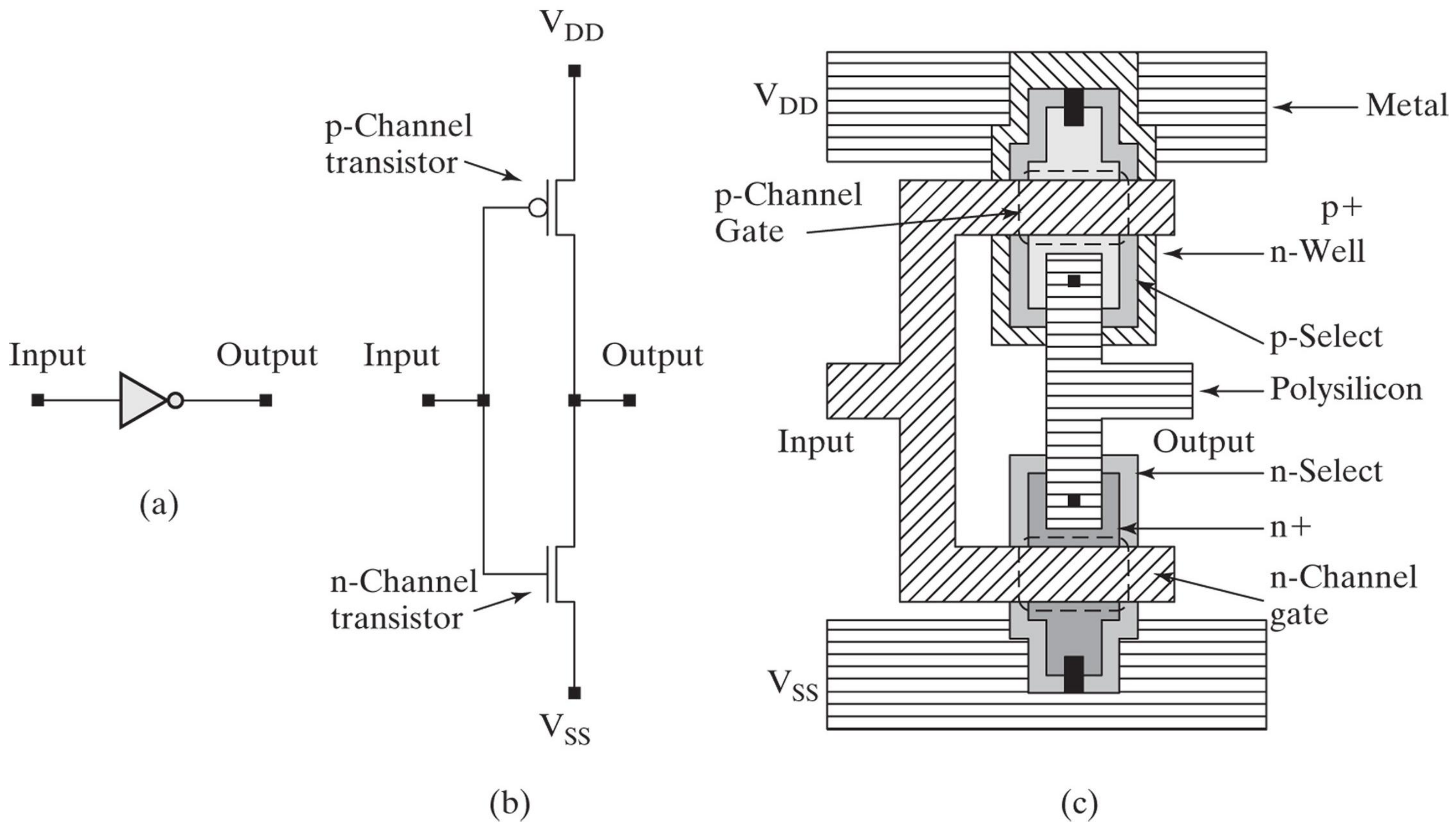


CMOS transistor-level schematics: (a) inverter with output load capacitance, (b) inverter with pull-up (charging) signal paths, and (c) inverter with pull-down (discharging) signal paths.

Three input CMOS NAND Gate

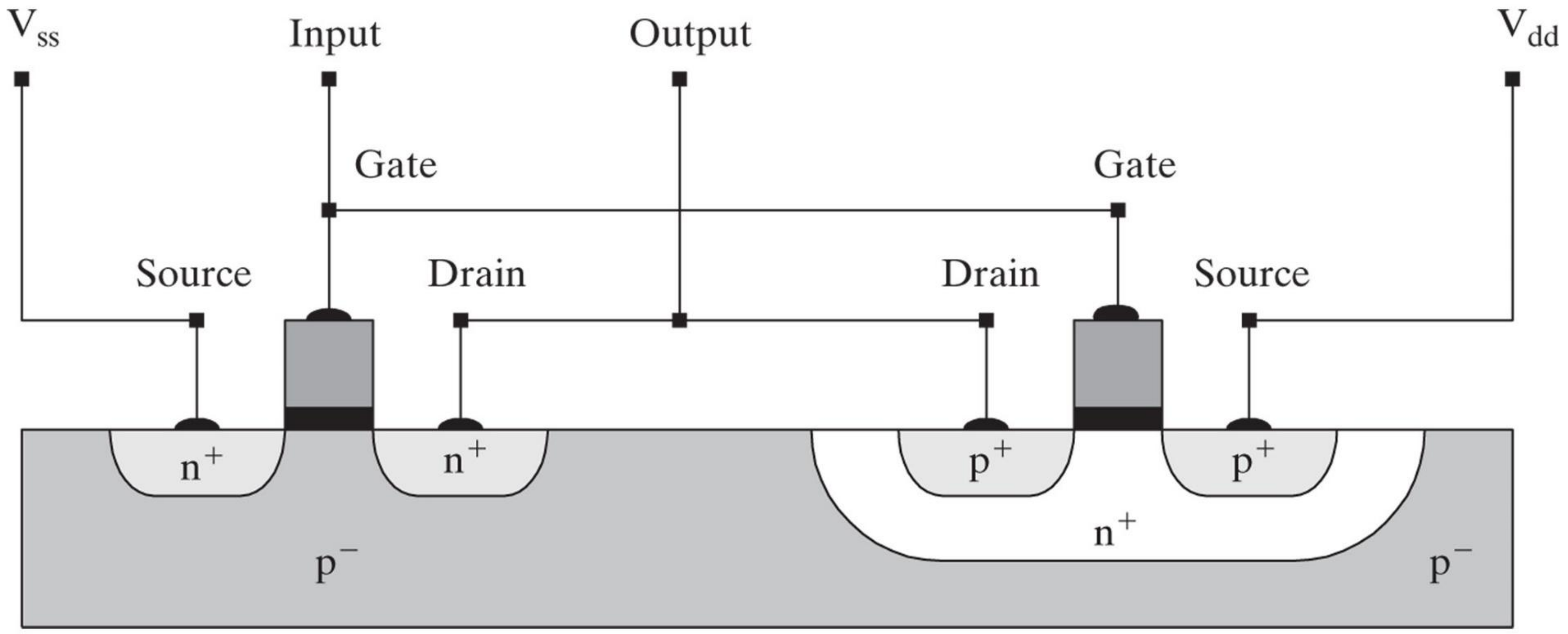


Views of a CMOS Inverter



Views of a CMOS inverter: (a) circuit-symbol view, (b) transistor-schematic view, and (c) simplified composite fabrication mask view.

Inverter-Cross Section



Simplified side view showing the doping regions of a CMOS inverter.

