Simplification of Boolean Functions

Logic Design of Digital Systems (300-1209) section 1

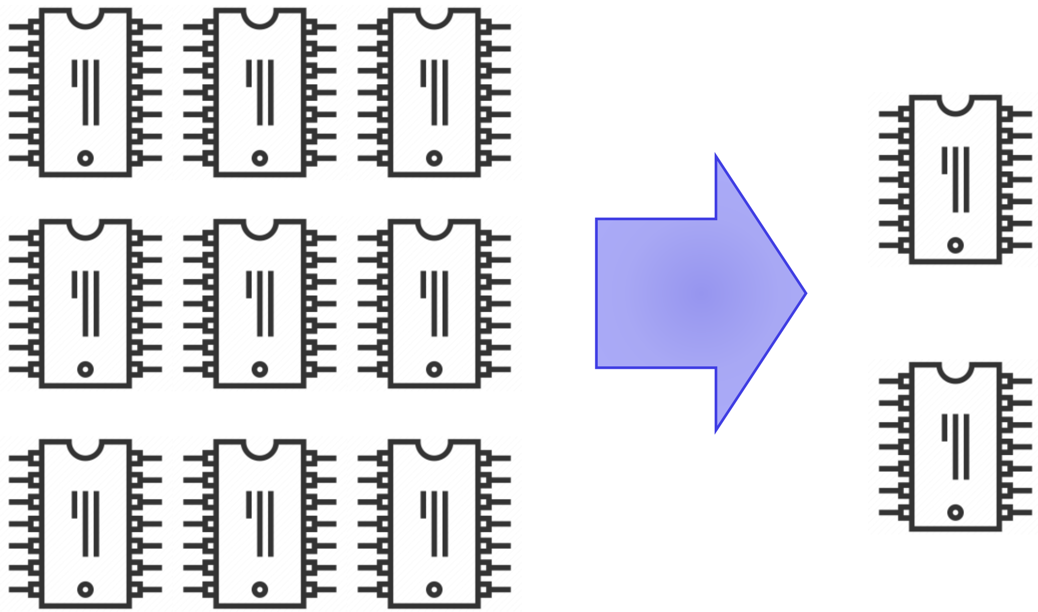
LECTURE 04

KRISADA PHROMSUTHIRAK

$$f(A,B,C) = \sum(1,4,7)$$

Why we need to simplify Boolean Functions

Boolean expressions are realized in practice using electronic circuits known as gates. Simplifying a Boolean expression reduces the number of gates, thus reducing the cost, size and area of the integrated circuit or chip.



The Map method

The complexity of the digital logic gates that implement a Boolean function is directly related to the complexity of the algebraic expression from which the function is implemented.

Although the truth table representation of a function is unique, expressed algebraically, it can appear in many different forms

$$xy + xz + yz' = xy(z+z') + xz\cdot(y+y') + yz'(x+x')$$
$$= xyz + xyz' + xyz + xy'z + xyz' + x'yz'$$
$$= xyz + xyz' + xy'z + x'yz'$$
$$= (xyz + xy'z) + (xyz' + x'yz')$$
$$= [xz(y+y')] + [yz'(x+x')]$$
$$= xz + yz'$$

#

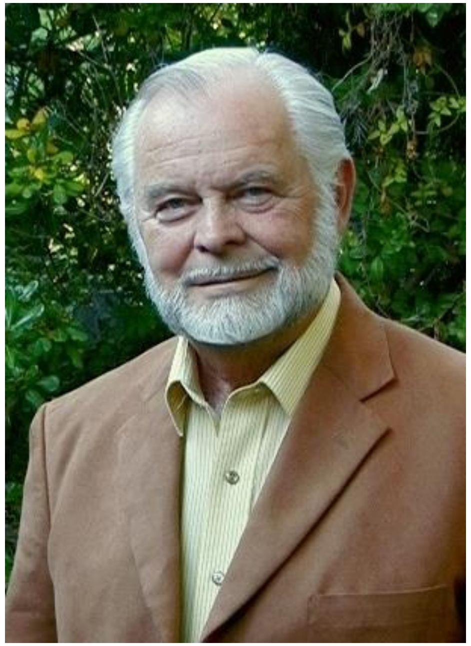
Postulate 2: Identity	(a) $x + 0 = x$	(b) $x \cdot 1 = x$
Postulate 3: Commutative	(a) $x + y = y + x$	(b) $xy = yx$
Postulate 4: Distributive	(a) $x(y+z) = xy + xz$	(b) $x + yz = (x+y)(x+z)$
Postulate 5: Complement	(a) $x + x' = 1$	(b) $x \cdot x' = 0$
Theorem 1:	(a) $x + x = x$	(b) $x \cdot x = x$
Theorem 2:	(a) $x + 1 = 1$	(b) $x \cdot 0 = 0$
Theorem 3: Involution	$(x')' = x$	
Theorem 5: DeMorgan's	(a) $(x+y)' = x'y'$	(b) $(xy)' = x' + y'$
Theorem 6: Absorption	(a) $x + xy = x$	(b) $x(x+y) = x$

