

DIGITAL ELECTRONICS CIRCUIT(BCA 103)

**DEPARTMENT OF COMPUTER SCIENCE
PROGRAMME: BCA**



**CENTRAL UNIVERSITY OF ODISHA
KORAPUT**

DIGITAL LOGIC

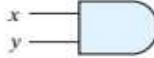



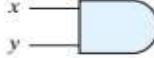

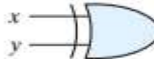
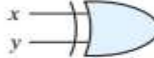
- Since Boolean functions are expressed in terms of AND, OR, and NOT operations, it is easier to implement a Boolean function with these type of gates.
- Factors to be weighed in considering the construction of other types of logic gates are:
 - (1) the feasibility and economy of producing the gate with physical components,
 - (2) the possibility of extending the gate to more than two inputs,
 - (3) the basic properties of the binary operator, such as commutativity and associativity, and
 - (4) the ability of the gate to implement Boolean functions alone or in conjunction with other gates.

- There are 2^{2n} functions for n binary variables.
- Thus, for two variables, $n = 2$, and the number of possible Boolean functions is 16.
- The 16 functions listed can be subdivided into three categories:
 1. **Two** functions that produce a constant 0 or 1.
 2. **Four** functions with unary operations: complement and transfer.
 3. **Ten** functions with binary operators that define eight different operations: AND, OR, NAND, NOR, exclusive-OR, equivalence, inhibition, and implication.
- Of the 16 functions, two are equal to a constant and four are repeated.
- There are only 10 functions left to be considered as candidates for logic gates.

•

- Two—inhibition and implication—are not commutative or associative and thus are impractical to use as standard logic gates.
- The other eight—complement, transfer, AND, OR, NAND, NOR, exclusive-OR, and equivalence—are used as standard gates in digital design.

DIGITAL LOGIC GATES

Name	Graphic symbol	Algebraic function	Truth table															
AND		$F = x \cdot y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = x + y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Inverter		$F = x'$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	F	0	1	1	0									
x	F																	
0	1																	
1	0																	
Buffer		$F = x$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	x	F	0	0	1	1									
x	F																	
0	0																	
1	1																	
NAND		$F = (xy)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = (x + y)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

- Each gate has one or two binary input variables, designated by x and y , and one binary output variable, designated by F .
- The inverter circuit inverts the logic sense of a binary variable, producing the NOT, or complement, function.
- The small circle in the output of the graphic symbol of an inverter (referred to as a *bubble*) designates the logic complement.
- The triangle symbol by itself designates a buffer circuit.
- A buffer produces the *transfer* function, but does not produce a logic operation, since the binary value of the output is equal to the binary value of the input.
- This circuit is used for power amplification of the signal and is equivalent to two inverters connected in cascade.

- The NAND function is the complement of the AND function, as indicated by a graphic symbol that consists of an AND graphic symbol followed by a small circle.
- The NOR function is the complement of the OR function and uses an OR graphic symbol followed by a small circle.
- NAND and NOR gates are used extensively as standard logic gates and are in fact far more popular than the AND and OR gates.
- This is because NAND and NOR gates are easily constructed with transistor circuits and because digital circuits can be easily implemented with them.
- The exclusive-OR gate has a graphic symbol similar to that of the OR gate, except for the additional curved line on the input side.
- The equivalence, or exclusive-NOR, gate is the complement of the exclusive-OR, as indicated by the small circle on the output side of the graphic symbol.

EXTENSION TO MULTIPLE INPUTS

- The gates, except for the inverter and buffer—can be extended to have more than two inputs.
- A gate can be extended to have multiple inputs if the binary operation it represents is commutative and associative.
- The AND and OR operations, defined in Boolean algebra, possess these two properties.