Standard Cell Library

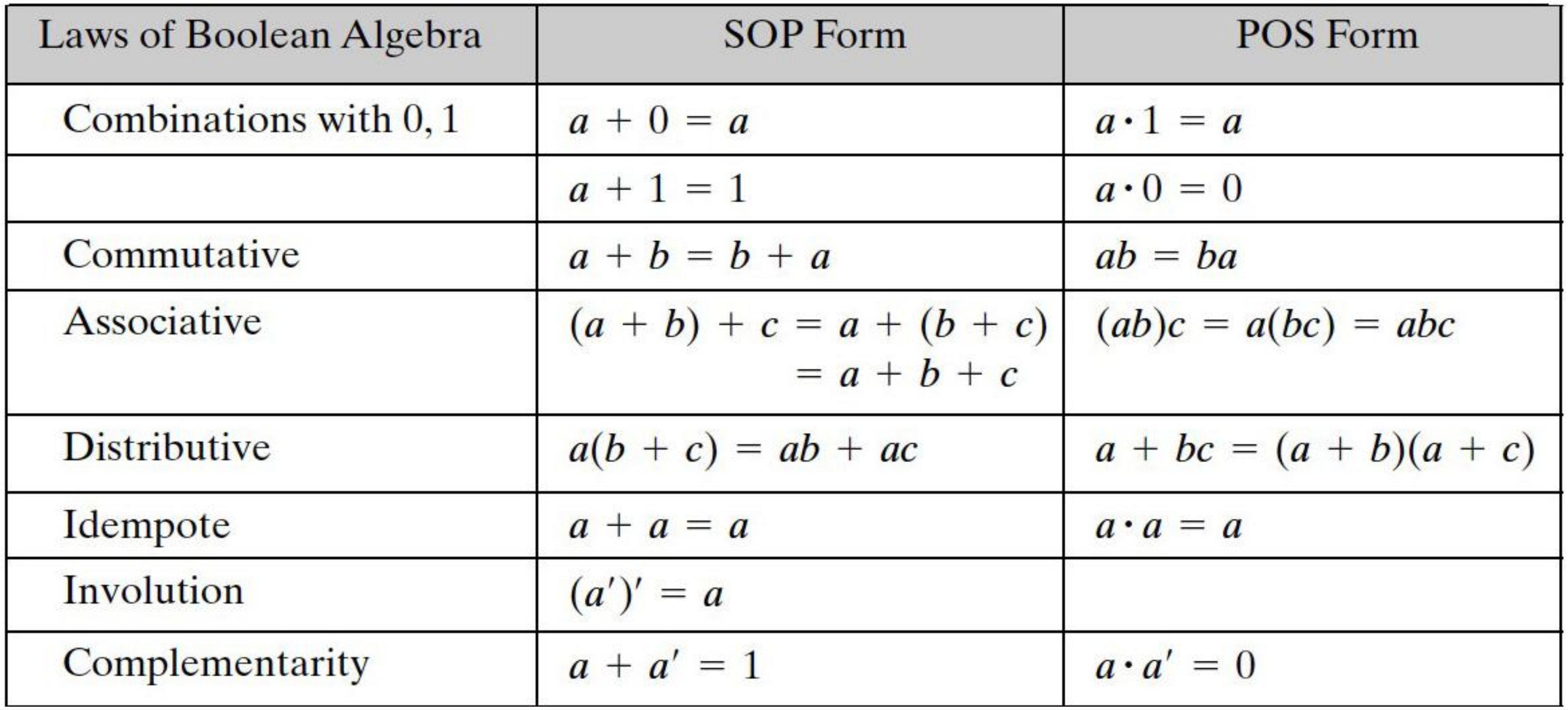
- Circuits that implement basic and moderately complex Boolean functions are characterized for
  - Functional, electrical, and timing properties,
  - Packaged in a library for repeated use in multiple designs.

- Such libraries commonly contain basic logic gates, flip-flops, latches, muxes, and adders.

- Synthesis tools build complex integrated circuits by mapping the result of logic synthesis onto the parts of a cell library to implement the specified functionality with acceptable performance.

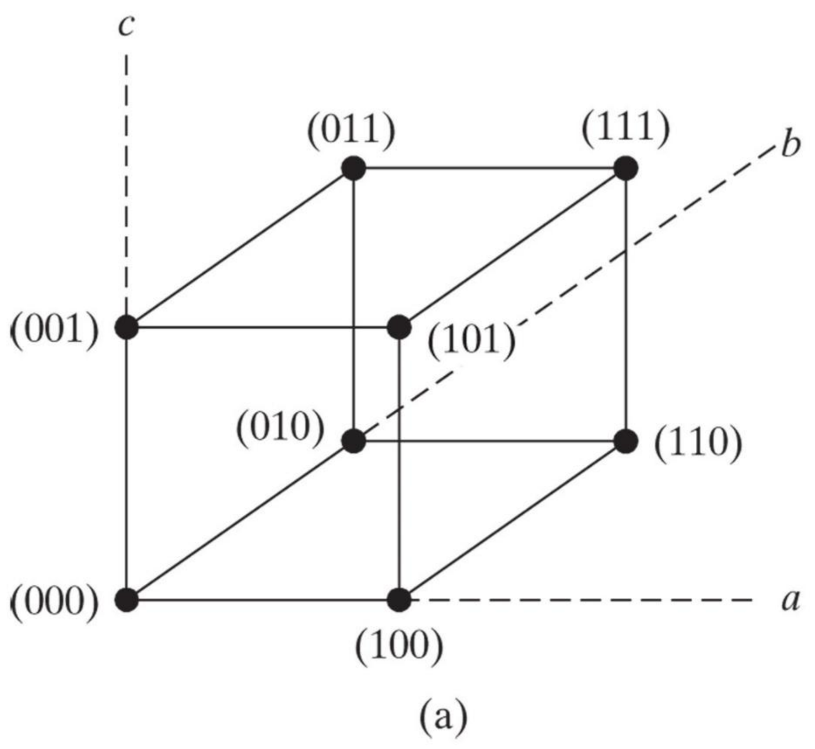
Boolean Algebra

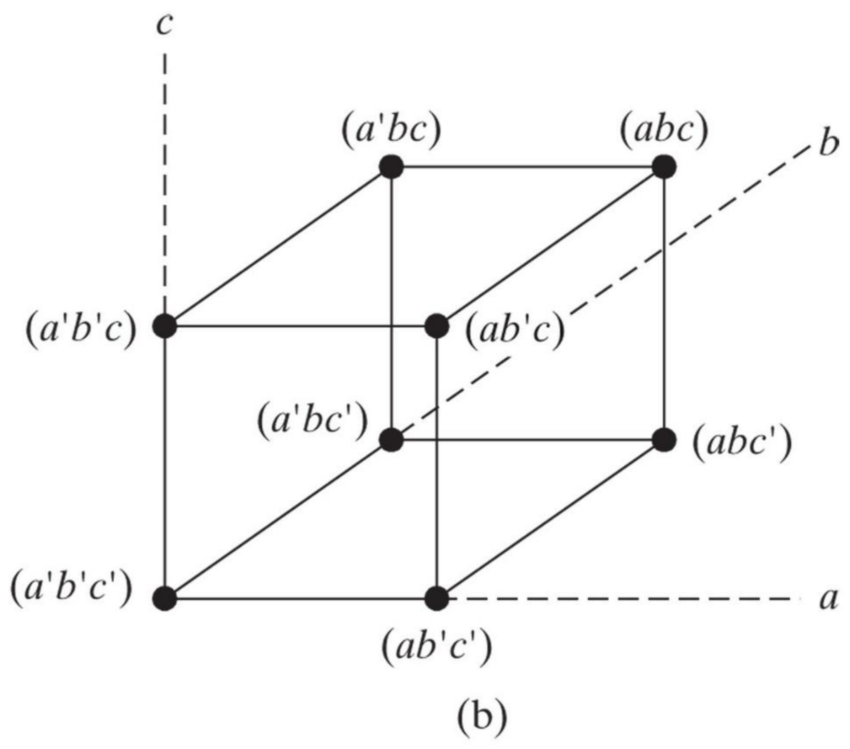
- The operations of logic circuits are described by Boolean algebra.
- A Boolean algebra consists or a set of values B = {0, 1} and the operators "+" and " · .