The method in previous lecture is awkward because it lacks specific rules to predict each succeeding step in the manipulative process.

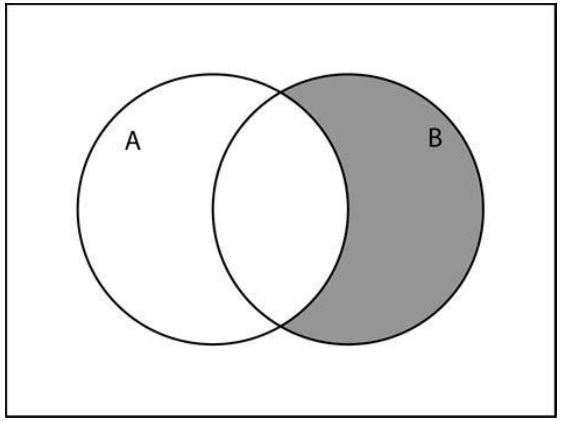
The Map method

The map method provides a simple straightforward procedure for minimizing Boolean functions. This method may be regarded either as a **pictorial form of a truth table** or as an extension of the Venn diagram.

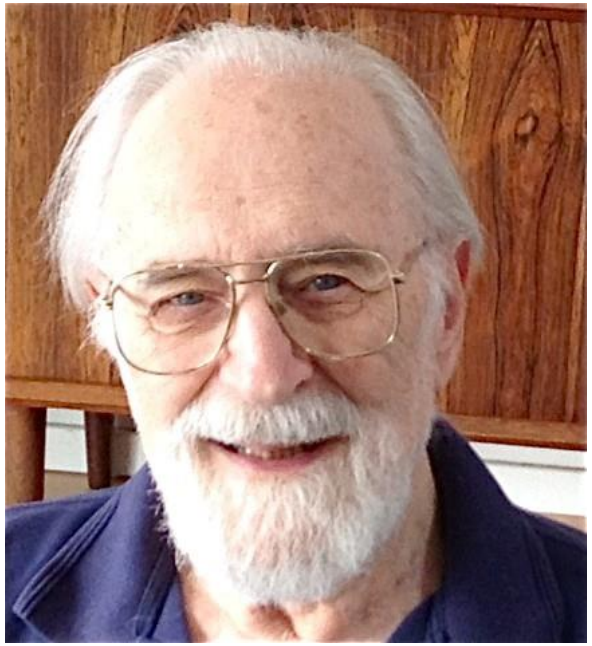
The map method, first proposed by *Edward W. Veitch* and slightly modified by *Maurice Karnaugh*, is also known as the "Veitch diagram" or the "Karnaugh map."



Edward W. Veitch



https://www.ithistory.org/



Maurice Karnaugh

Two- and Three-Variable Maps

Review from the previous lecture

#	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	$ABC = m_7$	$A' + B' + C' = M_7$

Sum of Minterms

$$f(A,B,C) = A'B'C' + A'BC'$$

$$= m_0 + m_2$$

$$= \sum(0,2)$$

Product of Maxterms

$$f(A,B,C) = (A + B + C) \cdot (A + B' + C)$$

$$= M_0 \cdot M_2$$

$$= \prod(0,2)$$

Two- and Three-Variable Maps

We can change the expression of Boolean algebra into table form (also known as Karnaugh map). The Karnaugh map reduces the need for extensive calculations by taking advantage of humans' pattern-recognition capability.

