REVIEW OF NUMBER SYSTEMS

- In general, N bits can represent 2^N different values.
- For M values, $\lceil \log_2 M \rceil$ bits are needed.

1 bit → represents up to 2 values (0 or 1)
2 bits → rep. up to 4 values (00, 01, 10 or 11)
3 bits → rep. up to 8 values (000, 001, 010. …, 110, 111)
4 bits → rep. up to 16 values (0000, 0001, 0010, …, 1111)

32 values → requires 5 bits
64 values → requires 6 bits
1024 values → requires 10 bits
40 values → requires 6 bits
100 values → requires 7 bits

DECIMAL (BASE 10) NUMBER SYSTEM

- Weighting factors (or weights) are in powers-of-10:

  … $10^3\ 10^2\ 10^1\ 10^0 . 10^{-1}\ 10^{-2}\ 10^{-3}\ 10^{-4}$ …

- To evaluate the decimal number 593.68, the digit in each position is multiplied by the corresponding weight:

  $5\times10^2 + 9\times10^1 + 3\times10^0 + 6\times10^{-1} + 8\times10^{-2} = (593.68)_{10}$

OTHER NUMBER SYSTEMS & BASE-R TO DECIMAL CONVERSION

- Binary (base 2): weights in powers-of-2.
  - Binary digits (bits): ***0,1***.
- Octal (base 8): weights in powers-of-8.
  - Octal digits: ***0,1,2,3,4,5,6,7***.
- Hexadecimal (base 16): weights in powers-of-16.
  - Hexadecimal digits: ***0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F***.
- Base *R*: weights in powers-of-R.

OTHER NUMBER SYSTEMS & BASE-R TO DECIMAL CONVERSION

- $(1101.101)_2 = 1\times2^3 + 1\times2^2 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-3}$
  $= 8 + 4 + 1 + 0.5 + 0.125 = (13.625)_{10}$
- $(572.6)_8 = 5\times8^2 + 7\times8^1 + 2\times8^0 + 6\times8^{-1}$
  $= 320 + 56 + 2 + 0.75 = (378.75)_{10}$
- $(2A.8)_{16} = 2\times16^1 + 10\times16^0 + 8\times16^{-1}$
  $= 32 + 10 + 0.5 = (42.5)_{10}$
- $(341.24)_5 = 3\times5^2 + 4\times5^1 + 1\times5^0 + 2\times5^{-1} + 4\times5^{-2}$
  $= 75 + 20 + 1 + 0.4 + 0.16 = (96.56)_{10}$

SUM-OF-WEIGHTS METHOD

- Determine the set of binary weights whose sum is equal to the decimal number.

$(9)_{10}$ = 8 + 1 = $2^3 + 2^0$ = $(1001)_2$

$(18)_{10}$ = 16 + 2 = $2^4 + 2^1$ = $(10010)_2$

$(58)_{10}$ = 32 + 16 + 8 + 2 = $2^5 + 2^4 + 2^3 + 2^1$ = $(111010)_2$

$(0.625)_{10}$ = 0.5 + 0.125 = $2^{-1} + 2^{-3}$ = $(0.101)_2$

REPEATED DIVISION-BY-2 METHOD

- To convert a whole number to binary, use successive division by 2 until the quotient is 0. The remainders form the answer, with the first remainder as the *least significant bit (LSB)* and the last as the *most significant bit (MSB)*.

$(43)_{10}$ = $(101011)_2$

2	43	
2	21 rem 1	← LSB
2	10 rem 1	
2	5 rem 0	
2	2 rem 1	
2	1 rem 0	
	0 rem 1	← MSB

REPEATED MULTIPLICATION-BY-2 METHOD

- To convert decimal fractions to binary, repeated multiplication by 2 is used, until the fractional product is 0 (or until the desired number of decimal places). The carried digits, or *carries*, produce the answer, with the first carry as the MSB, and the last as the LSB.