Laws of Boolean Algebra	SOP Form	POS Form
Combinations with 0, 1	$a + 0 = a$	$a \cdot 1 = a$
	$a + 1 = 1$	$a \cdot 0 = 0$
Commutative	$a + b = b + a$	$ab = ba$
Associative	$(a + b) + c = a + (b + c) = a + b + c$	$(ab)c = a(bc) = abc$
Distributive	$a(b + c) = ab + ac$	$a + bc = (a + b)(a + c)$
Idempote	$a + a = a$	$a \cdot a = a$
Involution	$(a')' = a$	
Complementarity	$a + a' = 1$	$a \cdot a' = 0$

Boolean Space

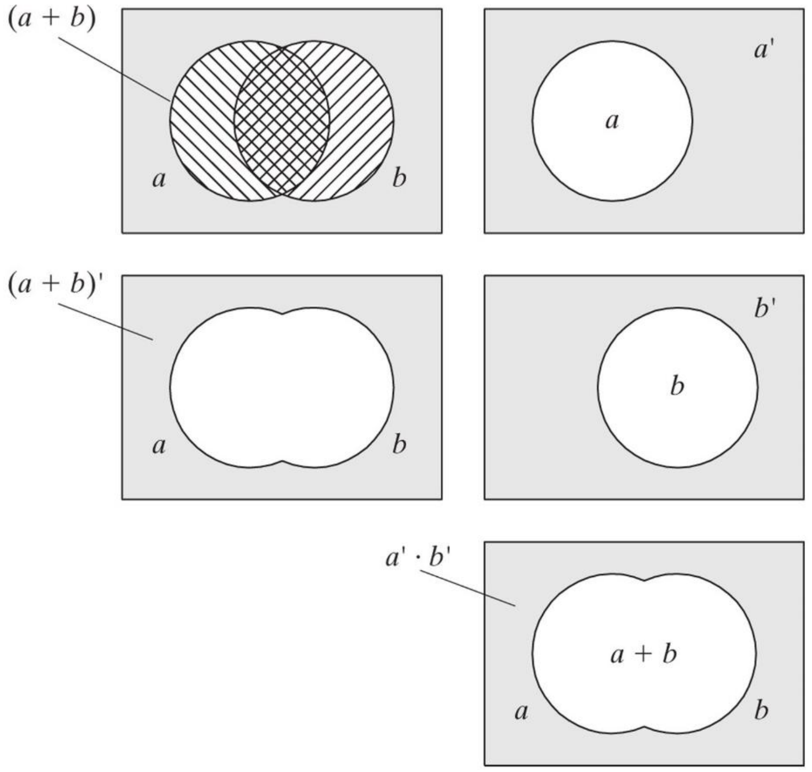




Points in a Boolean space: (a) represented by vectors of binary variables and (b) represented symbolically.

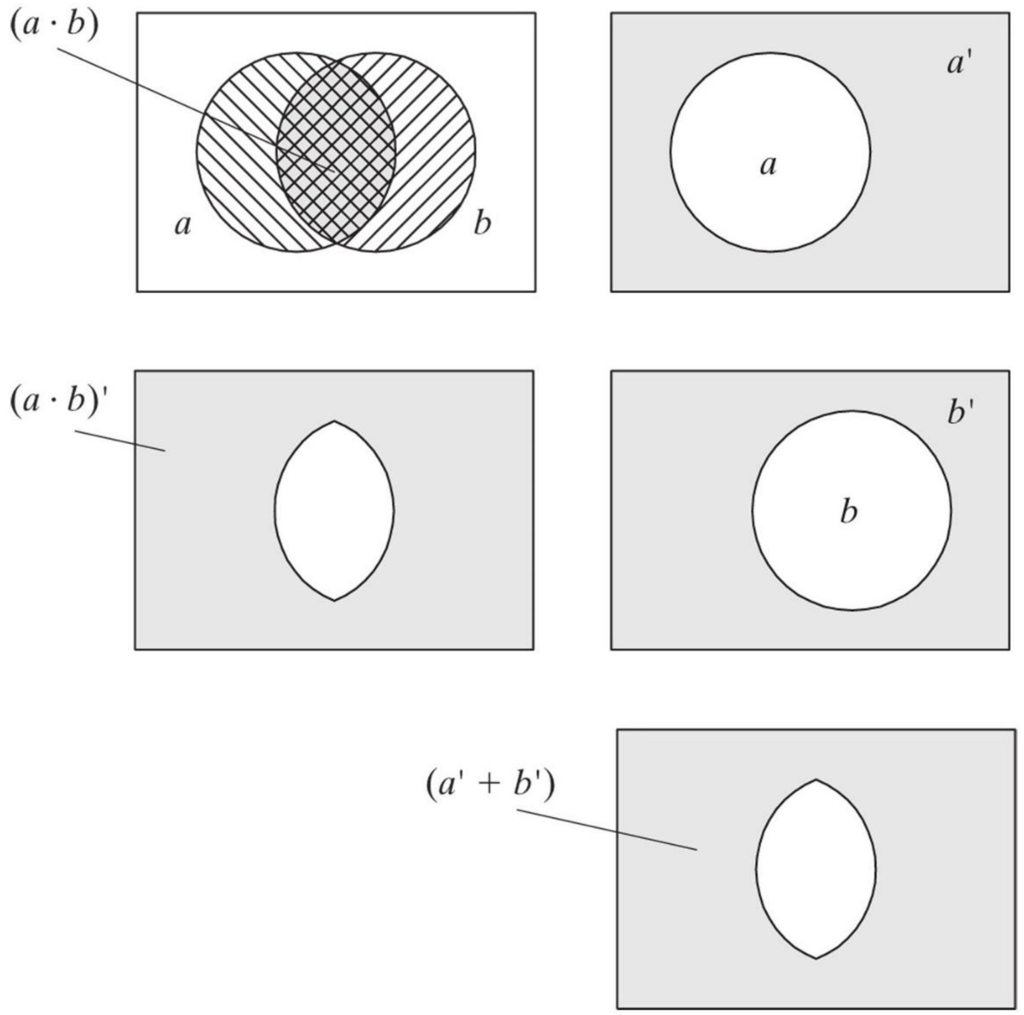
DeMorgan’s Laws

- DeMorgan's laws allow us to transform a circuit from an SOP form to a POS form, and vice versa.



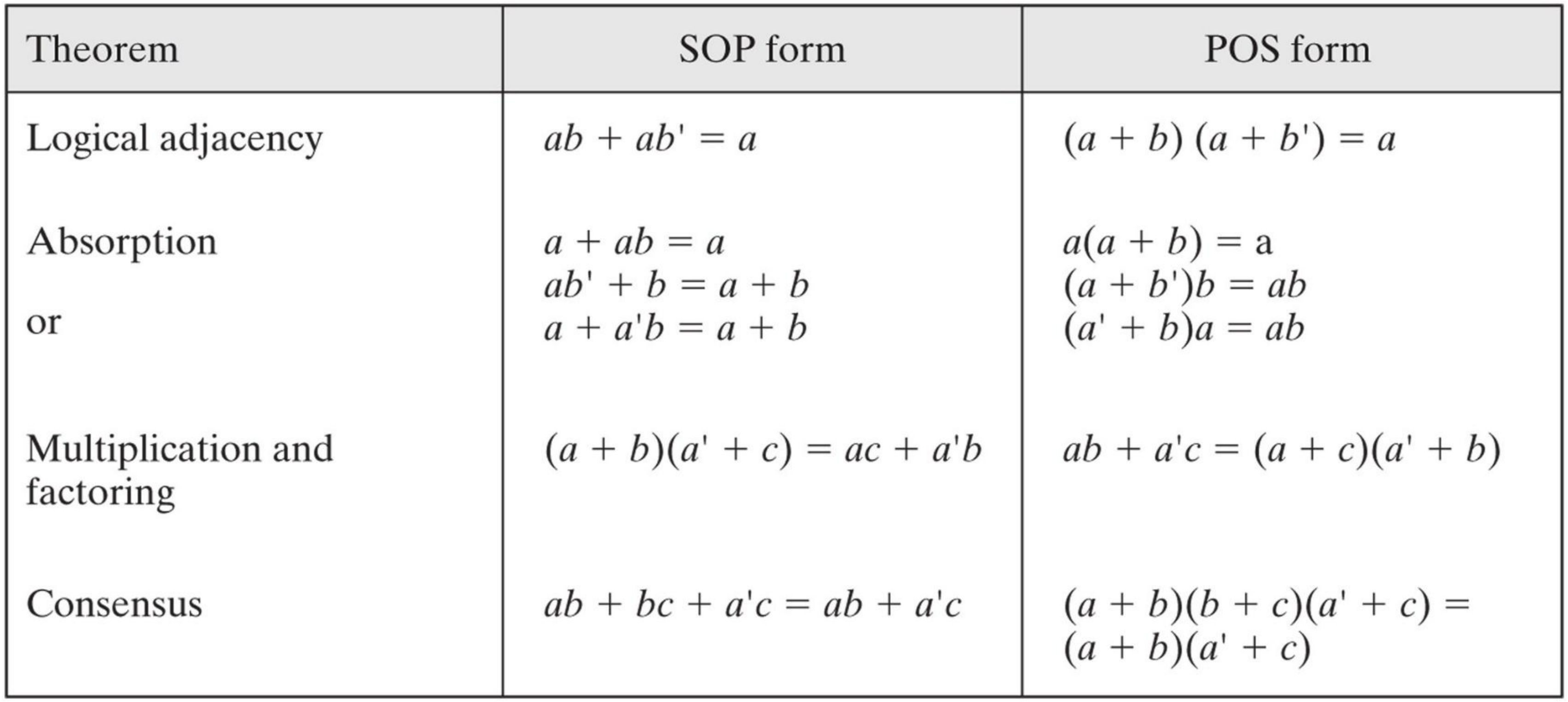
Venn diagrams illustrating DeMorgan’s law: $(a + b)' = a' \cdot b'$