X \ Y	0	1
0	$x'y'$	$x'y$
1	xy'	xy

$$x' \rightarrow 0$$
$$y' \rightarrow 0$$

$$x \rightarrow 1$$
$$y \rightarrow 1$$

Two- and Three-Variable Maps

<u>Example</u> 1 Express the following Boolean function in K-Map

$$f(x, y) = x'y + xy' + xy$$

$$f(x, y, z) = x'yz + x'yz' + xy'z' + xy'z$$

Two- and Three-Variable Maps

To understand the usefulness of the map for simplifying Boolean functions, we must recognize the basic property possessed by adjacent squares. Any two adjacent squares in the map differ by only one variable which is primed in one square and unprimed in the other.

Example 2-1 Minimizing Boolean function with Map

$$f(x, y, z) = x'yz + x'yz' + xy'z' + xy'z$$

#	X	Y	Z	Out #1	Out #2
0	0	0	0	F	
1	0	0	1	F	
2	0	1	0	T	
3	0	1	1	T	
4	1	0	0	T	
5	1	0	1	T	
6	1	1	0	F	
7	1	1	1	F	

Two- and Three-Variable Maps

<u>Example</u> 2-2 Minimizing Boolean function with Map

$$f(x, y, z) = x'yz + xy'z' + xyz + xyz'$$

Two- and Three-Variable Maps

<u>Example</u> 2-3 Minimizing Boolean function with Map

$$f(A, B, C) = A'C + A'B + AB'C + BC$$

Two- and Three-Variable Maps

<u>Example</u> 2-4 Minimizing Boolean function with Map

$$f(A, B, C) = \sum (0,2,4,5,6)$$

Four-Variable Maps

The map minimization of four-variable Boolean functions is similar to the method used to minimize three-variable functions. Adjacent squares are defined to be squares next to each other. In addition, the map is considered to lie on a surface with the top and bottom edges, as well as the right and left edges, touching each other to form adjacent squares.

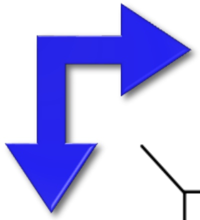
wx \ yz	00	01	11	10
00	$w'x'y'z'$	$w'x'y'z$	$w'x'yz$	$w'x'yz'$
01	$w'xy'z'$	$w'xy'z$	$w'xyz$	$w'xyz'$
11	$wxy'z'$	$wxy'z$	$wxyz$	$wxyz'$
10	$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

One square: represent a term of four literals (one minterm).
Two adjacent: represent a term of three literals.
Four adjacent: represent a term of two literals.
Eight adjacent: represent a term of one literals.
Sixteen adjacent: represent the function equal to 1.