- For the OR function, we have

$$x + y = y + x \text{ (commutative) and}$$

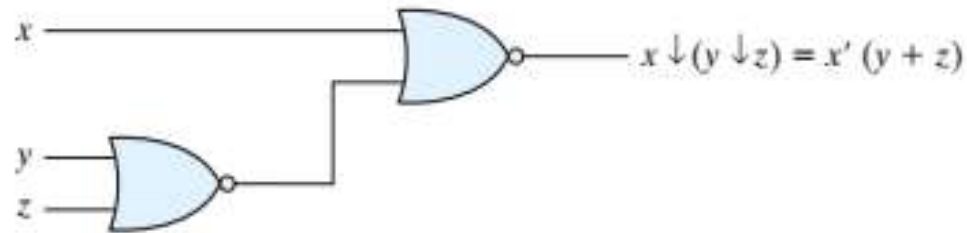
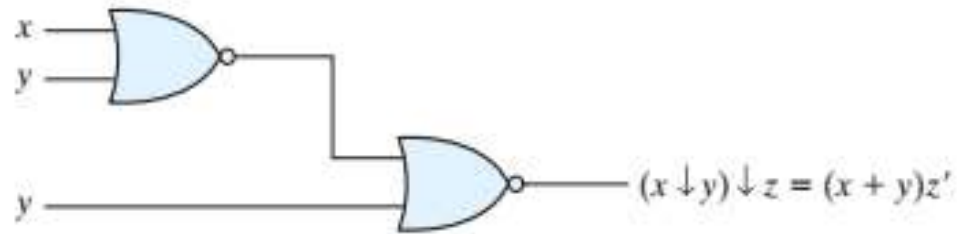
$$(x + y) + z = x + (y + z) = x + y + z \text{ (associative)}$$

which indicates that the gate inputs can be interchanged and that the OR function can be extended to three or more variables.

-

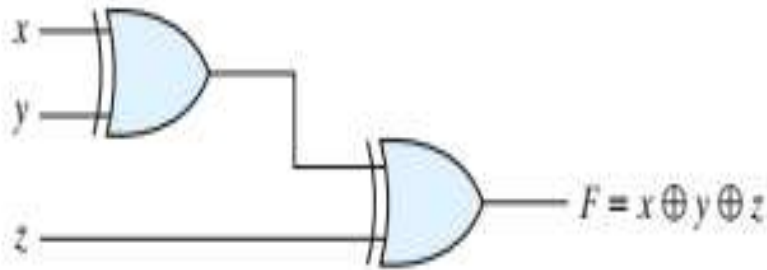
- The NAND and NOR functions are commutative, and their gates can be extended to have more than two inputs, provided that the definition of the operation is modified slightly.
- The difficulty is that the NAND and NOR operators are not associative (i.e., $(x \downarrow y) \downarrow z \neq x \downarrow (y \downarrow z)$)
- i.e. $(x \downarrow y) \downarrow z = [(x + y)' + z]' = (x + y)z' = xz' + yz'$
 $x \downarrow (y \downarrow z) = [x + (y + z)']]' = x'(y + z) = x'y + x'z$
- To overcome this difficulty, we define the multiple NOR (or NAND) gate as a complemented OR (or AND) gate. Thus, by definition, we have:
 NOR : $x \downarrow y \downarrow z = (x + y + z)'$
 NAND: $x \uparrow y \uparrow z = (xyz)'$
- In writing cascaded NOR and NAND operations, one must use the correct parentheses to signify the proper sequence of the gates.

Demonstrating the non-associativity of the NOR operator

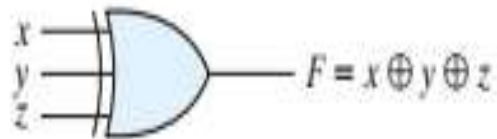


- The exclusive-OR and equivalence gates are both commutative and associative and can be extended to more than two inputs.
- However, multiple-input exclusive-OR gates are uncommon from the hardware standpoint.
- The definition of the function must be modified when extended to more than two variables.
- Exclusive-OR is an *odd* function (i.e., it is equal to 1 if the input variables have an odd number of 1's).
- The output (Exclusive-OR) F is equal to 1 if only one input is equal to 1 or if all three inputs are equal to 1 (i.e., when the total number of 1's in the input variables is *odd*).

Three-input exclusive-OR gate



(a) Using 2-input gates



(b) 3-input gate

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

(c) Truth table

Gate-Level Minimization

- *Gate-level minimization* is the design task of finding an optimal gate-level implementation of the Boolean functions describing a digital circuit.
- This task is well understood, but is difficult to execute by manual methods when the logic has more than a few inputs.
- Fortunately, computer-based logic synthesis tools can minimize a large set of Boolean equations efficiently and quickly.
- 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, when it is expressed algebraically it can appear in many different, but equivalent, forms.