DeMorgan’s Laws



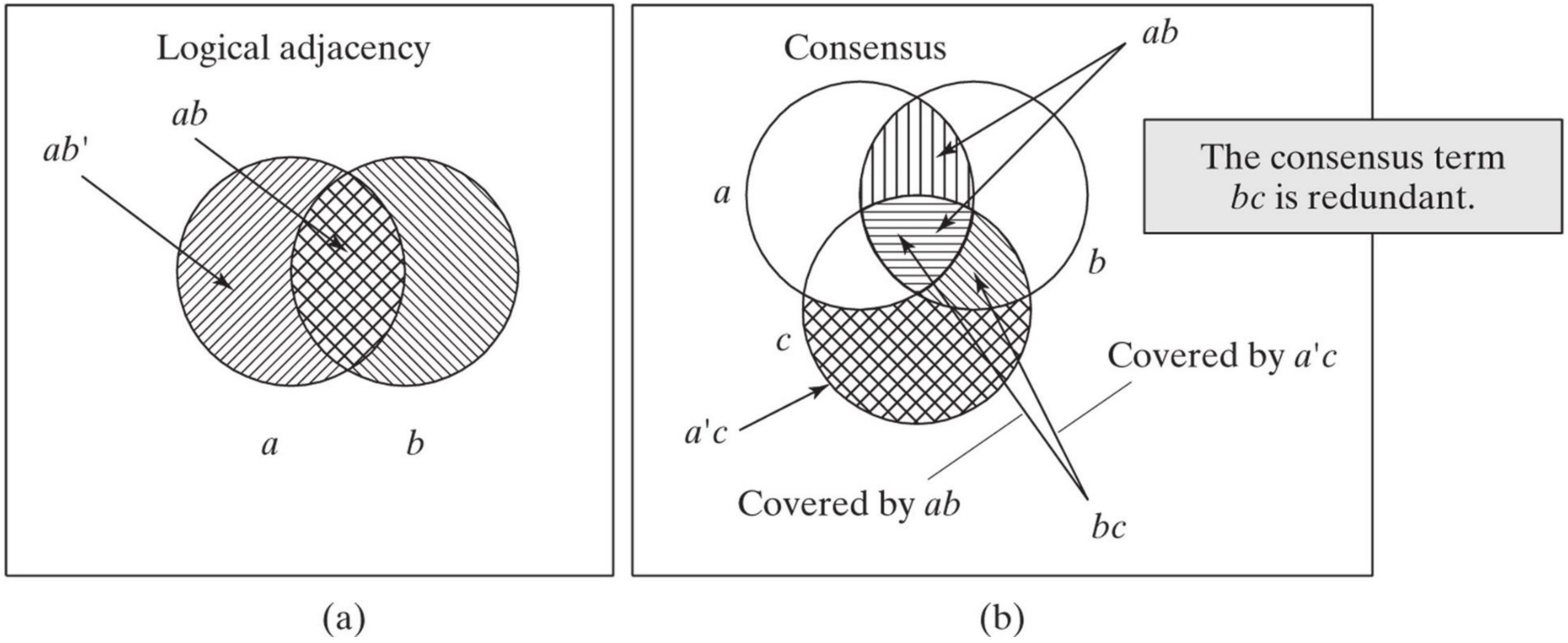
Venn diagrams illustrating DeMorgan’s law: *(a · b)’* = a’ + b’.

Theorems for Boolean Algebraic Minimization



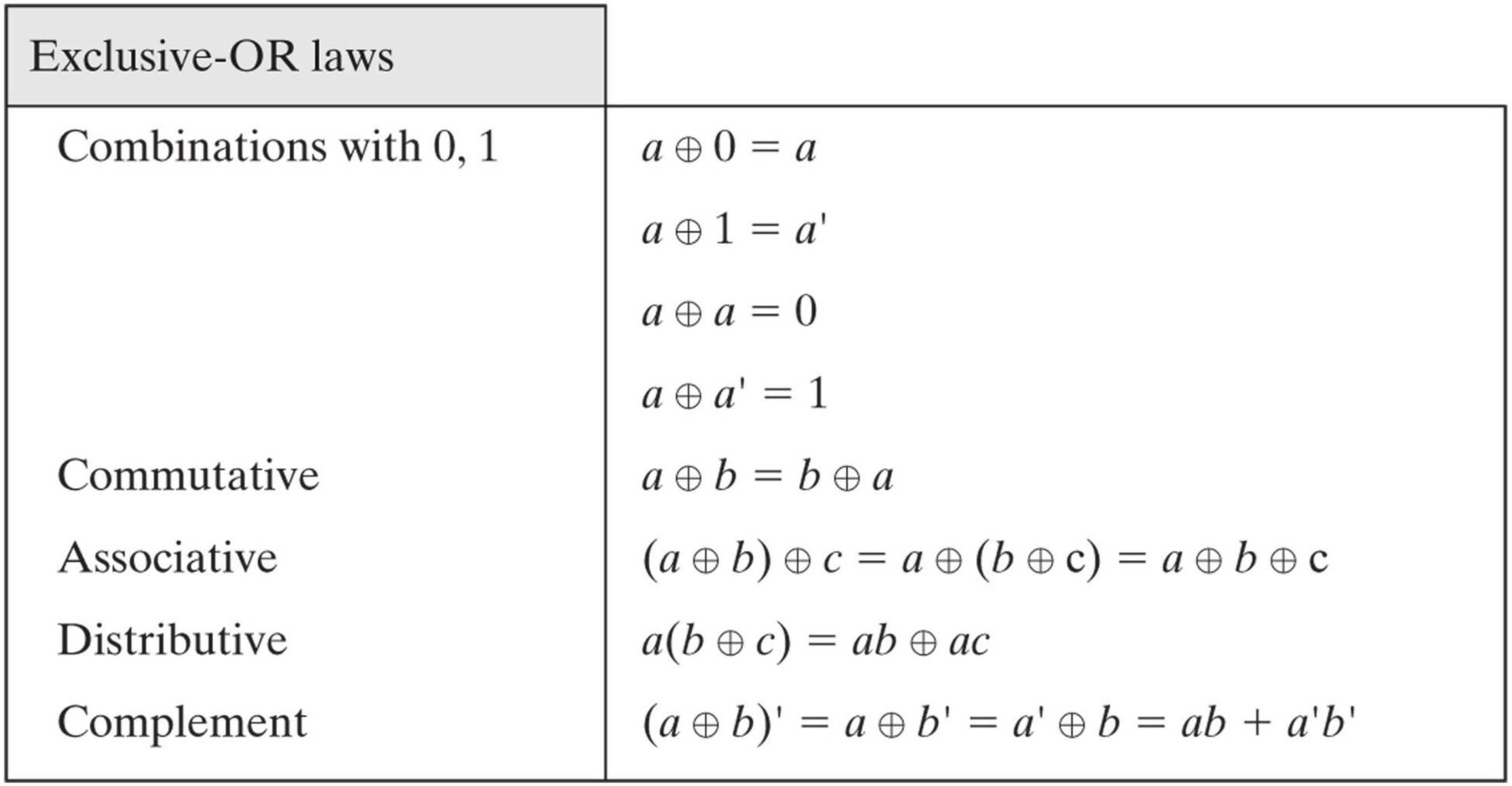
Theorem	SOP form	POS form
Logical adjacency	$ab + ab' = a$	$(a + b)(a + b') = a$
Absorption	$a + ab = a$	$a(a + b) = a$
	$ab' + b = a + b$	$(a + b')b = ab$
or	$a + a'b = a + b$	$(a' + b)a = ab$
Multiplication and factoring	$(a + b)(a' + c) = ac + a'b$	$ab + a'c = (a + c)(a' + b)$
Consensus	$ab + bc + a'c = ab + a'c$	$(a + b)(b + c)(a' + c) = (a + b)(a' + c)$

Theorems for Boolean Algebraic Minimization



Venn diagrams: (a) logical adjacency and (b) consensus.