Four-Variable Maps

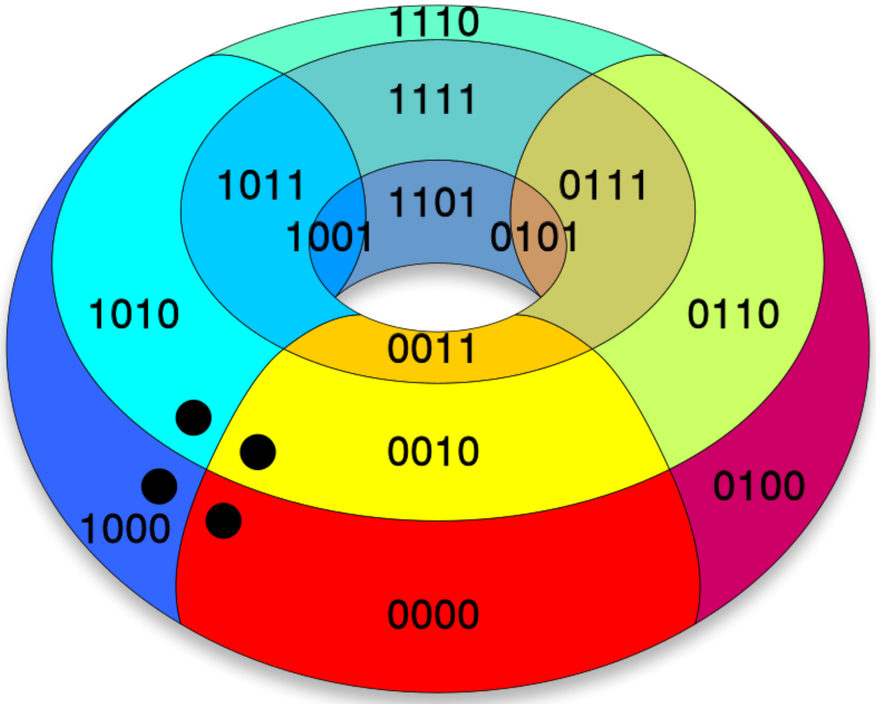
Can switch



CD \ AB	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

ABCD	ABCD
0000 - 0	1000 - 8
0001 - 1	1001 - 9
0010 - 2	1010 - 10
0011 - 3	1011 - 11
0100 - 4	1100 - 12
0101 - 5	1101 - 13
0110 - 6	1110 - 14
0111 - 7	1111 - 15

0000	0100	1100	1000
0001	0101	1101	1001
0011	0111	1111	1011
0010	0110	1110	1010



Four-Variable Maps

<u>Example</u> 2-5 Minimizing Boolean function with Map

$$f(w,x,y,z) = \sum(0,1,2,4,5,6,8,9,12,13,14)$$

ABCD	ABCD
0000 - 0	1000 - 8
0001 - 1	1001 - 9
0010 - 2	1010 - 10
0011 - 3	1011 - 11
0100 - 4	1100 - 12
0101 - 5	1101 - 13
0110 - 6	1110 - 14
0111 - 7	1111 - 15

Four-Variable Maps

<u>Example</u> 2-6 Minimizing Boolean function with Map

$$f(A,B,C,D) = A'B'C' + B'CD' + A'BCD' + AB'C'$$

Five- and Six-Variable Maps

Maps of more than four variables are not as simple to use. The number of squares becomes excessively large and the geometry for combining adjacent squares becomes more involved. The number of squares is always equal to the number of minterms.

$AB \backslash CDE$	000	001	011	010	110	111	101	100
00	0	1	3	2	6	7	5	4
01	8	9	11	10	14	15	13	12
11	24	25	27	26	30	31	29	28
10	16	17	19	18	22	23	21	20

Bin	#	Bin	#	Bin	#	Bin	#
00000	0	01000	8	10000	16	11000	24
00001	1	01001	9	10001	17	11001	25
00010	2	01010	10	10010	18	11010	26
00011	3	01011	11	10011	19	11011	27
00100	4	01100	12	10100	20	11100	28
00101	5	01101	13	10101	21	11101	29
00110	6	01110	14	10110	22	11110	30
00111	7	01111	15	10111	23	11111	31

27 ⟷ 31

11011 ⟷ 11111

Five- and Six-Variable Maps

<u>Example</u> 2-7 Minimizing Boolean function with Map

$$f(A,B,C,D,E) = \sum(0,2,4,6,9,11,13,15,17,21,25,27,29,31)$$

$AB \backslash CDE$	000	001	011	010	110	111	101	100
00	1			1	1			1
01		1	1			1	1	
11		1	1			1	1	
10		1					1	

$AB \backslash CDE$	000	001	011	010	110	111	101	100
00	1			1	1			1
01		1	1			1	1	
11		1	1			1	1	
10		1					1	

Five- and Six-Variable Maps

ABC \ DEF	000	001	011	010	110	111	101	100
000	0	1	3	2	6	7	5	4
001	8	9	11	10	14	15	13	12
011	24	25	27	26	30	31	29	28
010	16	17	19	18	22	23	21	20
110	48	49	51	50	54	55	53	52
111	56	57	59	58	62	63	61	60
101	40	41	43	42	46	47	45	44
100	32	33	35	34	38	39	37	36

D C B C A E F F

Five- and Six-Variable Maps

	Number of adjacent squares	Number of literals in a term in an n-variable map					
k	2^k	$n = 2$	$n = 3$	$n = 4$	$n = 5$	$n = 6$	$n = 7$
0	1	2	3	4	5	6	7
1	2	1	2	3	4	5	6
2	4	0	1	2	3	4	5
3	8		0	1	2	3	4
4	16			0	1	2	3
5	32				0	1	2
6	64					0	1

Product of Sum Simplification

The minimized Boolean functions derived from the map in all previous examples were expressed in the sum of products form. With a minor modification, the product of sums form can be obtained.