$(0.3125)_{10}$ = $(.0101)_2$

	Carry	
0.3125×2=0.625	0	←MSB
0.625×2=1.25	1	
0.25×2=0.50	0	
0.5×2=1.00	1	←LSB

BINARY-OCTAL/HEXADECIMAL CONVERSION

- Binary → Octal: Partition in groups of 3
  $(10\ 111\ 011\ 001\ .\ 101\ 110)_2 = (2731.56)_8$
- Octal → Binary: reverse
  $(2731.56)_8 = (10\ 111\ 011\ 001\ .\ 101\ 110)_2$
- Binary → Hexadecimal: Partition in groups of 4
  $(101\ 1101\ 1001\ .\ 1011\ 1000)_2 = (5D9.B8)_{16}$
- Hexadecimal → Binary: reverse
  $(5D9.B8)_{16} = (101\ 1101\ 1001\ .\ 1011\ 1000)_2$

BINARY CODED DECIMAL (BCD)

Decimal digit	0	1	2	3	4
BCD	0000	0001	0010	0011	0100
Decimal digit	5	6	7	8	9
BCD	0101	0110	0111	1000	1001

- Examples:
  - $(234)_{10}$ = $(0010\ 0011\ 0100)_{BCD}$
  - $(7093)_{10}$ = $(0111\ 0000\ 1001\ 0011)_{BCD}$
  - $(1000\ 0110)_{BCD}$ = $(86)_{10}$
  - $(1001\ 0100\ 0111\ 0010)_{BCD}$ = $(9472)_{10}$

Notes: BCD is not equivalent to binary.
Example: $(234)_{10} = (11101010)_2$

EXERCISE PROBLEMS

Convert each binary number to decimal:
(a) 110011.11 **(b)** 101010.01 **(c)** 1000001.111

Convert each decimal number to binary
(a) 263.26 **(b)** 5436.762 **(c)** 234543.0975

Convert each hexadecimal number to binary:
(a) FA_{16} **(b)** $1A3_{16}$ **(c)** $ABCD_{16}$

Convert each binary number to hexadecimal:
(a) 011101010100 **(b)** 100101111000 **(c)** 0001011010000011

Convert the following binary numbers to octal:
(a) 110101111 **(b)** 1001100010 **(c)** 10111111001

Convert the following octal numbers to binary:
(a) 46_8 **(b)** 723_8 **(c)** 5624_8

REPRESENTING NEGATIVE NUMBERS

- Are the negative numbers just numbers with a minus sign in the front? This is probably true…but there are issues to represent negative numbers in computing systems
- Common schemas:
  - Sign-magnitude
  - Complementary representations:
    - 1's complement
    - ***2's complement - most common & important***

SIGN MAGNITUDE

- Left most bit used to represent sign
  - 0 = positive value
  - 1 = negative value
  - behaves like a "flag"
- It is important to decide how many bits we will use to represent the number
- Example: Representing +5 and -5 on 8 bits:
  - +5: **00000101**
  - -5: **10000101**
- ***So the very first step we have to decide on the number of bits to represent number***

DIFFICULTIES WITH SIGN MAGNITUDE

- Two representations of zero
  - Using 8-bit sign-magnitude…
    - `0: 00000000`
    - `0: 10000000`
- Arithmetic is awkward!
  - 8-bit sign-magnitude:
    - 00000001 + 00000010 = 00000011
    - 00000010 + 10000001 = 00000001 (it requires a different algorithm, can't just add and carry, meaning more complexity in hardware in order to implement an ALU)

1'S BINARY COMPLEMENT

- Decide on the number of bits (word length) to represent numbers
- Then represent the negative numbers by the largest number minus the absolute value of the negative number.
- Example:
  - 8-digit 1's complement of –101
    - 11111111 – 00000101 = 111111010 (= (2^8 –1) –5 in base 10)
    - Notice: very easy: flip or "invert" the 1's and 0's to compute 1's complement of a number
    - To get back the **abs value**, invert again
- Most negative:
  - representation 10000000 …11111111 | 0… 01111111
  - original number -01111111 …-00000000 | 0… 01111111

1'S BINARY COMPLEMENT

Finding the 1's Complement

The 1's **complement** of a binary number is found by changing all 1s to 0s and all 0s to 1s, as illustrated below:

1 0 1 1 0 0 1 0 Binary number
↓↓↓↓↓↓↓↓
0 1 0 0 1 1 0 1 1's complement

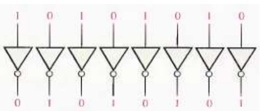
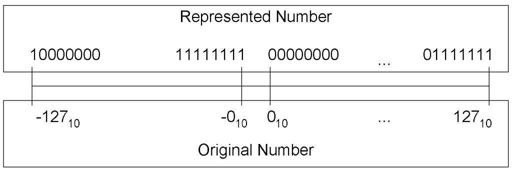


FIGURE 2–2
Example of inverters used to obtain the 1's complement of a binary number.

