Chapter 2 Combinational systems

- Definition: combinational logic, block, system
- Logic gates and truth table
- Don’t care condition
- switching algebra (Boolean)
- Complement
- Functions with gates

Digital logic circuits

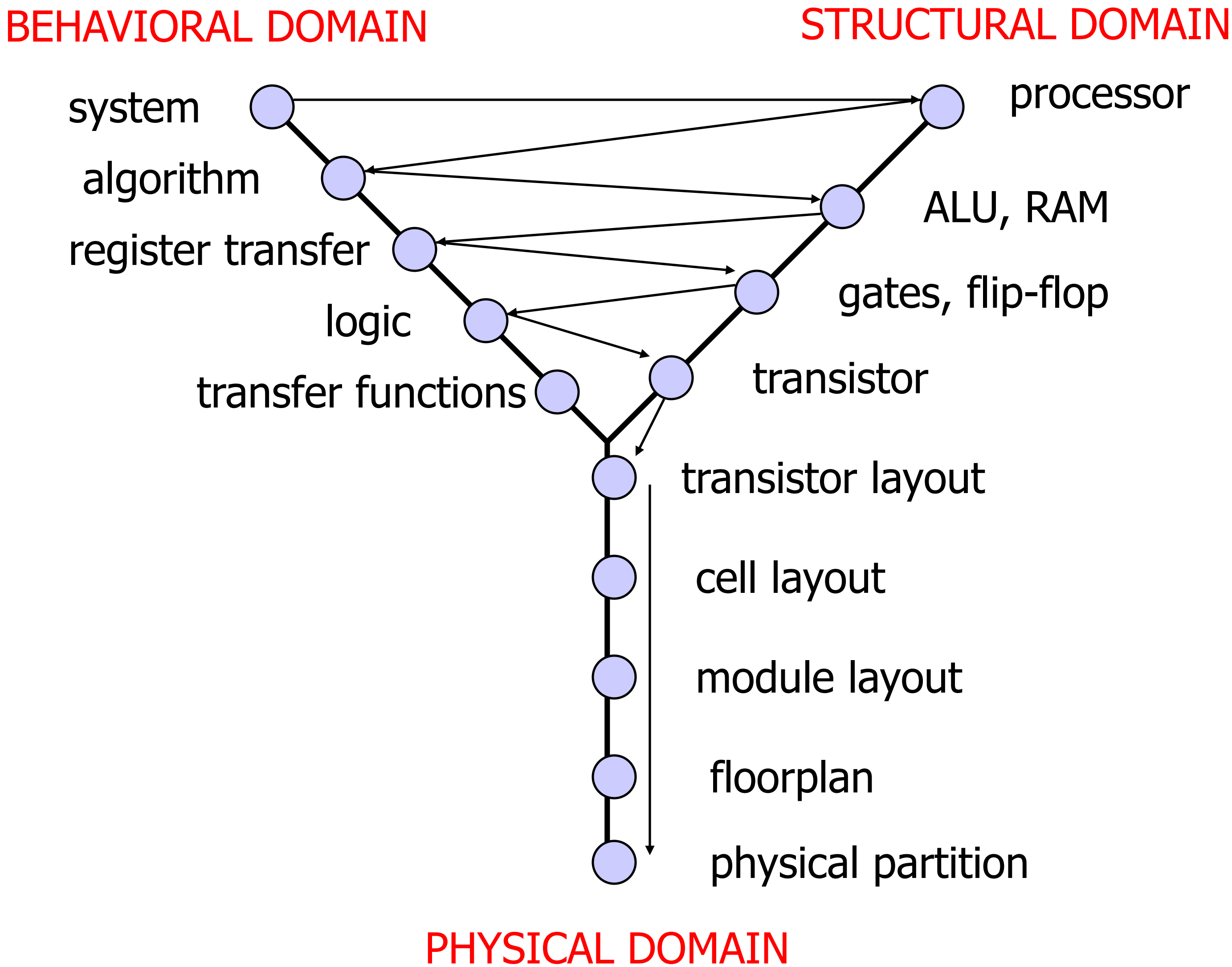
- Digital circuits: hardware components that manipulate binary information
- Logic gates implement logic functions.
- Basic logical operators are the logic functions AND, OR and NOT
- Boolean Algebra: a useful mathematical system for specifying and transforming logic functions.
- We study Boolean algebra as a foundation for designing and analyzing digital systems!

- Hierarchical design: circuit design hierarchy (Y-chart)
- Basic circuit element can be
  - Transistor
  - Logic gate
  - Wire

Levels of abstractions

Level	Behavioral forms	Structural components	Physical objects
Transistor	differential equations, current-voltage diagrams	transistors, resistors, capacitors	analog and digital cells
Gate	Boolean equations, finite state machines	gates, flip-flops	modules, units
Register	algorithms, flowcharts, instruction sets	adders, registers, counters, queues	microchips
Processor	executable specification, programs	processors, controllers, memories	printed-circuit boards, multichip modules

Y chart



BEHAVIORAL DOMAIN
STRUCTURAL DOMAIN
system
algorithm
register transfer
logic
transfer functions
processor
ALU, RAM
gates, flip-flop
transistor
transistor layout
cell layout
module layout
floorplan
physical partition
PHYSICAL DOMAIN

Logical operations

The three basic logical operations are:

AND

OR

NOT

AND is denoted by a dot (·)
OR is denoted by a plus (+)
NOT is denoted by an overbar (¯), a single quote mark (') after, or (~) before the variable

Examples:

Y = A•B is read "Y is equal to A AND B."
Z = X + Y is read "z is equal to x OR y."
X = ~A is read "X is equal to NOT A."