If we mark the empty squares by 0's and combine them into valid adjacent squares, we obtain a simplified expression of the complement of the function (F'). The complement of F' gives us back the function F. Because of the generalized DeMorgan's theorem, the function so obtained is automatically in the product of sums form.

Sum of Minterms

$$f(A,B,C) = A'B'C' + A'BC'$$

$$= m_0 + m_2$$

$$= \sum(0,2)$$

Product of Maxterms

$$f(A,B,C) = (A + B + C) \cdot (A + B' + C)$$

$$= M_0 \cdot M_2$$

$$= \prod(0,2)$$

Product of Sum Simplification

Example 3-1 Simplify the following Boolean function in **(a) sum of products** and (b) product of sums.

$$f(A,B,C,D) = \sum(0,1,2,5,8,9,10)$$

Product of Sum Simplification

<u>Example</u> 3-1 Simplify the following Boolean function in (a) sum of products and **(b) product of sums**.

$$f(A,B,C,D) = \sum(0,1,2,5,8,9,10)$$

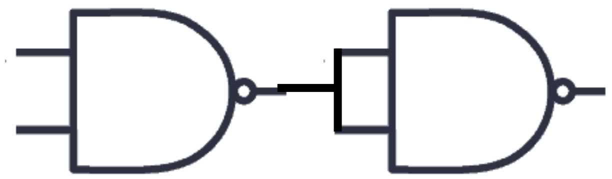
Product of Sum Simplification

<u>Example</u> 3-2 Simplify the following Boolean function in (a) sum of products and (b) product of sums.

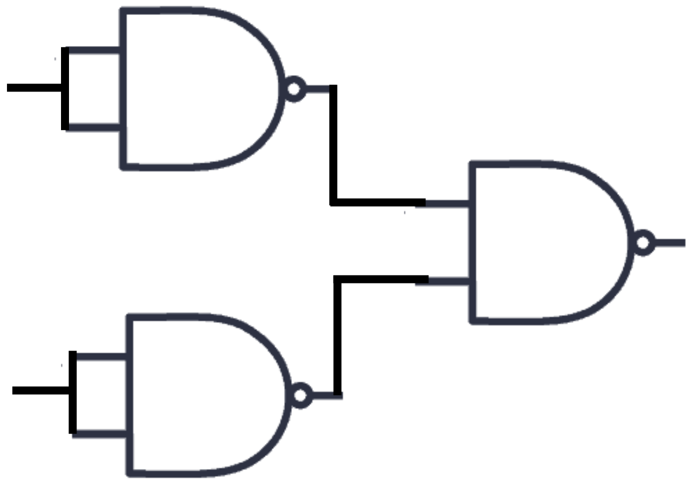
$$f(A, B, C, D) = \sum(\qquad\qquad\qquad\qquad)$$

NAND & NOR Implementation

NAND and NOR logic gates are known as **universal gates** because they can implement any Boolean logic without needing any other gate. They can be used to design any logic gate too. Moreover, they are widely used in ICs because they are easier and economical to fabricate.