DIFFICULTIES WITH 1'S COMPLEMENT

- Two representations of zero
  - Using 6-digit 1's complement…
    - `0: 000000`
    - `0: 111111`
- Arithmetic is still a little awkward!



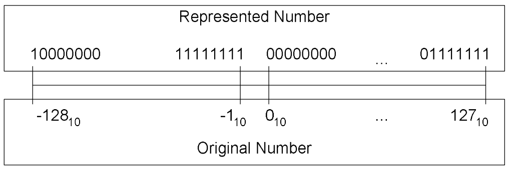
1's binary complement for 8 digit numbers

2'S COMPLEMENT

- Decide on the number of digits (word length) to represent numbers
- Then represent the negative numbers by the [largest number + 1] minus the absolute value of the negative number.
- Example:
  - 8-digit 2's complement of -5
    - 100000000 – 00000101 = 1111111011(= 2^8 -5 in base 10)
    - To get back the **abs value**, subtract again from 2^8

2'S COMPLEMENT

- The 2's complement of a number can be found in two ways
  - Subtract the value from the modulus [largest number +1]
  - Find 1's complement (by inverting the value) and adding 1 to the result (2'complement = 1's complement +1)



2's complement for 8 digit numbers

2'S COMPLEMENT

Finding the 2's Complement

The 2's complement of a binary number is found by adding 1 to the LSB of the 1's complement.

2's complement = (1's complement) + 1

Find the 2's complement of 10110010.

Solution

10110010	Binary number
01001101	1's complement
+ 1	Add 1
01001110	2's complement

Related Problem Determine the 2's complement of 11001011.

1'S COMPLEMENT VS. 2'S COMPLEMENT

- Both methods are used in computer design
- 1's complement
  - offers a simpler method to change the sign of a number
  - Requires an extra end-around carry step
  - Algorithm must test for and convert -0 to 0 at the and of each operation.
- 2's complement
  - Simplifies the addition operation
  - Additional add operation required every time a sign change is required (by inverting and adding 1)

EXERCISE PROBLEMS

Example. *Find the decimal equivalent of the following binary numbers assuming the binary numbers have been represented in sign-magnitude form.*

(a) 0101100 *(b) 101000* *(c) 1111* *(d) 011011*

(*a*) Sign bit is 0, which indicates the number is positive.
Magnitude $101100 = (44)_{10}$
Therefore $(0101100)_2 = (+44)_{10}$.

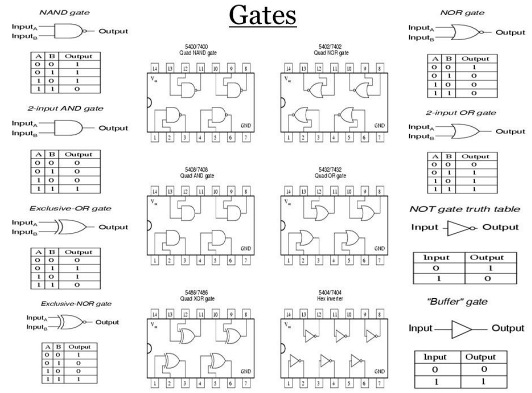
(*b*) Sign bit is 1, which indicates the number is negative.
Magnitude $01000 = (8)_{10}$
Therefore $(101000)_2 = (-8)_{10}$.

(*c*) Sign bit is 1, which indicates the number is negative.
Magnitude $111 = (7)_{10}$
Therefore $(1111)_2 = (-7)_{10}$.

(*d*) Sign bit is 0, which indicates the number is positive.
Magnitude $11011 = (27)_{10}$
Therefore $(011011)_2 = (+27)_{10}$.