- **Note: The statement:**

  1 + 1 = 2 (read "one <u>plus</u> one equals two")

  is not the same as

  1 + 1 = 1 (read "1 <u>or</u> 1 equals 1")

Operator definition

- Operations are defined on the values "0" and "1" for each operator:

AND	**OR**	**NOT**
$0 \cdot 0 = 0$	$0 + 0 = 0$	$\overline{0} = 1$
$0 \cdot 1 = 0$	$0 + 1 = 1$	$\overline{1} = 0$
$1 \cdot 0 = 0$	$1 + 0 = 1$	
$1 \cdot 1 = 1$	$1 + 1 = 1$	

Truth table

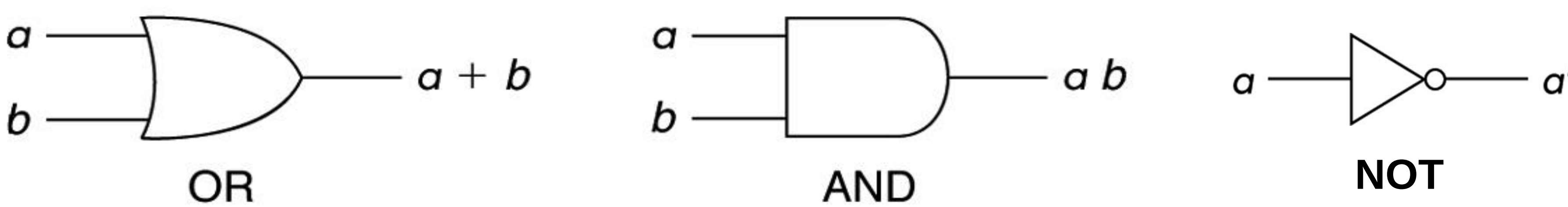
- *Truth table* - a tabular listing of the values of a function for all possible combinations of values on its arguments
- Example: Truth tables for the basic logic operations:

A B	C = A + B
0 0	0
0 1	1
1 0	1
1 1	1

A B	C = A · B
0 0	0
0 1	0
1 0	0
1 1	1

A	C = ~A
0	1
1	0

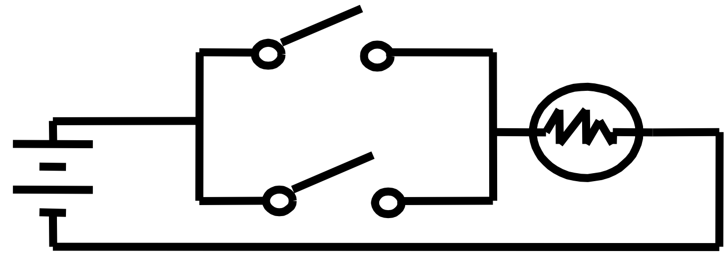
- symbols



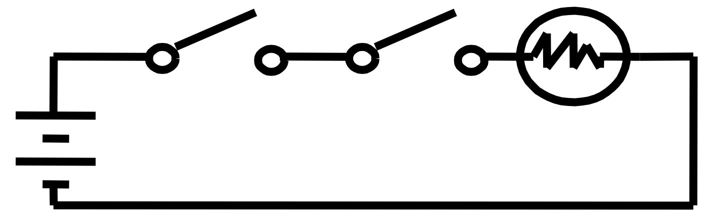
Logic function implementation

- **Using Switches**
  - **For inputs:**
    - **logic 1 is <u>switch closed</u>**
    - **logic 0 is <u>switch open</u>**
  - **For outputs:**
    - **logic 1 is <u>light on</u>**
    - **logic 0 is <u>light off</u>.**
  - **NOT uses a switch such that:**
    - **logic 1 is <u>switch open</u>**
    - **logic 0 is <u>switch closed</u>**

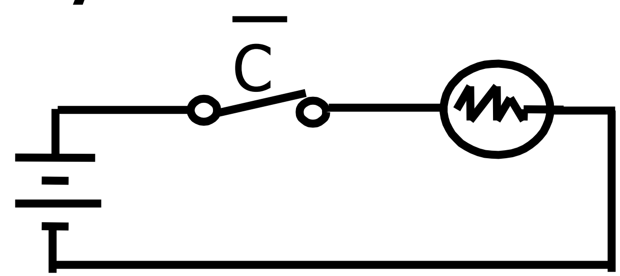
Switches in parallel => OR



Switches in series => AND



Normally-closed switch => NOT



Logic Diagrams and Expressions

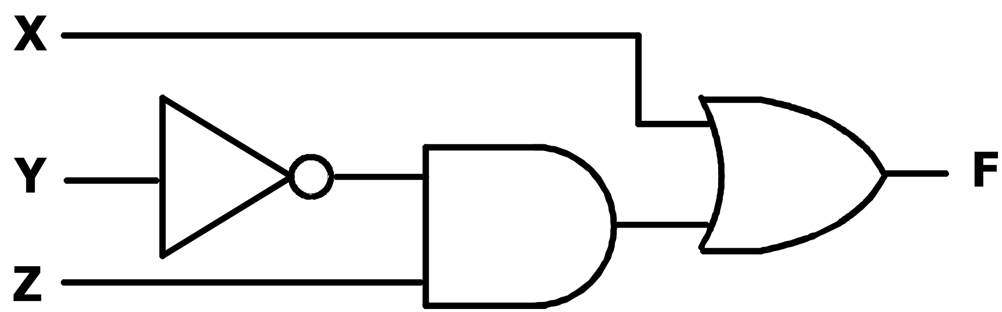
Truth Table

X Y Z	$F = X + \overline{Y} \times Z$
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

Equation

$$F = X + \overline{Y}\,Z$$

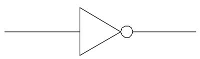
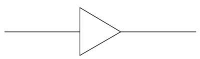
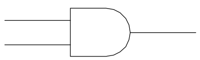
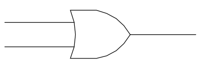
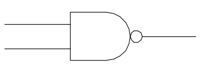
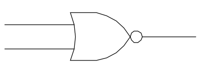
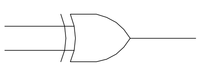
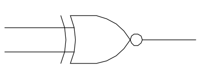
Logic Diagram



- Boolean equations, truth tables and logic diagrams describe the same function!