Exclusive-OR laws



Exclusive-OR laws	
Combinations with 0, 1	$a \oplus 0 = a$
	$a \oplus 1 = a'$
	$a \oplus a = 0$
	$a \oplus a' = 1$
Commutative	$a \oplus b = b \oplus a$
Associative	$(a \oplus b) \oplus c = a \oplus (b \oplus \mathrm{c}) = a \oplus b \oplus \mathrm{c}$
Distributive	$a(b \oplus c) = ab \oplus ac$
Complement	$(a \oplus b)' = a \oplus b' = a' \oplus b = ab + a'b'$

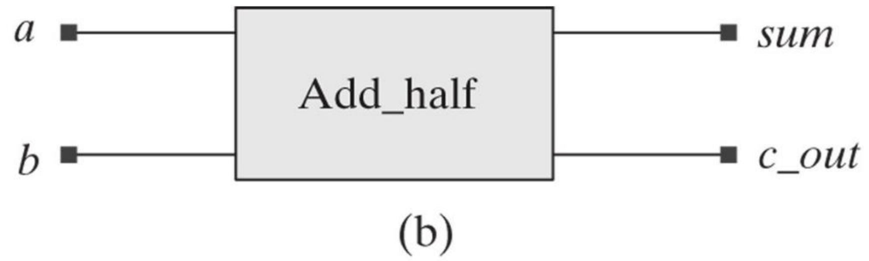
Representation of Combinational Logic

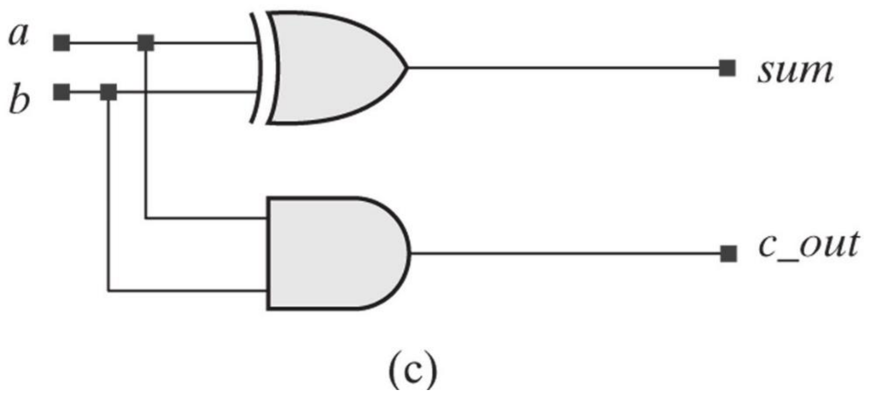
- Three common representations of combinational logic:
  - Structural (i.e., gate-level) schematics
  - Truth tables
  - Boolean equations
- Binary decision diagram (BDD)s are used primarily within EDA software tools because they can be more efficient and easier to manipulate than truth tables.

Half Adder

Inputs		Outputs	
a	*b*	*c_out*	*sum*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(a)



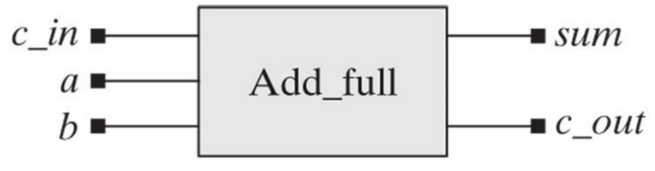


Half adder: (a) truth table, (b) block diagram symbol, and (c) schematic.

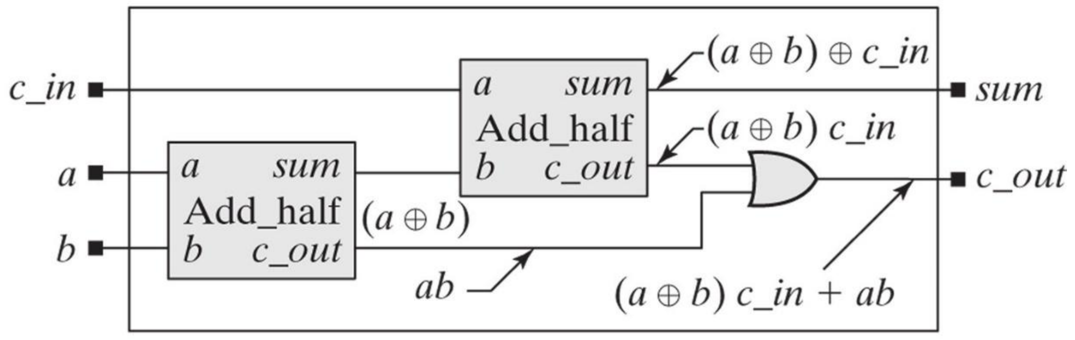
Full Adder

Inputs			Outputs	
a	*b*	*c_in*	*c_out*	*sum*
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

(a)



(b)



(c)

Full adder: (a) truth table, (b) block diagram symbol, and (c) schematic for a full adder composed of half adders and glue logic.

Sum-of-Products Representation

- A cube is formed as the product of literals in which a literal appears in either uncomplemented or complemented form.

- A *Boolean expression* is a set of cubes and is typically expressed in an SOP form as the "OR" of product terms (cubes), rather than in set notation.

- A *minterm* is a cube in which every variable appears. The variable will be in either true (uncomplemented) or complemented form (but not both).

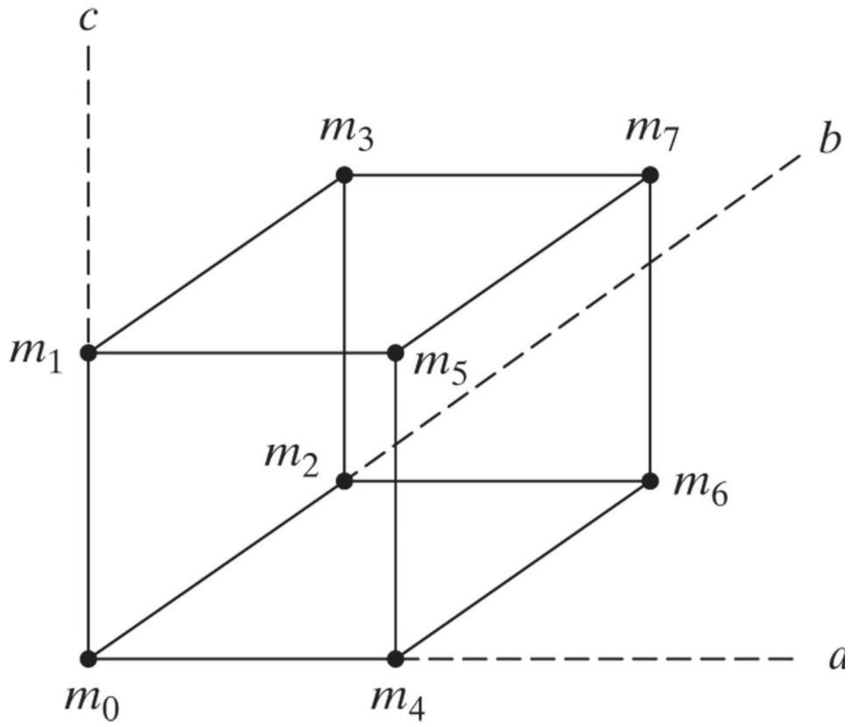
Sum-of-Products Representation

- A Boolean expression in SOP form is said to be canonical if every cube in the expression has a unique representation in which all of the literals are in complemented or uncomplemented form.