1. Determine the 1's complement of each binary number:
   (a) 00011010 (b) 11110111 (c) 10001101
2. Determine the 2's complement of each binary number:
   (a) 00010110 (b) 11111100 (c) 10010001

Gates



BOOLEAN FUNCTIONS

- **Boolean function** is an expression formed with binary variables, the two binary operators, OR and AND, and the unary operator, NOT, parenthesis and the equal sign.
- Its result is also a binary value.
- We usually use **.** for AND, **+** for OR, and **'** or **¬** for NOT. Sometimes, we may omit the **.** if there is no ambiguity.

LAWS OF BOOLEAN ALGEBRA

Law/Theorem	Law of Addition	Law of Multiplication
Identity Law	$x + 0 = x$	$x \cdot 1 = x$
Complement Law	$x + x' = 1$	$x \cdot x' = 0$
Idempotent Law	$x + x = x$	$x \cdot x = x$
Dominant Law	$x + 1 = 1$	$x \cdot 0 = 0$
Involution Law	$(x')' = x$	
Commutative Law	$x + y = y + x$	$x \cdot y = y \cdot x$
Associative Law	$x+(y+z) = (x+y)+z$	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$
Distributive Law	$x \cdot (y+z) = x \cdot y+x \cdot z$	$x+y \cdot z = (x+y) \cdot (x+z)$
Demorgan's Law	$(x+y)' = x' \cdot y'$	$(x \cdot y)' = x' + y'$
Absorption Law	$x + (x \cdot y) = x$	$x \cdot (x + y) = x$

BOOLEAN FUNCTIONS

- Examples:
  F1= x.y.z'
  F2= x + y'.z

  F3=(x'.y'.z)+(x'.y.z)+(x.y')
  F4=x.y'+x'.z

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

From the truth table, F3=F4.
Can you also prove by algebraic manipulation that F3=F4?

COMPLEMENT OF FUNCTIONS

- Given a function, F, the **complement** of this function, F', is obtained by interchanging 1 with 0 in the function's output values.

Example: F1 = x.y.z'

Complement:
F1' = (x.y.z')'
= x' + y' + (z')' DeMorgan
= x' + y' + z Involution

x	y	z	F1	F1'
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

STANDARD FORMS

- Certain types of Boolean expressions lead to gating networks which are desirable from implementation viewpoint.
- Two Standard Forms:
  Sum-of-Products and ***Product-of-Sums***
- **Literals**: a variable on its own or in its complemented form. Examples: x, x' , y, y'
- **Product Term**: a single literal or a logical product (AND) of several literals.
  Examples: x, x.y.z', A'.B, A.B, e.g'.w.v

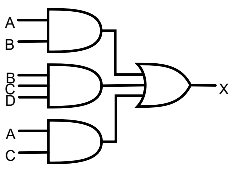
STANDARD FORMS

- **Sum Term**: a single literal or a logical sum (OR) of several literals.
  Examples: x, x+y+z', A'+B, A+B, c+d+h'+j
- **Sum-of-Products (SOP) Expression**: a product term or a logical sum (OR) of several product terms.
  Examples: x, x+y.z', x.y'+x'.y.z, A.B+A'.B',
  A + B'.C + A.C' + C.D
- **Product-of-Sums (POS) Expression**: a sum term or a logical product (AND) of several sum terms.
  Examples: x, x.(y+z'), (x+y').(x'+y+z),
  (A+B).(A'+B'), (A+B+C).D'.(B'+D+E')

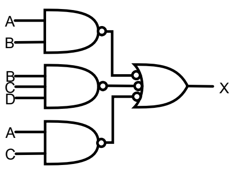
IMPLEMENTATION OF AN SOP

$X=AB+BCD+AC$

- AND/OR implementation



- NAND/NAND implementation



GENERAL EXPRESSION → SOP

- Any logic expression can be changed into SOP form by applying Boolean algebra techniques.

ex:

$$A(B+CD) = AB + ACD$$

$$AB + B(CD+EF) = AB + BCD + BEF$$

$$(A+B)(B+C+D) = AB + AC + AD + BB + BC + BD$$

$$\overline{(\overline{A}+\overline{B})+C} = \overline{(\overline{A}+\overline{B})}\,\overline{C} = (A+B)\overline{C} = A\overline{C} + B\overline{C}$$

CONVERTING SOP EXPRESSIONS TO TRUTH TABLE FORMAT (EXAMPLE)