- Truth tables are unique; expressions and logic diagrams are not. This gives flexibility in implementing functions.

Basic logic library

NAME	GRAPHIC SYMBOL	FUNCTIONAL EXPRESSION	COST(NUMBER OF TRANSISTORS)	GATE DELAY(NS)	Truth table
Inverter		$F = x'$	2	1	x | F 0 | 1 1 | 0
Driver		$F = x$	4	2	x | F 0 | 0 1 | 1
AND		$F = xy$	6	2.4	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x+y$	6	2.4	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NAND		$F = (xy)'$	4	1.4	x y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = (x+y)'$	4	1.4	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR		$F = x \oplus y$	14	4.2	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
XNOR		$F = x \odot y$	12	3.2	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Design process

- **Design process is a sequence of steps which leads from a product concept to manufacturing drawings that show how to build that product**
- **Computer design**
  - **Server, desktop, embedded**
- **System design**
- **ASIC (application specific integrated circuit)**
- **IP blocks (processor, memory, I/O, control units)**

Switching algebra

P1a.	$a + b = b + a$	P1b.	$ab = ba$	Commutative
P2a.	$a + (b + c) = (a + b) + c$	P2b.	$a(bc) = (ab)c$	Associative
P3a.	$a + 0 = a$	P3b.	$a \cdot 1 = a$	Identity
P3aa.	$0 + a = a$	P3bb.	$1 \cdot a = a$	
P4a.	$a + 1 = 1$	P4b.	$a \cdot 0 = 0$	Null
P4aa.	$1 + a = 1$	P4bb.	$0 \cdot a = 0$	
P5a.	$a + a' = 1$	P5b.	$a \cdot a' = 0$	Complement
P5aa.	$a' + a = 1$	P5bb.	$a' \cdot a = 0$	
P6a.	$a + a = a$	P6b.	$a \cdot a = a$	Idempotency
P7.	$(a')' = a$			Involution

Switching algebra

P8a.	$a(b + c) = ab + ac$	P8b.	$a + bc = (a + b)(a + c)$	Distributive
P9a.	$ab + ab' = a$	P9b.	$(a + b)(a + b') = a$	Adjacency
P9aa.	$a'b' + a'b + ab + ab' = 1$	P9bb.	$(a' + b')(a' + b)(a + b)(a + b') = 0$	
P10a.	$a + a'b = a + b$	P10b.	$a(a' + b) = ab$	Simplification
P11a.	$(a + b)' = a'b'$	P11b.	$(ab)' = a' + b'$	DeMorgan
P11aa.	$(a + b + c\ldots)' = a'b'c'\ldots$	P11bb.	$(abc\ldots)' = a' + b' + c'\ldots$	
P12a.	$a + ab = a$	P12b.	$a(a + b) = a$	Absorption
P13a.	$at_1 + a't_2 + t_1t_2 = at_1 + a't_2$	P13b.	$(a + t_1)(a' + t_2)(t_1 + t_2) = (a + t_1)(a' + t_2)$	Consensus
P14a.	$ab + a'c = (a + c)(a' + b)$			

Example 1: Boolean Algebraic Proof

- $A + A \cdot B = A$ (Absorption Theorem)

Proof Steps	Justification (identity or theorem)
$A + A \cdot B$	
$= A \cdot 1 + A \cdot B$	$X = X \cdot 1$
$= A \cdot (1 + B)$	$X \cdot Y + X \cdot Z = X \cdot (Y + Z)$ (Distributive Law)
$= A \cdot 1$	$1 + X = 1$
$= A$	$X \cdot 1 = X$

- Our primary reason for doing proofs is to learn:
  - Careful and efficient use of the identities and theorems of Boolean algebra, and
  - How to choose the appropriate identity or theorem to apply to make forward progress, irrespective of the application.

Example 2: Boolean Algebraic Proofs

- $AB + A'C + BC = AB + A'C$ (Consensus Theorem)

Proof Steps	Justification (identity or theorem)
$AB + A'C + BC$	
$= AB + A'C + 1 \cdot BC$	?
$= AB + A'C + (A + A') \cdot BC$	?
$=$	

Example 3: Boolean Algebraic Proofs

- $(X+Y)'Z+XY' = Y'(X+Z)$
  proof steps   Justification

Proof of DeMorgan's Laws

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

It is important that we do not USE DeMorgan's Laws in doing this proof. This requires a different proof method. We will show that, $x' \cdot y'$, satisfies the definition of the complement of $(x + y)$, defined as $(x + y)'$ by DeMorgan's Law. To show this we need to show that $A + A' = 1$ and $A \cdot A' = 0$ with $A = x + y$ and $A' = x' \cdot y'$. This proves that $x' \cdot y' = (x + y)'$.

Part 1: Show $x + y + x' \cdot y' = 1$.

$x + y + x' \cdot y'$	
$= (x + y + x')(x + y + y')$	$X + YZ = (X + Y)(X + Z)$ (Distributive Law)
$= (x + x' + y)(x + y + y')$	$X + Y = Y + X$ (Commutative Law)
$= (1 + y)(x + 1)$	$X + X' = 1$
$= 1 \cdot 1$	$1 + X = 1$
$= 1$	$1 \cdot X = 1$

Part 2: Show $(x + y) \cdot x' \cdot y' = 0$.