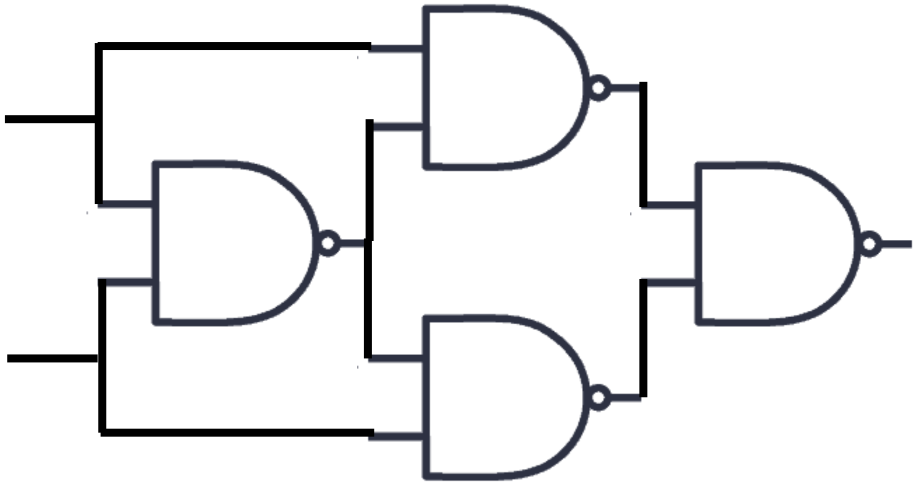
AND



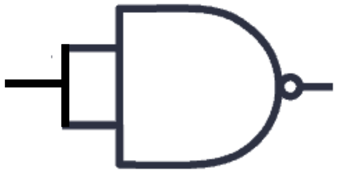
OR

NAND & NOR Implementation

NAND and NOR logic gates are known as **universal gates** because they can implement any Boolean logic without needing any other gate. They can be used to design any logic gate too. Moreover, they are widely used in ICs because they are easier and economical to fabricate.



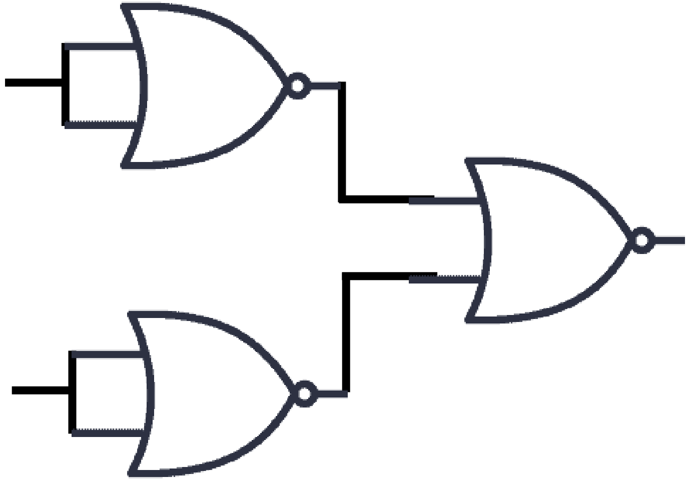
XOR



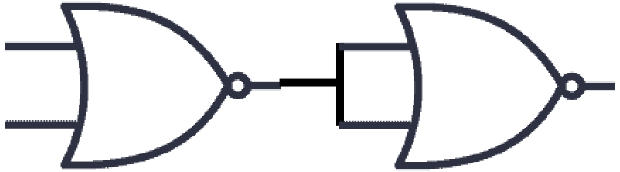
NOT

NAND & NOR Implementation

NAND and NOR logic gates are known as **universal gates** because they can implement any Boolean logic without needing any other gate. They can be used to design any logic gate too. Moreover, they are widely used in ICs because they are easier and economical to fabricate.