The Map Method

- The map method provides a simple, straightforward procedure for minimizing Boolean functions.
- This method may be regarded as a pictorial form of a truth table. The map method is also known as the *Karnaugh map* or *K-map*.
- A K-map is a diagram made up of squares, with each square representing one minterm of the function that is to be minimized.
- Since any Boolean function can be expressed as a sum of minterms, it follows that a Boolean function is recognized graphically in the map from the area enclosed by those squares whose minterms are included in the function.
- In fact, the map presents a visual diagram of all possible ways a function may be expressed in standard form.

- The simplified expressions produced by the map are always in one of the two standard forms: sum of products or product of sums.
- It will be assumed that the simplest algebraic expression is an algebraic expression with a minimum number of terms and with the smallest possible number of literals in each term.
- This expression produces a circuit diagram with a minimum number of gates and the minimum number of inputs to each gate.
- The simplest expression is not unique: It is sometimes possible to find two or more expressions that satisfy the minimization criteria.
- In that case, either solution is satisfactory.

Two-Variable K-Map

•

		y	
		0	1
x	0	m_0	m_1
	1	m_2	m_3

		y	
		0	1
x	0	$x'y'$	$x'y$
	1	xy'	xy

Two-Variable K-Map

- Ex: xy

		y	
		0	1
x	0	m_0	m_1
	1	m_2	m_3

		y	
		0	1
x	0	$x'y'$	$x'y$
	1	xy'	xy

		y	
		0	1
x	0	0	0
	1	0	1

Two-Variable K-Map

- Ex: $x + y$

These squares are found from the minterms of the function:

$$m_1 + m_2 + m_3 = x'y + xy' + xy = x'y + x(y + y') = x'y + x = (x + y)(x + x') = x + y$$

		y	
		0	1
x	0	m_0	m_1
	1	m_2	m_3

		y	
		0	1
x	0	$x'y'$	$x'y$
	1	xy'	xy

		y	
		0	1
x	0	0	1
	1	1	1

Three Variable K-Map

		y			
		00	01	11	10
x	0	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
	1	$xy'z'$	$xy'z$	xyz	xyz'

		y			
		00	01	11	10
x	0	m_0	m_1	m_3	m_2
	1	m_4	m_5	m_7	m_6

- There are eight minterms for three binary variables; therefore, the map consists of eight squares.
- Note that the minterms are arranged, not in a binary sequence, but in a sequence similar to the Gray code.
- The characteristic of this sequence is that **only one bit changes in value from one adjacent column to the next.**

- To understand the usefulness of the map in 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.

- From the postulates of Boolean algebra, the sum of two minterms in adjacent squares can be simplified to a single product term consisting of only two literals.
- Any two minterms in adjacent squares (vertically or horizontally, but not diagonally, adjacent) that are ORed together will cause a removal of the dissimilar variable.
- **Ex:** $m_5 + m_7 = x\textcolor{red}{y}'z + x\textcolor{red}{y}z = xz(y' + y) = xz$
Here, the two squares differ by the variable y , which can be removed when the sum of the two minterms is formed.

Three Variable K-Map

- Simplify the Boolean function $F(x, y, z) = \sum(2, 3, 4, 5)$

		yz			
		00	01	11	10
x	0	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
	1	$xy'z'$	$xy'z$	xyz	xyz'

		yz			
		00	01	11	10
x	0	m_0	m_1	m_3	m_2
	1	m_4	m_5	m_7	m_6

		yz			
		00	01	11	10
x	0	0	0	1	1
	1	1	1	0	0

		yz			
		00	01	11	10
x	0	0	0	1	1
	1	1	1	0	0

- $m_3 + m_2 = x'y\bar{z} + x'yz' = x'y$
 - $m_4 + m_5 = xy'\bar{z} + xy'z = xy'$
- Hence, $F(x, y, z) = \sum(2, 3, 4, 5) = x'y + xy'$