$(x + y) \cdot x' \cdot y'$	
$= (x \cdot x' \cdot y' + y \cdot x' \cdot y')$	$X(Y + Z) = XY + XZ$ (Distributive Law)
$= (x \cdot x' \cdot y' + y \cdot y' \cdot x')$	$XY = YX$ (Commutative Law)
$= (0 \cdot y' + 0 \cdot x')$	$X \cdot X' = 0$
$= (0 + 0)$	$0 \cdot X = 0$
$= 0$	$X + 0 = X$ (With $X = 0$)

Boolean Function Evaluation

$$F1 = xy\bar{z}$$
$$F2 = x + \bar{y}z$$
$$F3 = \bar{x}\bar{y}\bar{z} + \bar{x}\,y\,z + x\bar{y}$$
$$F4 = x\bar{y} + \bar{x}\,z$$

x	y	z	F1	F2	F3	F4
0	0	0	0	0		
0	0	1	0	1		
0	1	0	0	0		
0	1	1	0	0		
1	0	0	0	1		
1	0	1	0	1		
1	1	0	1	1		
1	1	1	0	1		

Expression Simplification

- An application of Boolean algebra
- Simplify to contain the smallest number of literals (complemented and uncomplemented variables):

$$AB + \overline{A}CD + \overline{A}BD + \overline{A}C\overline{D} + ABCD$$

= AB + ABCD + A'CD + A'CD' + A'BD

= AB + AB(CD) + A'C (D + D') + A'BD

= AB + A'C + A'BD

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

= B (A + D) + A'C 5 literals

Complementing Functions

- Use DeMorgan's Theorem to complement a function:

  1. Interchange AND and OR operators
  2. Complement each constant value and literal

- Examples
  - Complement $F = x'yz' + xy'z'$
  - Complement $G = (a' + bc)d' + e$

Complement example

- Example 2.5

  $F = wx'y + xy' + wxz$

  $F' =$

- Example 2.6

  $f = ab'(c + d'e) + a'bc'$

  $f' = [ab'(c + d'e) + a'bc']'$ using [P11a], [P11b]

  $= [ab'(c + d'e)]'[a'bc']'$

  $= [a' + b + (c + d'e)'][a + b' + c]$

  $= [a' + b + c'(d'e)'][a + b' + c]$

  $= [a' + b + c'(d + e')][a + b' + c]$

Boolean Functions

■ Truth Table for $F_1 = xy + xy'z + x'yz$
its Complement $F_1' = (x'+y')(x'+y+z')(x+y'+z')$

	VARIABLE VALUES			FUNCTION VALUES	
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

ex) $F_1 = 1$

⇒ x=1 and y=1, or if x=1, y=0, and z=1, or if x=0, y=1, and z=1.

• As a general rule, the truth table for any Boolean function of n variable has 2^n rows.

Implementation of Functions with AND, OR, and NOT Gates

■ Block diagram

$f = x'yz' + x'yz + xy'z' + xy'z + xyz$

• Block diagram of *f* in sum of standard products form.

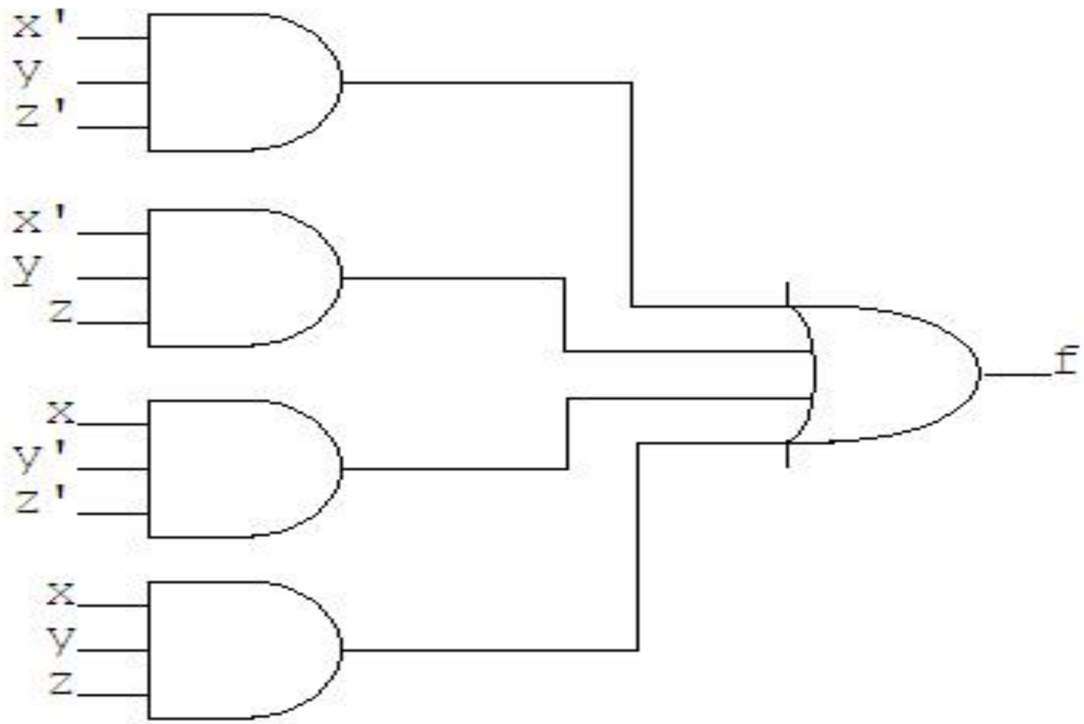


Figure 2.7 in text

Canonical Forms - Minterm

■ Minterm

- Let i = $b_{n-1}...b_0$ be a binary number
- A minterm of variables $x_{n-1,} x_{n-2},..., x_0$

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

$$y_k = \{ \begin{matrix} x_k & \textit{if } bk = 1 \quad \textit{all } k \quad 0 \le k \le n-1 \\ x_k' & \textit{if } bk = 0 \end{matrix}$$

- All minterms of the three binary variables, x,y, and z

x y z	MINTERMS	NOTATION
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

Canonical Forms - Minterm

$F_1 = m_3 + m_5 + m_6 + m_7$ = x'yz+xy'z+xyz'+xyz *1-minterms*

$F_1' = m_0 + m_1 + m_2 + m_4$ = x'y'z'+x'y'z+x'yz'+xy'z' *0-minterms*

- An important property of Boolean algebra :
  Any Boolean function can be expressed as a sum of its 1-minterms.