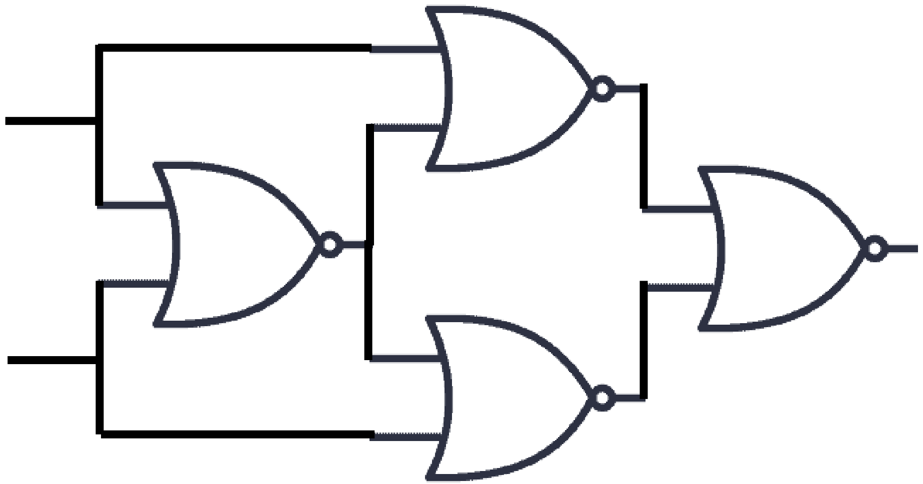
AND



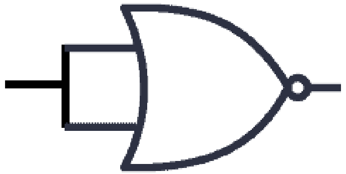
OR

NAND & NOR Implementation

NAND and NOR logic gates are known as **universal gates** because they can implement any Boolean logic without needing any other gate. They can be used to design any logic gate too. Moreover, they are widely used in ICs because they are easier and economical to fabricate.



XOR



NOT

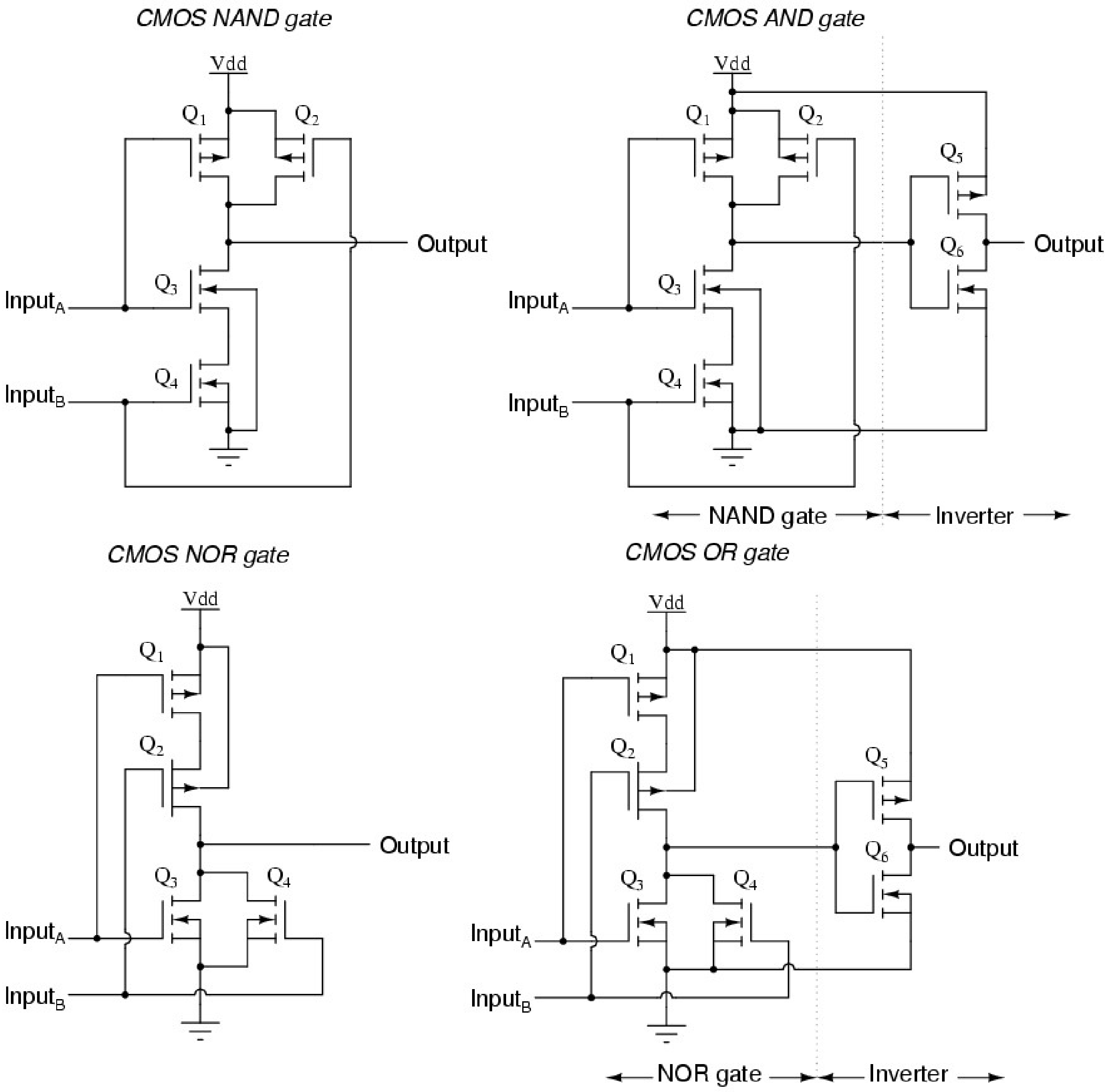
NAND & NOR Implementation

Why do we use NAND & NOR gates instead of using AND, OR gate?