Three Variable K-Map

- Simplify the Boolean function $F(x, y, z) = \sum(3, 4, 6, 7)$

x \ yz				
	00	01	11	10
0	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
1	$xy'z'$	$xy'z$	xyz	xyz'

x \ yz				
	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

x \ yz				
	00	01	11	10
0	0	0	1	0
1	1	0	1	1

x \ yz				
	00	01	11	10
0	0	0	1	0
1	1	0	1	1

- $m_3 + m_7 = x'yz + xyz = yz$
 - $m_4 + m_6 = xy'z' + xyz' = xz'$
- Hence, $F(x, y, z) = \sum(3, 4, 6, 7) = yz + xz'$

- The number of adjacent squares that may be combined must always represent a number that is a power of two, such as 1, 2, 4, and 8.
- As more adjacent squares are combined, we obtain a product term with fewer literals.
 - One square represents one minterm, giving a term with three literals.
 - Two adjacent squares represent a term with two literals.
 - Four adjacent squares represent a term with one literal.
 - Eight adjacent squares encompass the entire map and produce a function that is always equal to 1.

Three Variable K-Map

- Simplify the Boolean function $F(x, y, z) = \sum(0, 2, 4, 5, 6)$

		yz			
		00	01	11	10
x	0	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
	1	$xy'z'$	$xy'z$	xyz	xyz'

		yz			
		00	01	11	10
x	0	m_0	m_1	m_3	m_2
	1	m_4	m_5	m_7	m_6

		yz			
		00	01	11	10
x	0	1	0	0	1
	1	1	1	0	1

		yz			
		00	01	11	10
x	0	1	0	0	1
	1	1	1	0	1

- $m_0 + m_2 + m_4 + m_6 = x'y'z' + x'yz' + xy'z' + xyz' = z'$
- $m_4 + m_5 = xy'z' + xy'z = xy'$ Hence, $F(x, y, z) = \sum(0, 2, 4, 5, 6) = z' + xy'$

- If a function is not expressed in sum-of-minterms form, it is possible to use the map to obtain the minterms of the function and then simplify the function to an expression with a minimum number of terms.
- It is necessary, however, to make sure that the algebraic expression is in sum-of-products form.
- Each product term can be plotted in the map in one, two, or more squares. The minterms of the function are then read directly from the map.

Three Variable K-Map



For the Boolean function $F = A'C + A'B + AB'C + BC$

(a) Express this function as a sum of minterms. $F = A'C + A'B + AB'C + BC = \sum(1, 2, 3, 5, 7)$

(b) Find the minimal sum-of-products expression.

A \ BC		BC			
		00	01	11	10
A	0	$A'B'C'$	$A'B'C$	$A'BC$	$A'BC'$
	1	$AB'C'$	$AB'C$	ABC	ABC'

A \ BC		BC			
		00	01	11	10
A	0	m_0	m_1	m_3	m_2
	1	m_4	m_5	m_7	m_6

A \ BC		BC			
		00	01	11	10
A	0	0	1	1	1
	1	0	1	1	0

A \ BC		BC			
		00	01	11	10
A	0	0	1	1	1
	1	0	1	1	0

- $m_1 + m_3 + m_5 + m_7 = A'B'C + A'BC + AB'C + ABC = C$
- $m_3 + m_2 = A'BC + A'BC' = A'B$ Hence, $F(A, B, C) = \sum(1, 2, 3, 5, 7) = C + A'B$

Four Variable K-Map

yz wx		00	01	11	10
		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		$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$

yz wx		00	01	11	10
		00	01	11	10
00		m_0	m_1	m_3	m_2
01		m_4	m_5	m_7	m_6
11		m_{12}	m_{13}	m_{15}	m_{14}
10		m_8	m_9	m_{11}	m_{10}