- Sum-of-minterms form
  F (list of variables) = Σ (list of 1-minterm indices)

  ex) $F_1(x,y,z) = \Sigma(3,5,6,7)$
  $F_1'(x,y,z) = \Sigma(0,1,2,4)$

Sum of minterm expressions

EXAMPLE 3.1 Sum-of-minterms expansion

PROBLEM Express the Boolean function F=x+yz as a sum of minterms

SOLUTION

F = x+yz
= x(y+y')(z+z')+(x+x')yz
= xyz+xy'z+xyz'+xy'z'+xyz+x'yz

After removing duplicates,
F = x'yz+xy'z'+xy'z+xyz'+xyz
= $m_3 + m_4 + m_5 + m_6 + m_7$
= $\sum (3,4,5,6,7)$

EXAMPLE 3.2 Conversion to a sum of minterms

PROBLEM Convert the Boolean function F=x+yz into a sum of minterms by using a truth table.
(Table 3.10)

SOLUTION

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

Truth Table for
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**

Canonical Forms - Maxterm

- Eight maxterms of the three binary variables, x,y, and z

x	y	z	MAXTERMS	NOTATION
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

- The expression for F_1=x’yz+xy’z+xyz’+xyz

$(F_1)'$ = (x’yz+xy’z+xyz’+xyz)’

= (x+y’+z’)(x’+y+z’)(x’+y’+z)(x’+y’+z’)

= $M_3M_5M_6M_7$

Similarly, by complementing the expression

$F_1 = (F_1')'$

= (x’y’z’+x’y’z+x’yz’+xy’z’)’

= (x+y+z)(x+y+z’)(x+y’+z)(x’+y+z)

= $M_0M_1M_2M_4$

Product of maxterm

- An second unique property of Boolean algebra :
  Any Boolean function can be expressed as a product of its 0-maxterms.

- Product of maxterms form
  $$F\text{ (list of variables)} = \prod \text{(list of 0-maxterm indices)}$$

- A general conversion procedure to convert from on canonical form to another, interchange the symbols Σ and Π, and list the numbers that were excluded from the original form.

EXAMPLE 3.3 Product-of-maxterms expansion

PROBLEM Derive the product-of-maxterms form for the Boolean function

$F=x'y'+xz$

SOLUTION

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

Product of maxterm

into two maxterms

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)

Finally,

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

$= M_2 M_3 M_4 M_6 = \prod(2,3,4,6)$

EXAMPLE 3.4 Conversion to product of maxterms

PROBLEM Convert the Boolean function F= x'y'+xz into the product-of-maxterms form.

SOLUTION

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

minterms m_0, m_1, m_5, m_7 from Table

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

the missing minterms m_2, m_3, m_4, m_6

Truth Table for
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**

Minterm and Maxterm Expansions

- 3 variables

Row No.	A B C	Minterms	Maxterms
0	0 0 0	$A'B'C' = m_0$	$A + B + C = M_0$
1	0 0 1	$A'B'C = m_1$	$A + B + C' = M_1$
2	0 1 0	$A'BC' = m_2$	$A + B' + C = M_2$
3	0 1 1	$A'BC = m_3$	$A + B' + C' = M_3$
4	1 0 0	$AB'C' = m_4$	$A' + B + C = M_4$
5	1 0 1	$AB'C = m_5$	$A' + B + C' = M_5$
6	1 1 0	$ABC' = m_6$	$A' + B' + C = M_6$
7	1 1 1	$ABC = m_7$	$A' + B' + C' = M_7$

Minterm and Maxterm Expansions

$$f(A,B,C) = m_3 + m_4 + m_5 + m_6 + m_7$$

$$f' = m_0 + m_1 + m_2 = \sum m(0,1,2)$$

$$f(A,B,C) = M_0 M_1 M_2 \longrightarrow f' = \prod M(3,4,5,6,7) = M_3 M_4 M_5 M_6 M_7$$

- *Minterm* and *Maxterm* expansions are complement each other

$$f' = (m_3 + m_4 + m_5 + m_6 + m_7)' = m'_3\, m'_4\, m'_5\, m'_6\, m'_7 = M_3 M_4 M_5 M_6 M_7$$

$$f' = (M_0 M_1 M_2)' = M'_0 + M'_1 + M'_2 = m_0 + m_1 + m_2$$

Combinational Logic Design Using a Truth Table

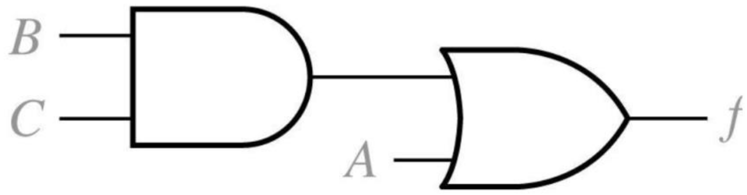
Original equation →

$$f = A'BC + AB'C' + AB'C + ABC' + ABC$$

Simplified equation →

$$f = A'BC + AB' + AB = A'BC + A = A + BC$$

Circuit realization →



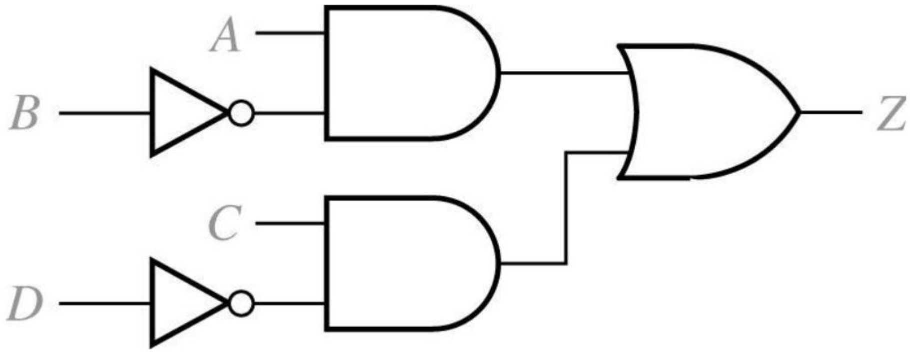
Conversion of English Sentences to Boolean Equations