- A *canonic* (standard) SOP function is also called a standard sum of products (SSOP).

- A *Boolean function* is a set of minterms (vertices) at which the function is asserted. A Boolean function in SOP form is expressed as a sum of minterms.

Sum-of-Products Representation



Correspondence between minterms and vertices in **B**^{3}.

$$sum = m_1 + m_2 + m_4 + m_7 = \Sigma\, m(1, 2, 4, 7)$$
$$c\_out = m_3 + m_5 + m_6 + m_7 = \Sigma\, m(3, 5, 6, 7)$$

Products-of-Sums Representation

- A Boolean function can also be expressed in a POS form in which the expression is written as a product of Boolean factors, each of which is a sum of literals.
- A Boolean expression in POS form is said to be *canonical* (i.e., a unique representation for a given function) if each factor has all of the literals in complemented or uncomplemented form, but not both.
- A *maxterm* is an OR-ed sum of literals in which each variable appears exactly once in true or complemented form.

Simplification of Boolean Functions

- Minimization is important because the cost of hardware implementing a Boolean expression is related to
  - The number of terms in the expression and
  - The the number of literals in a term, that is, in a cube in an SOP expression.
- A Boolean expression in SOP form is said to be *minimal,* if it contains a minimal number of product terms and literals (i.e., a given term cannot be replaced by another that has fewer literals).

Simplification of Boolean Functions

- Approaches for minimizing Boolean Expressions:

  Application of Boolean theorems,

  Karnaugh maps,

  Quine-McCluskey minimization, and

  Synthesis tools (Espresso-II).

Simplification of Boolean Functions

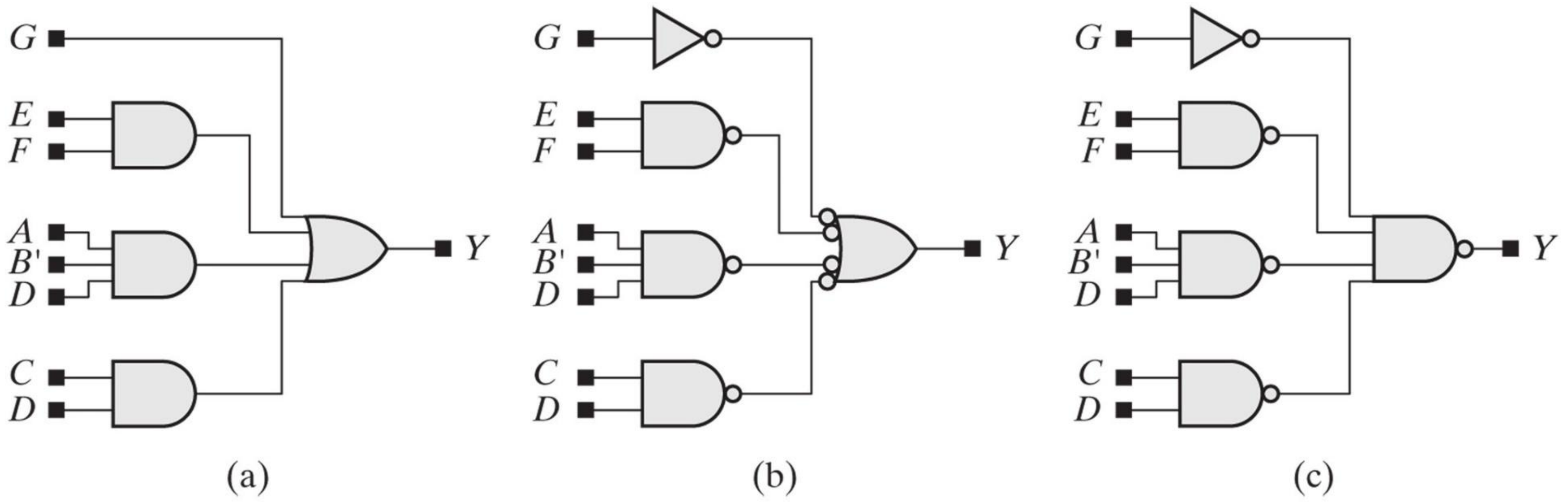
- Logic minimization searches for efficient representations of Boolean functions.

- In a Boolean expression, a cube that is contained in another cube is said to be redundant---a cube is *redundant* if its set of vertices is properly contained in the set of vertices of another cube of the function.

- A Boolean expression is *nonredundant (irredundant)* if no cube contains another cube.
- The cubes of an irredundant expression do not share a common vertex, that is, their corresponding sets of vertices are pairwise disjoint

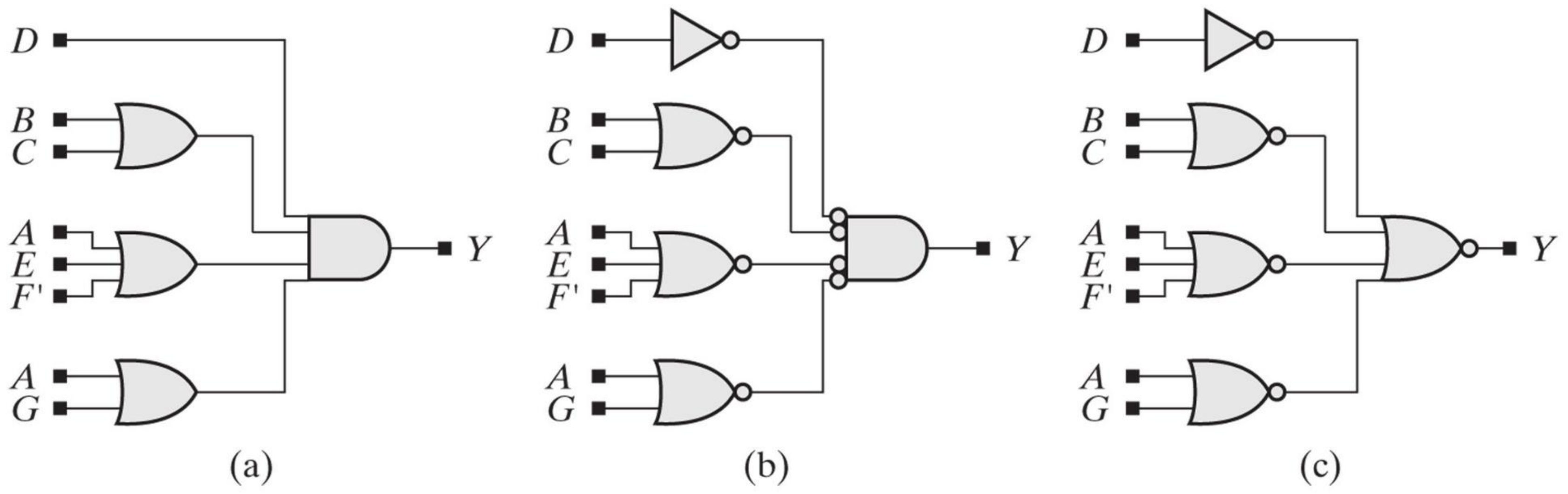
Building Blocks for Logic Design

NAND-NOR Structures



Circuit transformations to obtain a NAND/Inverter realization of an SOP expression.