Four Variable K-Map

- Simplify the Boolean function $F(w, x, y, z) = \sum(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

yz wx		00	01	11	10				
		$w'x'y'z'$	$w'x'y'z$	$w'x'yz$	$w'x'yz'$	m_0	m_1	m_3	m_2
00									
01						m_4	m_5	m_7	m_6
11						m_{12}	m_{13}	m_{15}	m_{14}
10						m_8	m_9	m_{11}	m_{10}

yz wx		00	01	11	10				
		1	1	0	1	1	1	0	1
00									
01									
11									
10									

yz wx		00	01	11	10				
		1	1	0	1	1	1	0	1
00									
01									
11									
10									

Four Variable K-Map

- Simplify the Boolean function $F(w, x, y, z) = \sum(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

yz wx \		00	01	11	10				
		00	01	11	10	m_0	m_1	m_3	m_2
00		$w'x'y'z'$	$w'x'y'z$	$w'x'yz$	$w'x'yz'$	m_0	m_1	m_3	m_2
01		$w'xy'z'$	$w'xy'z$	$w'xyz$	$w'xyz'$	m_4	m_5	m_7	m_6
11		$wxy'z'$	$wxy'z$	$wxyz$	$wxyz'$	m_{12}	m_{13}	m_{15}	m_{14}
10		$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$	m_8	m_9	m_{11}	m_{10}

yz wx \		00	01	11	10
		00	01	11	10
00		1	1	0	1
01		1	1	0	1
11		1	1	0	1
10		1	1	0	0

yz wx \		00	01	11	10
		00	01	11	10
00		1	1	0	1
01		1	1	0	1
11		1	1	0	1
10		1	1	0	0

Four Variable K-Map

- Simplify the Boolean function $F(w, x, y, z) = \sum(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

yz wx		00	01	11	10				
		$w'x'y'z'$	$w'x'y'z$	$w'x'yz$	$w'x'yz'$	m_0	m_1	m_3	m_2
00									
01						m_4	m_5	m_7	m_6
11						m_{12}	m_{13}	m_{15}	m_{14}
10						m_8	m_9	m_{11}	m_{10}

yz wx		00	01	11	10				
						m_0	m_1	m_3	m_2
00		1	1	0	1				
01		1	1	0	1				
11		1	1	0	1				
10		1	1	0	0				

yz wx		00	01	11	10				
						m_0	m_1	m_3	m_2
00		1	1	0	1				
01		1	1	0	1				
11		1	1	0	1				
10		1	1	0	0				

Four Variable K-Map

- Simplify the Boolean function $F(w, x, y, z) = \sum(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

wx \ yz		yz							
		00	01	11	10				
wx	00	$w'x'y'z'$	$w'x'y'z$	$w'x'yz$	$w'x'yz'$	m_0	m_1	m_3	m_2
	01	$w'xy'z'$	$w'xy'z$	$w'xyz$	$w'xyz'$	m_4	m_5	m_7	m_6
	11	$wxy'z'$	$wxy'z$	$wxyz$	$wxyz'$	m_{12}	m_{13}	m_{15}	m_{14}
	10	$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$	m_8	m_9	m_{11}	m_{10}

wx \ yz		yz			
		00	01	11	10
wx	00	1	1	0	1
	01	1	1	0	1
	11	1	1	0	1
	10	1	1	0	0

wx \ yz		yz			
		00	01	11	10
wx	00	1	1	0	1
	01	1	1	0	1
	11	1	1	0	1
	10	1	1	0	0

- i) $m_0 + m_1 + m_4 + m_5 + m_8 + m_9 + m_{12} + m_{13}$
 $= w'x'y'z' + w'x'y'z + w'xy'z' + w'xyz' + wx'y'z' + wx'y'z + wx'y'z' + wx'y'z$
 $= y'$
- ii) $m_0 + m_2 + m_4 + m_6 = w'x'y'z' + w'x'yz' + w'xy'z' + w'xyz'$
 $= w'z'$
- iii) $m_4 + m_6 + m_{12} + m_{14} = w'xy'z' + w'xyz' + wxy'z' + wxyz'$
 $= xz'$
- Hence, $F(w, x, y, z) = \sum(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$
 $= y' + w'z' + xz'$

Four Variable K-Map

- Simplify the Boolean function $F = A'B'C' + B'CD' + A'BCD' + AB'C'$

AB \ CD		CD							
		00	01	11	10				
AB	00	$A'B'C'D'$	$A'B'C'D$	$A'B'CD$	$A'B'CD'$	m_0	m_1	m_3	m_2
	01	$A'BC'D'$	$A'BC'D$	$A'BCD$	$A'BCD'$	m_4	m_5	m_7	m_6
	11	$ABC'D'$	$ABC'D$	$ABCD$	$ABCD'$	m_{12}	m_{13}	m_{15}	m_{14}
	10	$AB'C'D'$	$AB'C'D$	$AB'CD$	$AB'CD'$	m_8	m_9	m_{11}	m_{10}