1. The alarm will ring(Z) iff the alarm switch is turned on(A) ***and*** the door is not closed(B’), ***or*** it is after 6PM(C) and window is not closed(D’)

2. Boolean Equation

$$Z = AB' + CD'$$

3. Circuit realization



From The Truth Table to Algebraic Expressions

- Truth table
  - To find a minimum POS expression
    - Manipulate the previous POS expression to obtain f = (A + B + C)(A′ + B′)
    - Simplify the SOP expression for f′ and then use DeMorgan to convert it to a POS expression.

    Both approaches produce the same result.

  - How many different functions of n variables are there?

a	b	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

Variables	Terms
1	4
2	16
3	256
4	65,536
5	4,294,967,296

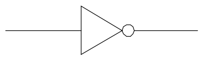
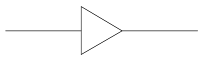
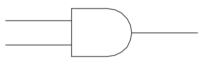
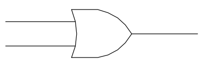
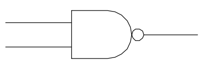
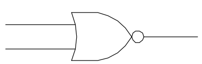
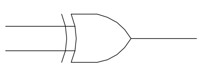
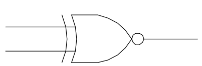
Number of functions of n variables

Other logic operations

NAME	OPERATOR SYMBOL	FUNCTION VALUES FOR x,y= 00	01	10	11	ALGEBRAIC EXPRESSION	COMMENT
Zero		0	0	0	0	$F_0 = 0$	Binary constant 0
AND	$x \bullet y$	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	1	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

Boolean Expressions for the 16 Functions of Two Variables

Basic logic library

NAME	GRAPHIC SYMBOL	FUNCTIONAL EXPRESSION	COST(NUMBER OF TRANSISTORS)	GATE DELAY(NS)	Truth table
Inverter		**F = x'**	2	1	x | F 0 | 1 1 | 0
Driver		**F = x**	4	2	x | F 0 | 0 1 | 1
AND		**F = xy**	6	2.4	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		**F = x+y**	6	2.4	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NAND		**F = (xy)'**	4	1.4	x y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		**F = (x+y)'**	4	1.4	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR		**F = x⊕y**	14	4.2	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
XNOR		**F = x⊙y**	12	3.2	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Exclusive-OR and Equivalence Operations

Exclusive-OR

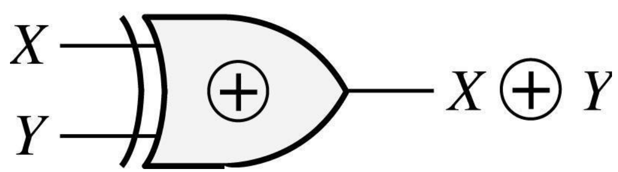
$$0 \oplus 0 = 0 \quad 0 \oplus 1 = 1$$

$$1 \oplus 0 = 1 \quad 1 \oplus 1 = 0$$

Truth Table

XY	$X \oplus Y$
0 0	0
0 1	1
1 0	1
1 1	0

Symbol



Exclusive-OR and Equivalence Operations

Theorems for Exclusive-OR:

$$X \oplus 0 = X$$

$$X \oplus 1 = X'$$

$$X \oplus X = 0$$

$$X \oplus X' = 1$$

$$X \oplus Y = Y \oplus X \text{ (commutative law)}$$

$$(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z) = X \oplus Y \oplus Z \text{ (associative law)}$$

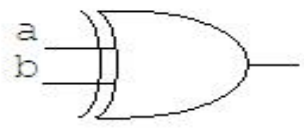
$$X(Y \oplus Z) = XY \oplus XZ \text{ (distributive law)}$$

$$(X \oplus Y)' = X \oplus Y' = X' \oplus Y = XY + X'Y'$$

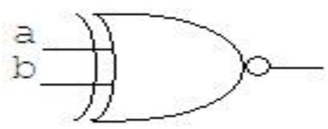
NAND, NOR, AND EXCLUSIVE-OR GATES

■ Three other gates

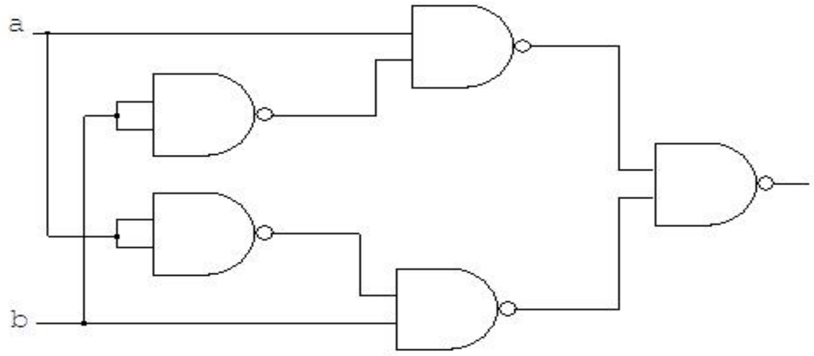
- Exclusive-OR gates.
  - An Exclusive-OR gate