- Develop a truth table for the standard SOP expression

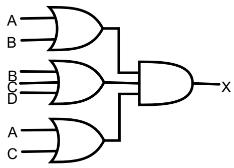
$$\overline{A}\overline{B}C + A\overline{B}\overline{C} + ABC$$

Inputs			Output	Product Term
A	B	C	X	
0	0	0	0	
0	0	1	1	$\overline{A}\overline{B}C$
0	1	0	0	
0	1	1	0	
1	0	0	1	$A\overline{B}\overline{C}$
1	0	1	0	
1	1	0	0	
1	1	1	1	ABC

IMPLEMENTATION OF A POS

$X=(A+B)(B+C+D)(A+C)$

- OR/AND implementation



CONVERTING POS EXPRESSIONS TO TRUTH TABLE FORMAT (EXAMPLE)

- Develop a truth table for the standard SOP expression

$(A+B+C)(A+\overline{B}+C)(A+\overline{B}+\overline{C})$
$(\overline{A}+B+\overline{C})(\overline{A}+\overline{B}+C)$

Inputs			Output	Product Term
A	B	C	X	
0	0	0	0	$(A+B+C)$
0	0	1	1	
0	1	0	0	$(A+\overline{B}+C)$
0	1	1	0	$(A+\overline{B}+\overline{C})$
1	0	0	1	
1	0	1	0	$(\overline{A}+B+\overline{C})$
1	1	0	0	$(\overline{A}+\overline{B}+C)$
1	1	1	1	

DETERMINING STANDARD EXPRESSION FROM A TRUTH TABLE (EXAMPLE)

I/P			O/P
A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

- There are four 1s in the output and the corresponding binary value are 011, 100, 110, and 111.

SOP

$011 \rightarrow \overline{A}BC$
$100 \rightarrow A\overline{B}\overline{C}$
$110 \rightarrow AB\overline{C}$
$111 \rightarrow ABC$

$$\boxed{X = \overline{A}BC + A\overline{B}\overline{C} + AB\overline{C} + ABC}$$

- There are <u>four 0s</u> in the output and the corresponding binary value are 000, 001, 010, and 101.

POS

$000 \rightarrow A+B+C$
$001 \rightarrow A+B+\overline{C}$
$010 \rightarrow A+\overline{B}+C$
$101 \rightarrow \overline{A}+B+\overline{C}$

$$\boxed{X = (A+B+C)(A+B+\overline{C})(A+\overline{B}+C)(\overline{A}+B+\overline{C})}$$

BOOLEAN FUNCTION SIMPLIFICATION

- $\mathbf{AB + \overline{A}C + BC = AB + \overline{A}C}$ **(Consensus Theorem)**

<u>**Proof Steps**</u>	<u>**Justification**</u>
$AB + \overline{A}C + BC$	
$= AB + \overline{A}C + 1 \cdot BC$	Identity element
$= AB + \overline{A}C + (A + \overline{A}) \cdot BC$	Complement
$= AB + \overline{A}C + ABC + \overline{A}BC$	Distributive
$= AB + ABC + \overline{A}C + \overline{A}CB$	Commutative
$= AB \cdot 1 + ABC + \overline{A}C \cdot 1 + \overline{A}CB$	Identity element
$= AB\,(1+C) + \overline{A}C\,(1 + B)$	Distributive
$= AB\,.\,1 + \overline{A}C\,.\,1$	$1+X = 1$
$= AB + \overline{A}C$	Identity element

BOOLEAN ALGEBRA SIMPLIFICATION

- (A + B)(A + C) = A + BC
- This rule can be proved as follows:
- (A + B)(A + C) = AA + AC + AB + BC(Distributive law)
  - = A + AC + AB + BC (AA = A)
  - = A(1 + C) + AB + BC (1 + C = 1)
  - = A. 1 + AB + BC
  - = A(1 + B) + BC (1 + B = 1)
  - = A. 1 + BC (A . 1 = A)
  - = A + BC

INTRODUCTION TO K-MAPS

- Systematic method to obtain *simplified* sum-of-products (SOPs) Boolean expressions.
- Objective: *Fewest* possible terms/literals.
- Diagrammatic technique based on a special form of *Venn diagram*.
- Advantage: Easy with visual aid.
- Disadvantage: Limited to 5 or 6 variables.