AB \ CD		CD			
		00	01	11	10
AB	00	1	1	0	1
	01	0	0	0	1
	11	0	0	0	0
	10	1	1	0	1

AB \ CD		CD			
		00	01	11	10
AB	00	1	1	0	1
	01	0	0	0	1
	11	0	0	0	0
	10	1	1	0	1

Four Variable K-Map

- Simplify the Boolean function $F = A'B'C' + B'CD' + A'BCD' + AB'C'$

AB \ CD		CD							
		00	01	11	10				
AB	00	$A'B'C'D'$	$A'B'C'D$	$A'B'CD$	$A'B'CD'$	m_0	m_1	m_3	m_2
	01	$A'BC'D'$	$A'BC'D$	$A'BCD$	$A'BCD'$	m_4	m_5	m_7	m_6
	11	$ABC'D'$	$ABC'D$	$ABCD$	$ABCD'$	m_{12}	m_{13}	m_{15}	m_{14}
	10	$AB'C'D'$	$AB'C'D$	$AB'CD$	$AB'CD'$	m_8	m_9	m_{11}	m_{10}

AB \ CD		CD			
		00	01	11	10
AB	00	1	1	0	1
	01	0	0	0	1
	11	0	0	0	0
	10	1	1	0	1

AB \ CD		CD			
		00	01	11	10
AB	00	1	1	0	1
	01	0	0	0	1
	11	0	0	0	0
	10	1	1	0	1

Four Variable K-Map

- Simplify the Boolean function $F = A'B'C' + B'CD' + A'BCD' + AB'C'$

AB \ CD		CD							
		00	01	11	10				
AB	00	$A'B'C'D'$	$A'B'C'D$	$A'B'CD$	$A'B'CD'$	m_0	m_1	m_3	m_2
	01	$A'BC'D'$	$A'BC'D$	$A'BCD$	$A'BCD'$	m_4	m_5	m_7	m_6
	11	$ABC'D'$	$ABC'D$	$ABCD$	$ABCD'$	m_{12}	m_{13}	m_{15}	m_{14}
	10	$AB'C'D'$	$AB'C'D$	$AB'CD$	$AB'CD'$	m_8	m_9	m_{11}	m_{10}

AB \ CD		CD			
		00	01	11	10
AB	00	1	1	0	1
	01	0	0	0	1
	11	0	0	0	0
	10	1	1	0	1

AB \ CD		CD			
		00	01	11	10
AB	00	1	1	0	1
	01	0	0	0	1
	11	0	0	0	0
	10	1	1	0	1

Four Variable K-Map

- Simplify the Boolean function $F = A'B'C' + B'CD' + A'BCD' + AB'C'$

AB \ CD		CD							
		00	01	11	10				
AB	00	$A'B'C'D'$	$A'B'C'D$	$A'B'CD$	$A'B'CD'$	m_0	m_1	m_3	m_2
	01	$A'BC'D'$	$A'BC'D$	$A'BCD$	$A'BCD'$	m_4	m_5	m_7	m_6
	11	$ABC'D'$	$ABC'D$	$ABCD$	$ABCD'$	m_{12}	m_{13}	m_{15}	m_{14}
	10	$AB'C'D'$	$AB'C'D$	$AB'CD$	$AB'CD'$	m_8	m_9	m_{11}	m_{10}

AB \ CD		CD			
		00	01	11	10
AB	00	1	1	0	1
	01	0	0	0	1
	11	0	0	0	0
	10	1	1	0	1

AB \ CD		CD			
		00	01	11	10
AB	00	1	1	0	1
	01	0	0	0	1
	11	0	0	0	0
	10	1	1	0	1

- i) $A'B'C'D' + A'B'C'D + AB'C'D' + AB'C'D = B'C'$
- ii) $A'B'C'D' + A'B'CD' + AB'C'D' + AB'CD' = B'D'$
- iii) $A'B'CD' + A'BCD' = A'CD'$