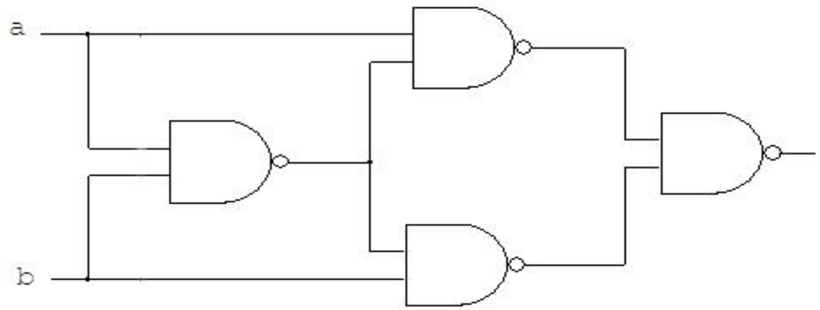


  - An Exclusive-NOR gate

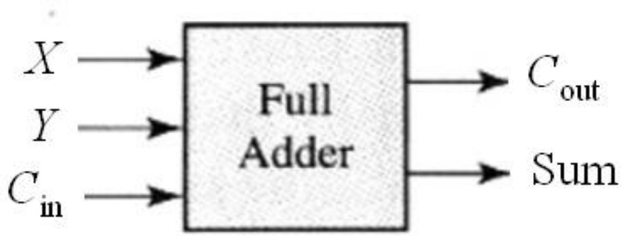


✓ Exclusive-OR gate implements the expression
- $(a \oplus b) = a'b + ab'$
- $(a \oplus b)' = a'b' + ab$





Binary Adders and Subtracters



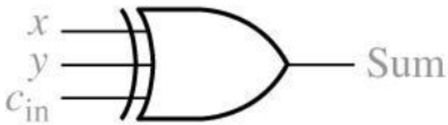
Truth Table for a Full Adder

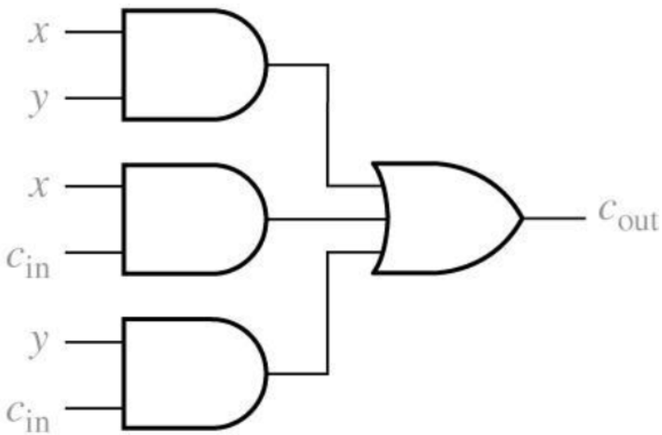
X	Y	C_{in}	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Full adder

$$
\begin{aligned}
Sum &= X'Y'C_{in} + X'YC'_{in} + XY'C'_{in} + XYC_{in} \\
&= X'(Y'C_{in} + YC'_{in}) + X(Y'C'_{in} + YC_{in}) \\
&= X'(Y \oplus C_{in}) + X(Y \oplus C_{in})' = X \oplus Y \oplus C_{in}
\end{aligned}
$$

$$
\begin{aligned}
C_{out} &= X'YC_{in} + XY'C_{in} + XYC'_{in} + XYC_{in} \\
&= (X'YC_{in} + XYC_{in}) + (XY'C_{in} + XYC_{in}) + (XYC'_{in} + XYC_{in}) \\
&= YC_{in} + XC_{in} + XY
\end{aligned}
$$





Full adder design

EXAMPLE Full-adder design

PROBLEM Design a full adder based on the specifications given in table, using the basic logic library that was given in Table.
The primary goal is to minimize the propagation delay from c_i to c_{i+1}, while the secondary goal is to use the smallest possible number of transistors.

SOLUTION

$$s_i = x'_i y'_i c_i + x'_i y_i c'_i + x_i y'_i c'_i + x_i y_i c_i$$
$$= (x'_i y_i + x_i y'_i) c'_i + (x'_i y'_i + x_i y_i) c_i$$
$$= (x_i \oplus y_i) c'_i + (x_i \odot y_i) c_i$$
$$= (x_i \oplus y_i) c'_i + (x_i \oplus y_i)' c_i$$
$$= (x_i \oplus y_i) \oplus c_i$$

$$c_{i+1} = x_i y_i c'_i + x_i y_i c_i + x'_i y_i c_i + x_i y'_i c_i$$
$$= x_i y_i (c'_i + c_i) + c_i (x'_i y_i + x_i y'_i)$$
$$= x_i y_i + c_i (x_i \oplus y_i)$$

Addition of Binary Digits

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**

Full adder design

■ Modify the expression (faster, less expensive NAND and NOR gates)

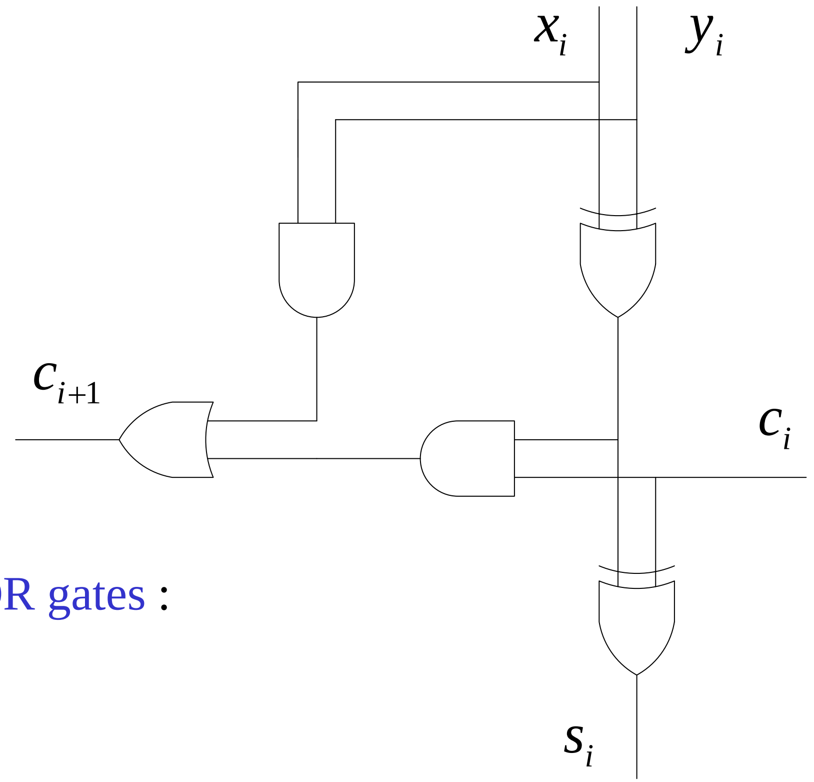
• De Morgan's theorems :