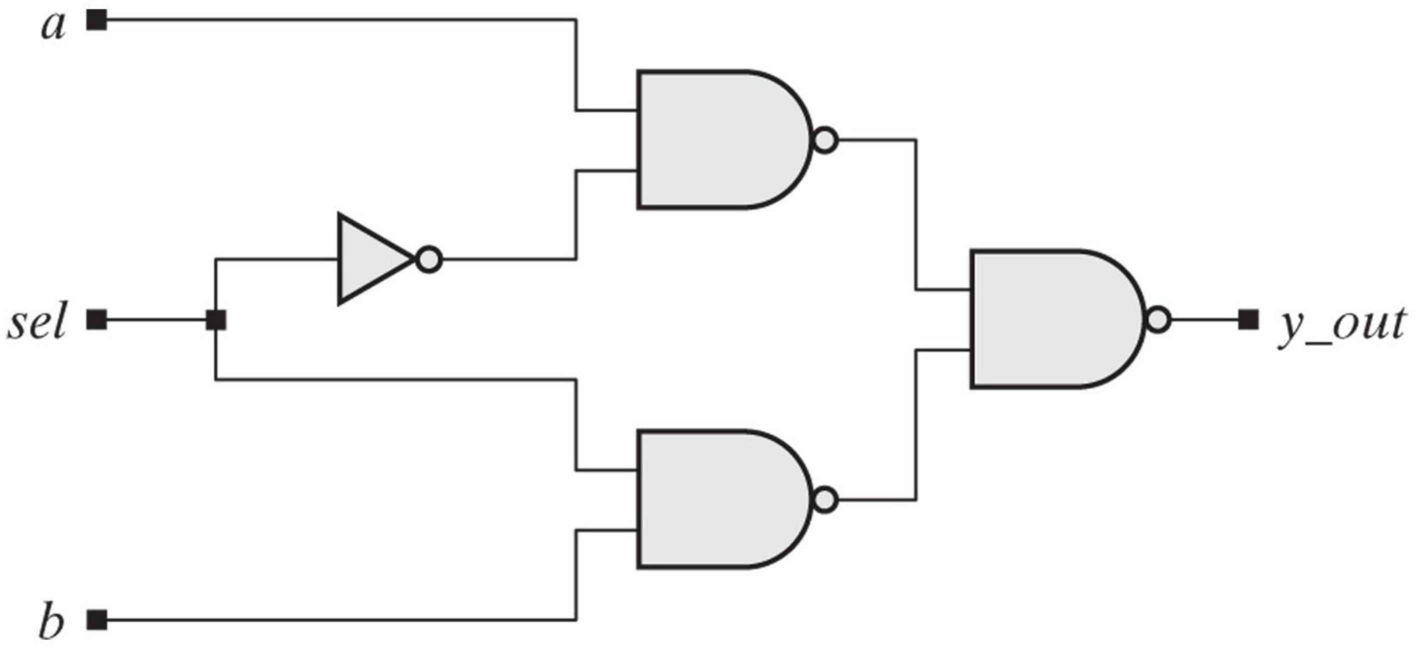
NAND-NOR Structures



Circuit transformations to obtain a NOR/Inverter realization of a POS expression.

Multiplexers

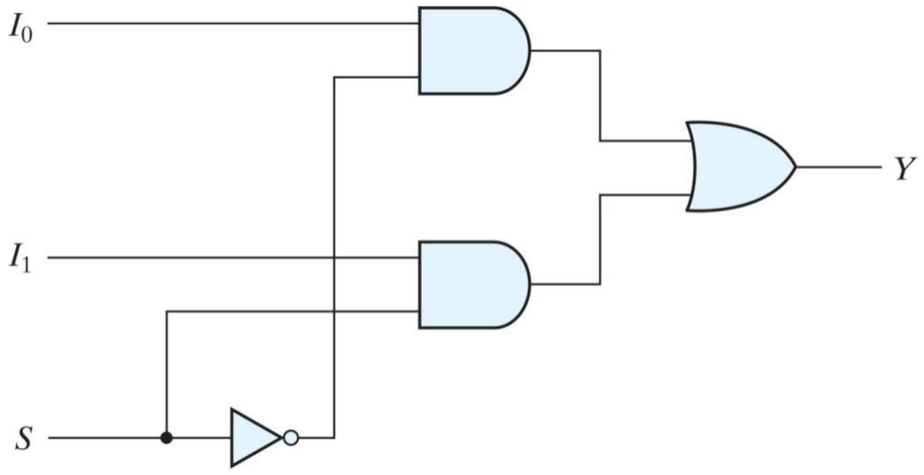
- Multiplexer circuits are used to steer data through functional units of computers and other digital systems.



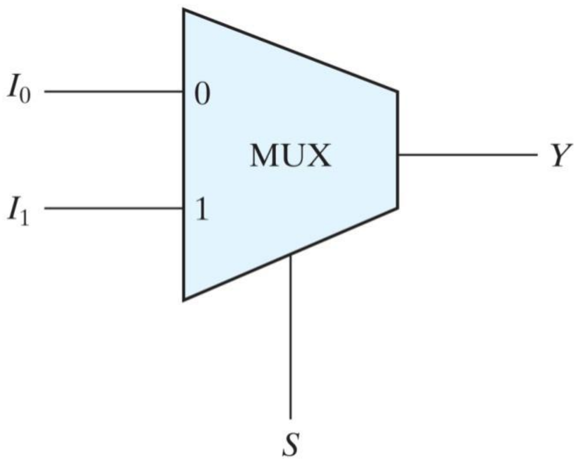
Gate-level schematic for a two-channel multiplexer circuit.

Multiplexers/Data selectors

- A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
  - The selection of a particular input line is controlled by a set of selection lines.
  - Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