Hence, $F = A'B'C' + B'CD' + A'BCD' + AB'C' = B'C' + B'D' + A'CD'$

-
-

PRODUCT-OF-SUMS SIMPLIFICATION

- Simplify the following Boolean function into (a) sum-of-products form and (b) product-of-sums form: $F(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10)$

AB \ CD		CD							
		00	01	11	10				
AB	00	$A'B'C'D'$	$A'B'C'D$	$A'B'CD$	$A'B'CD'$	m_0	m_1	m_3	m_2
	01	$A'BC'D'$	$A'BC'D$	$A'BCD$	$A'BCD'$	m_4	m_5	m_7	m_6
	11	$ABC'D'$	$ABC'D$	$ABCD$	$ABCD'$	m_{12}	m_{13}	m_{15}	m_{14}
	10	$AB'C'D'$	$AB'C'D$	$AB'CD$	$AB'CD'$	m_8	m_9	m_{11}	m_{10}

AB \ CD		CD			
		00	01	11	10
AB	00	1	1	0	1
	01	0	1	0	0
	11	0	0	0	0
	10	1	1	0	1

AB \ CD		CD			
		00	01	11	10
AB	00	1	1	0	1
	01	0	1	0	0
	11	0	0	0	0
	10	1	1	0	1

PRODUCT-OF-SUMS SIMPLIFICATION

- Simplify the following Boolean function into (a) sum-of-products form and (b) product-of-sums form: $F(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10)$

AB \ CD		CD							
		00	01	11	10				
AB	00	$A'B'C'D'$	$A'B'C'D$	$A'B'CD$	$A'B'CD'$	m_0	m_1	m_3	m_2
	01	$A'BC'D'$	$A'BC'D$	$A'BCD$	$A'BCD'$	m_4	m_5	m_7	m_6
	11	$ABC'D'$	$ABC'D$	$ABCD$	$ABCD'$	m_{12}	m_{13}	m_{15}	m_{14}
	10	$AB'C'D'$	$AB'C'D$	$AB'CD$	$AB'CD'$	m_8	m_9	m_{11}	m_{10}

AB \ CD		CD			
		00	01	11	10
AB	00	1	1	0	1
	01	0	1	0	0
	11	0	0	0	0
	10	1	1	0	1

AB \ CD		CD			
		00	01	11	10
AB	00	1	1	0	1
	01	0	1	0	0
	11	0	0	0	0
	10	1	1	0	1

PRODUCT-OF-SUMS SIMPLIFICATION

- Simplify the following Boolean function into (a) sum-of-products form and (b) product-of-sums form: $F(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10)$

AB \ CD		CD							
		00	01	11	10				
AB	00	$A'B'C'D'$	$A'B'C'D$	$A'B'CD$	$A'B'CD'$	m_0	m_1	m_3	m_2
	01	$A'BC'D'$	$A'BC'D$	$A'BCD$	$A'BCD'$	m_4	m_5	m_7	m_6
	11	$ABC'D'$	$ABC'D$	$ABCD$	$ABCD'$	m_{12}	m_{13}	m_{15}	m_{14}
	10	$AB'C'D'$	$AB'C'D$	$AB'CD$	$AB'CD'$	m_8	m_9	m_{11}	m_{10}

AB \ CD		CD			
		00	01	11	10
AB	00	1	1	0	1
	01	0	1	0	0
	11	0	0	0	0
	10	1	1	0	1

AB \ CD		CD			
		00	01	11	10
AB	00	1	1	0	1
	01	0	1	0	0
	11	0	0	0	0
	10	1	1	0	1

- The squares marked with 0's represent the minterms not included in F and therefore denote the complement of F .
- i) $ABC'D' + ABC'D + ABCD + ABCD' = AB$
- ii) $A'B'CD + A'BCD + ABCD + AB'CD = CD$
- iii) $A'BC'D' + ABC'D' + A'BCD' + ABCD' = BD'$
- Hence, (a) $F = AB + CD + BD'$
- Applying DeMorgan's theorem (by taking the dual and complementing each literal, we obtain the simplified function in product of sums form:

$$(b) F = (A' + B') (C' + D') (B' + D)$$

- Combining the squares with 1's gives the simplified function in sum-of-products form:

$$(a) F = B'D' + B'C' + A'C'D$$





