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

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

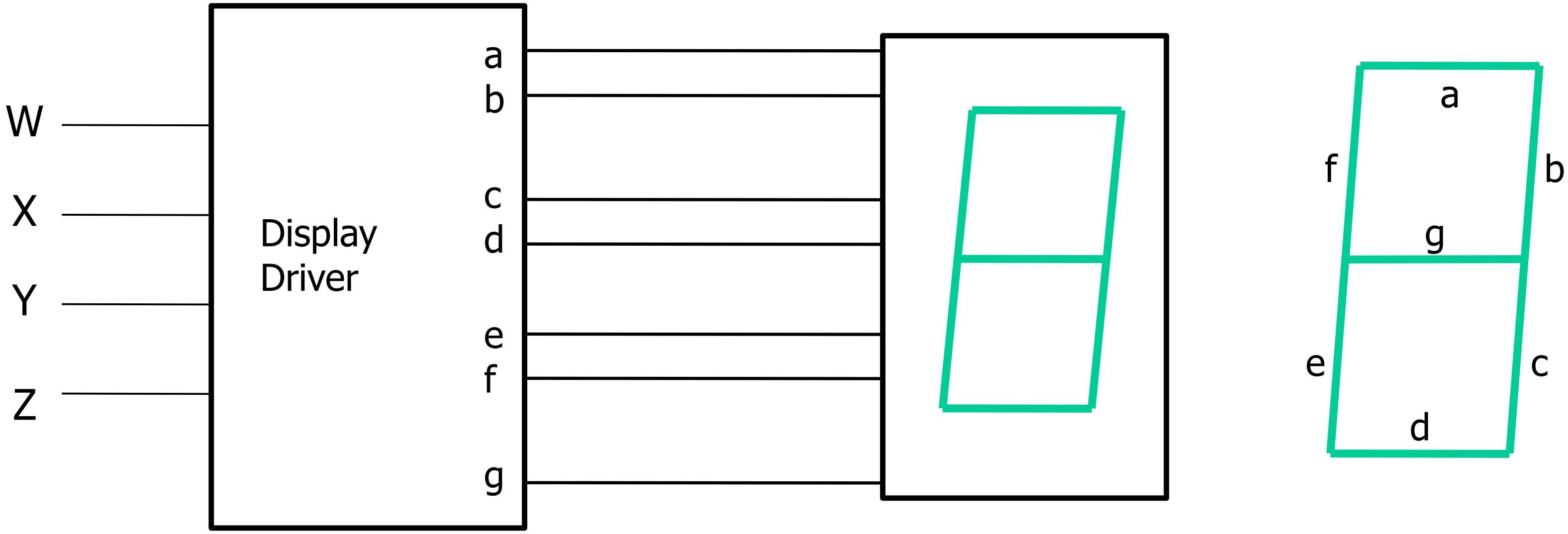
In addition, with two NANDs and one OR gates :

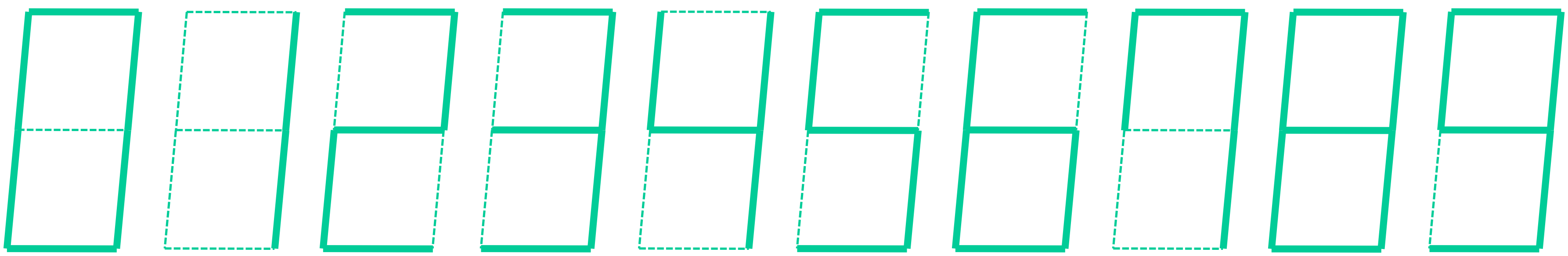
$$(x_i \odot y_i) = x_i y_i + x'_i y'_i$$
$$= ((x_i y_i)'(x'_i y'_i)')'$$
$$= ((x_i y_i)'(x_i + y_i))'$$



A seven-segment display

■ The Development of Truth Tables





Truth table for seven segment display

■ The Development of Truth Tables

Digit	W	X	Y	Z	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	X	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	X	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	X	0	1	1
-	1	0	1	0	X	X	X	X	X	X	X
-	1	0	1	1	X	X	X	X	X	X	X
-	1	1	0	0	X	X	X	X	X	X	X
-	1	1	0	1	X	X	X	X	X	X	X
-	1	1	1	0	X	X	X	X	X	X	X
-	1	1	1	1	X	X	X	X	X	X	X