by using NAND and NOR gates we save space and reduce the gate delay.

NAND / NOR	➔	4 MOSFETs
AND / OR	➔	6 MOSFETs

metal–oxide–semiconductor field-effect transistor (MOSFET)



Example Make a K-Map for the function $f = AB + A\bar{C} + C + AD + A\bar{B}CD$

Minimize it and realize the minimized expression in form **Product of Sum**

Try this!

Don’t care condition

The 1’s and 0’s in the map signify the combination of variables that makes the function equal to 1 or 0, respectively. However, there are applications where certain combinations of input variables never occur. So, we assign these variables as don't care variables.

<u>Example</u> 5-1 Simplify the Boolean function

$$f(w,x,y,z) = \sum(1,3,7,11,15)$$

with don’t care condition

$$d(w,x,y,z) = \sum(0,2,5)$$

$wx \backslash yz$	00	01	11	10
00	X	1	1	X
01		X	1	
11			1	
10			1	

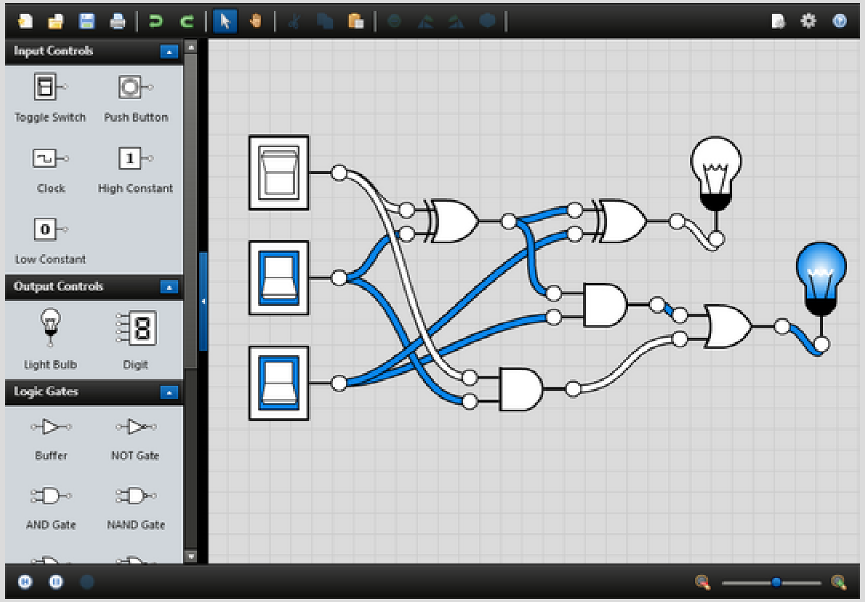
logic.ly

Pricing & Purchase Download the Free Trial Try Online

Teach logic gates + digital circuits effectively — with Logicly

- Design circuits quickly and easily with a modern and **intuitive user interface** with drag-and-drop, copy/paste, zoom & more.
- Take control of **debugging** by pausing the simulation and watching the signal propagate as you advance step-by-step.
- Don't worry about **multiple platforms** on student computers. Install on both Windows and macOS.

Buy Logicly Free Trial



https://logic.ly

Classwork Simplify Boolean function

Minimize it and realize the minimized expression.

$$f(A, B, C, D) = \sum(\qquad\qquad\qquad\qquad\qquad\qquad)$$

with don't care condition

$$d(w, x, y, z) = \sum(\qquad\qquad\qquad\qquad\qquad\qquad)$$


Simplicity is the
ultimate sophistication.
Leonardo da Vinci
quotefancy