SIMPLIFICATION USING K-MAPS

- Larger groups correspond to product terms of fewer literals. In the case of a 4-variable K-map:

1 cell	=	4 literals, e.g.: w.x.y.z, w'.x.y'.z
2 cells	=	3 literals, e.g.: w.x.y, w.y'.z'
4 cells	=	2 literals, e.g.: w.x, x'.y
8 cells	=	1 literal, e.g.: w, y', z
16 cells	=	no literal, e.g.: 1

RULES OF K-MAPS

- Based on the Unifying Theorem:

  A + A' = 1

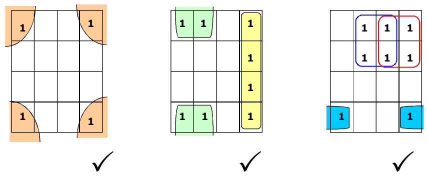
- In a K-map, each cell containing a '1' corresponds to a minterm of a given function *F*.
- Each group of adjacent cells containing '1' (group must have size *in powers of twos*: 1, 2, 4, 8, …) then corresponds to a simpler product term of *F*.
  - Grouping 2 adjacent squares eliminates 1 variable, grouping 4 squares eliminates 2 variables, grouping 8 squares eliminates 3 variables, and so on. In general, grouping 2^n squares eliminates *n* variables.

RULES OF K-MAPS

- Group as many squares as possible.
  - The larger the group is, the fewer the number of literals in the resulting product term.
- Select as few groups as possible to cover all the squares (minterms) of the function.
  - The fewer the groups, the fewer the number of product terms in the minimized function.

RULES OF K-MAPS

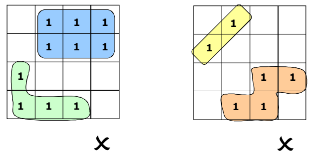
- Other possible valid groupings of a 4-variable K-map include:



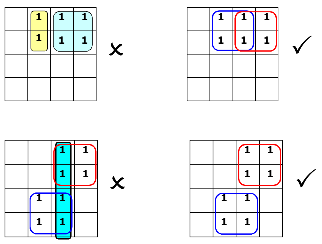
RULES OF K-MAPS

- Groups of minterms must be
  (1) rectangular, and
  (2) have size in powers of 2's.

Otherwise they are *invalid* groups. Some examples of *invalid groups*:



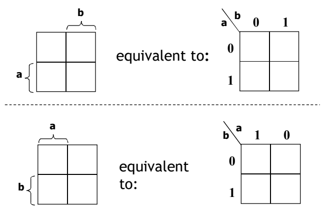
RULES OF K-MAPS



Large Group Size but Number of Groups should be minimum

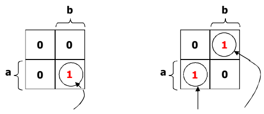
2-VARIABLE K-MAPS

- Equivalent labeling:



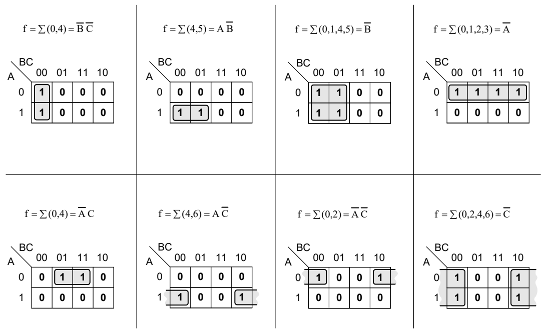
2-VARIABLE K-MAPS

- The K-map for a function is specified by putting
  - ❖ a ‘1’ in the square corresponding to a minterm
  - ❖ a ‘0’ otherwise
- For example:



C = a.b   S = a.b' + a'.b

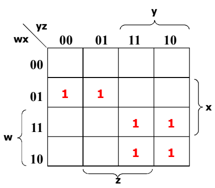
THREE-VARIABLE K-MAPS



SIMPLIFICATION USING K-MAPS

- Example:

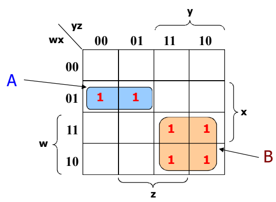
  F (w,x,y,z) = w’.x.y'.z' + w'.x.y'.z + w.x'.y.z'
  + w.x'.y.z + w.x.y.z' + w.x.y.z
  = Σ m(4, 5, 10, 11, 14, 15)



(cells with ‘0’ are not shown for clarity)

SIMPLIFICATION USING K-MAPS

- Each group of adjacent minterms (group size in powers of twos) corresponds to a possible product term of the given function.