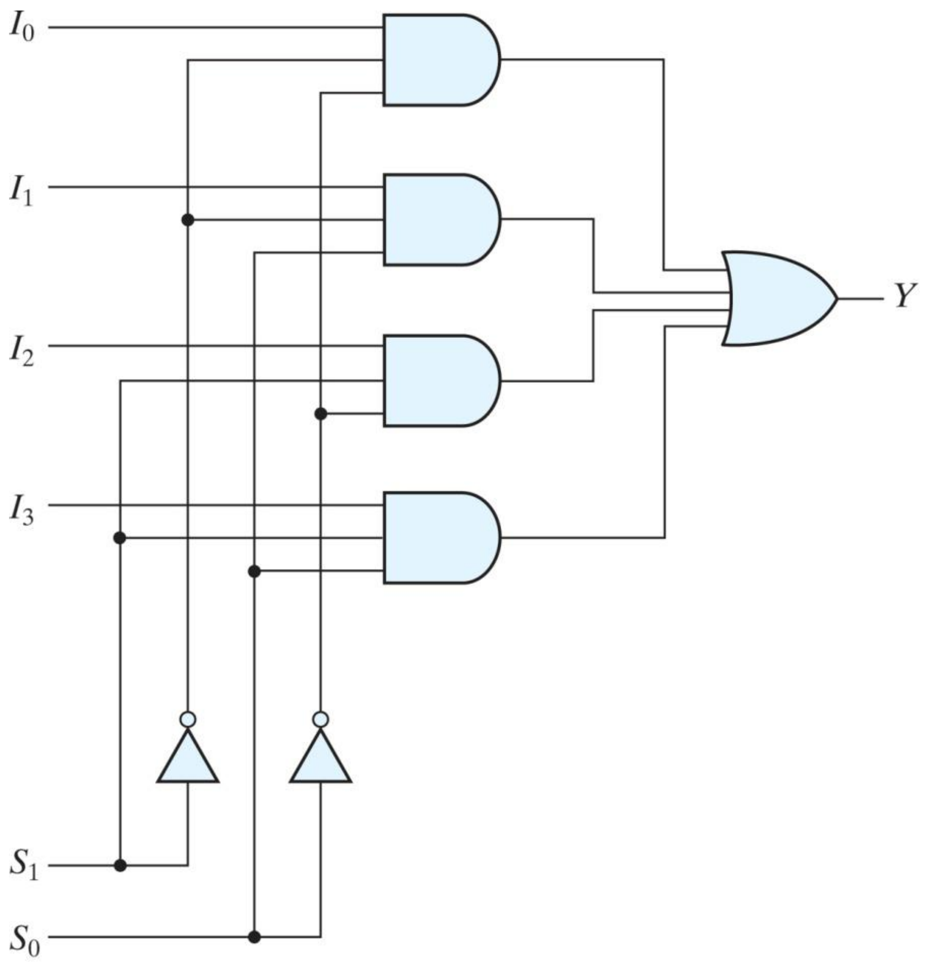


(a) Logic diagram



(b) Block diagram

4x1 multiplexer



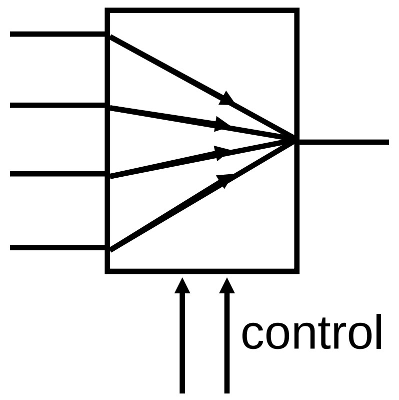
(a) Logic diagram

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

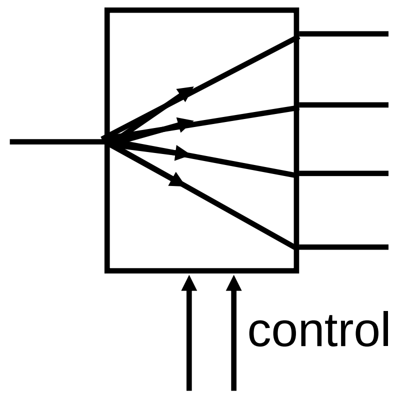
(b) Function table

Demultiplexer

- Multiplexer (MUX)
  - Routes one of many inputs to a single output
  - Also called a *selector*

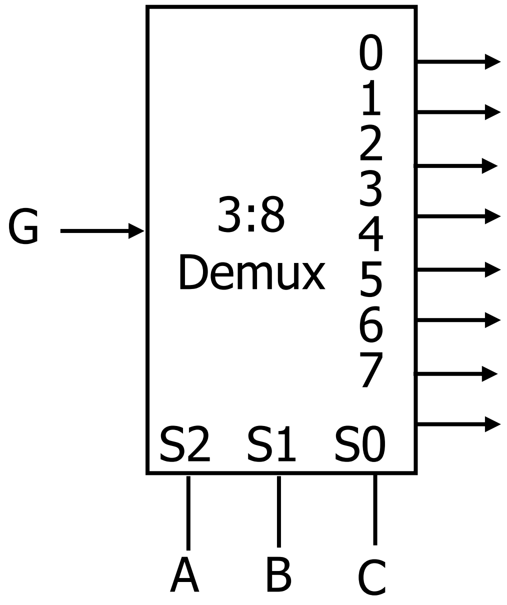


- Demultiplexer (DEMUX)
  - Single data input; n control inputs ("selects"); 2^n outputs
    - Single input connects to one of 2^n outputs
    - "Selects" decide which output is connected to the input



3x8 Demux

- The input is called an “enable” (G)



3:8 Decoder:

Out0 = G • S2' • S1' • S0'
Out1 = G • S2' • S1' • S0
Out2 = G • S2' • S1 • S0'
Out3 = G • S2' • S1 • S0
Out4 = G • S2 • S1' • S0'
Out5 = G • S2 • S1' • S0
Out6 = G • S2 • S1 • S0'
Out7 = G • S2 • S1 • S0

Encoders

- An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2^n (or fewer) input lines and n output lines.

Truth Table of an Octal-to-Binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

- It is assumed that only one input has a value of 1 at any given point in time.

Encoders (Contd.)

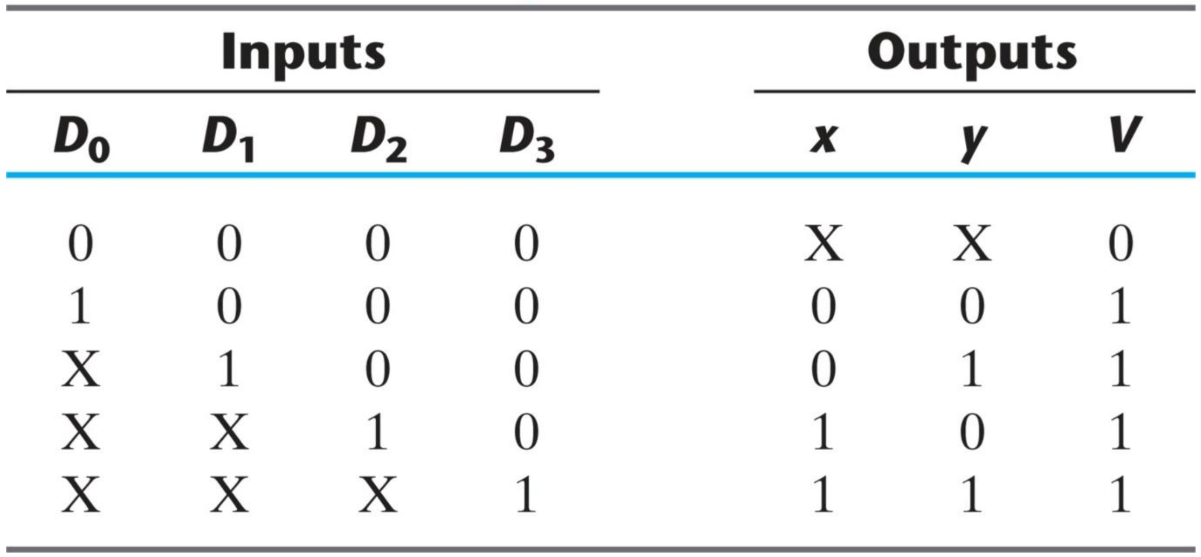
- The encoder can be implemented with OR gates whose inputs are determined directly from the truth table.

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

- Two problems
  - What if two inputs are active simultaneously?
  - Output with all 0’s generated when all the inputs are 0; but this same as when D_0 is equal to 1.

Priority Encoder

- A priority encoder is an encoder circuit that includes the priority function. The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.



Truth Table of a Priority Encoder

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

- Input D_3 has the highest priority.

Decoders

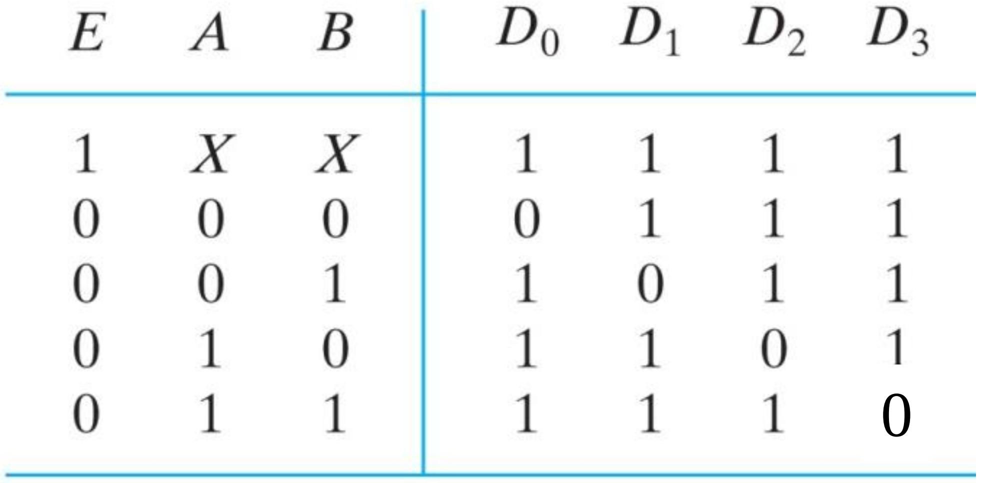
- A decoder is a combinational circuit that converts binary information from *n* input lines to an 2^n unique output lines.

- Some Applications:
    - Microprocessor memory system: selecting different banks of memory.
    - Microprocessor I/O: Selecting different devices.
    - Memory: Decoding memory addresses (e.g. in ROM).

2-to-4-line DECODER with Enable

- The decoder is enabled when *E* = 0. The output whose value = 0 represents the minterm is selected by inputs *A* and *B.*

- The decoder is inactive when *E* = 1 -> *D*0 …*D*3 = 1
  - A Decoder with enable input is called a decoder/demultiplexer.
- Demultiplexer receives directs it to the output l



E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

Thank you