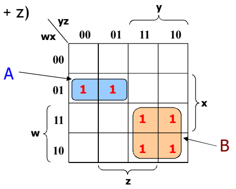


SIMPLIFICATION USING K-MAPS

- There are 2 groups of minterms: A and B, where:

A = w'.x.y'.z' + w'.x.y'.z
= w'.x.y'.(z' + z)
= w'.x.y'

B = w.x'.y.z' + w.x'.y.z + w.x.y.z' + w.x.y.z
= w.x'.y.(z' + z) + w.x.y.(z' + z)
= w.x'.y + w.x.y
= w.(x'+x).y
= w.y



SIMPLIFICATION USING K-MAPS

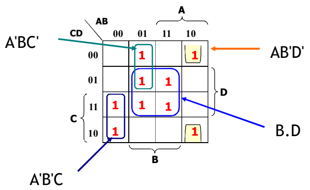
- Each product term of a group, w'.x.y' and w.y, represents the *sum of minterms* in that group.
- Boolean function is therefore the sum of product terms (SOP) which represent all groups of the minterms of the function.

F(w,x,y,z) = A + B = w'.x.y' + w.y

SIMPLEST SOP EXPRESSIONS

- Example:

f(A,B,C,D) = $\sum m(2,3,4,5,7,8,10,13,15)$

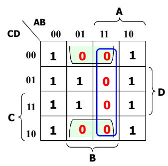


f(A,B,C,D) = B.D + A'.B'.C + A.B'.D' + A'.B.C'

GETTING POS EXPRESSIONS

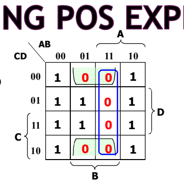
- Simplified POS expression can be obtained by grouping the maxterms (i.e. 0s) of given function.
- Example:

Given F=$\sum$m(0,1,2,3,5,7,8,9,10,11), we first draw the K-map, then group the maxterms together:



GETTING POS EXPRESSIONS

K-map of F



- To get POS of F, we have:

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

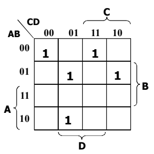
DON'T-CARE CONDITIONS

- Don't-care conditions can be used to help simplify Boolean expression further in K-maps.
- They could be chosen to be either '1' or '0', depending on which gives the simpler expression.
- We usually use the notation Σd to denote the set of don't-care minterms. For example, the function P can be written as:

  P = Σm(0, 3, 5, 6, 9) + Σd(10, 11, 12, 13, 14, 15)

DON'T-CARE CONDITIONS

- P = Σm(0, 3, 5, 6, 9) + Σd(10, 11, 12, 13, 14, 15)

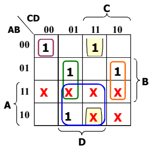


- For comparison:
  - WITHOUT don't-cares:

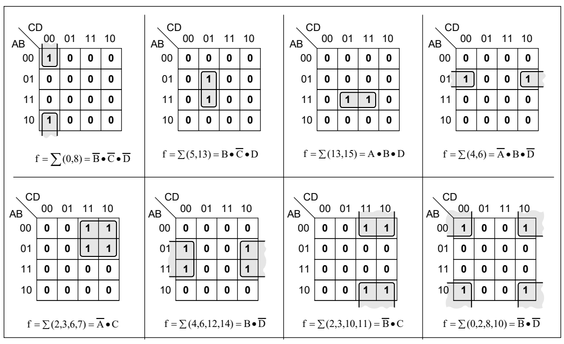
    P = A'.B'.C'.D' + A'.B'.C.D + A'.B.C'.D
    + A'.B.C.D' + A.B'.C'.D

  - WITH don't-cares:

    P = A'.B'.C'.D' + B'.C.D + B.C'.D
    + B.C.D' + A.D



FOUR-VARIABLE K-MAPS



FOUR-VARIABLE K-MAPS