PDS0101

Introduction to Digital Systems

Combinational Logic Analysis II

Lecture outcome

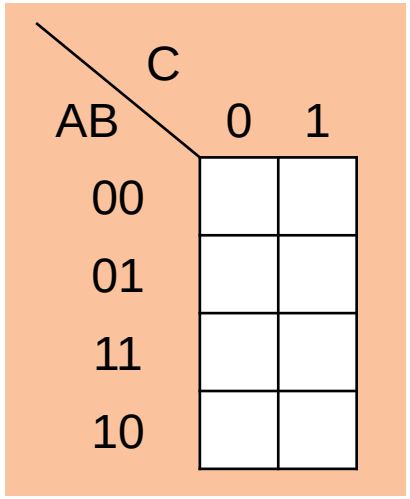
- By the end of today's lecture you should know
  - How to use a Karnaugh map to simplify boolean expressions
  - How to use a Karnaugh map to simplify truth table functions
  - How don't care conditions can be used in K-maps
  - The applications of Boolean algebra and the K-map method to a system application

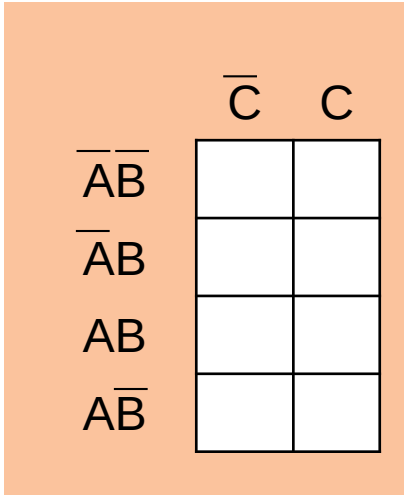
Karnaugh maps

- The Karnaugh map (K-map) is a tool for simplifying combinational logic with up to 3 or 4 variables.
- Karnaugh mapping is also used as an alternative to minimize the number of logic gates that are required in a digital circuit.
- This will replace Boolean reduction when the circuit is large.
- First we must write the Boolean equation in a SOP sum-of-product form, then place each term possibility on a map.
  - The map is made up of a table of every possible SOP using the number of variables that are being used.
    - If two variables are used then a 2×2 map is used
    - If three variables are used then a 4×2 map is used
    - If four variables are used then a 4×4 map is used
    - If five variables are used then a 8×4 map is used

Creating k-maps

- Determine the minimum number of cells required to contain all possibilities of the variables
- Cells are usually labeled using 0's and 1's to represent the variable and its complement and are entered in ***gray code***, to force adjacent cells to be different by only one variable
  - Ones are read as the true variable and zeros are read as the complemented variable
- Alternatively, cells can be labeled with the variable letters. This makes it simple to read, but it takes more time preparing the map





Two variable k-map

- The two-variable k-map is a graph with its sides labeled with input variables A and B and their complements

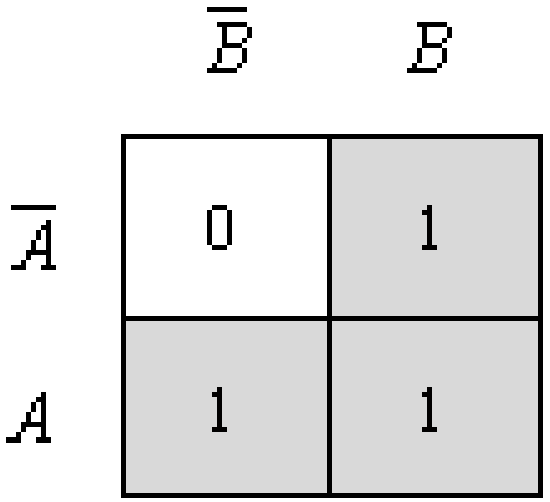
A	B	Product term
0	0	$\overline{A}\,\overline{B}$
0	1	$\overline{A}B$
1	0	$A\overline{B}$
1	1	AB

	$\overline{B}$	B
$\overline{A}$	$\overline{A}\,\overline{B}$	$\overline{A}B$
A	$A\overline{B}$	AB

- Example
  Map the following SOP expression into its k-map form

$$Y = \overline{A}B + A\overline{B} + AB$$

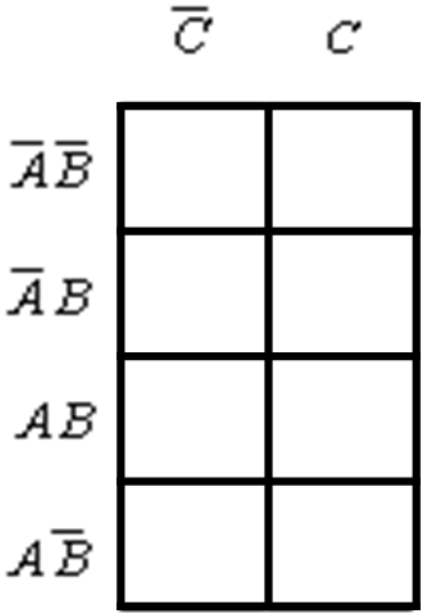
A	B	Y	
0	0	0	
0	1	1	$\overline{A}B$
1	0	1	$A\overline{B}$
1	1	1	AB



Three variable K-map

- The three variable k-map lists the possible combinations as a pair of variables against a singular variable in an 8-cell graph

A	B	C	Y
0	0	0	$\bar{A}\bar{B}\bar{C}$
0	0	1	$\bar{A}\bar{B}C$
0	1	0	$\bar{A}B\bar{C}$
0	1	1	$\bar{A}BC$
1	0	0	$A\bar{B}\bar{C}$
1	0	1	$A\bar{B}C$
1	1	0	$AB\bar{C}$
1	1	1	ABC



Three variable K-map

- Map the following SOP expression into its k-map form

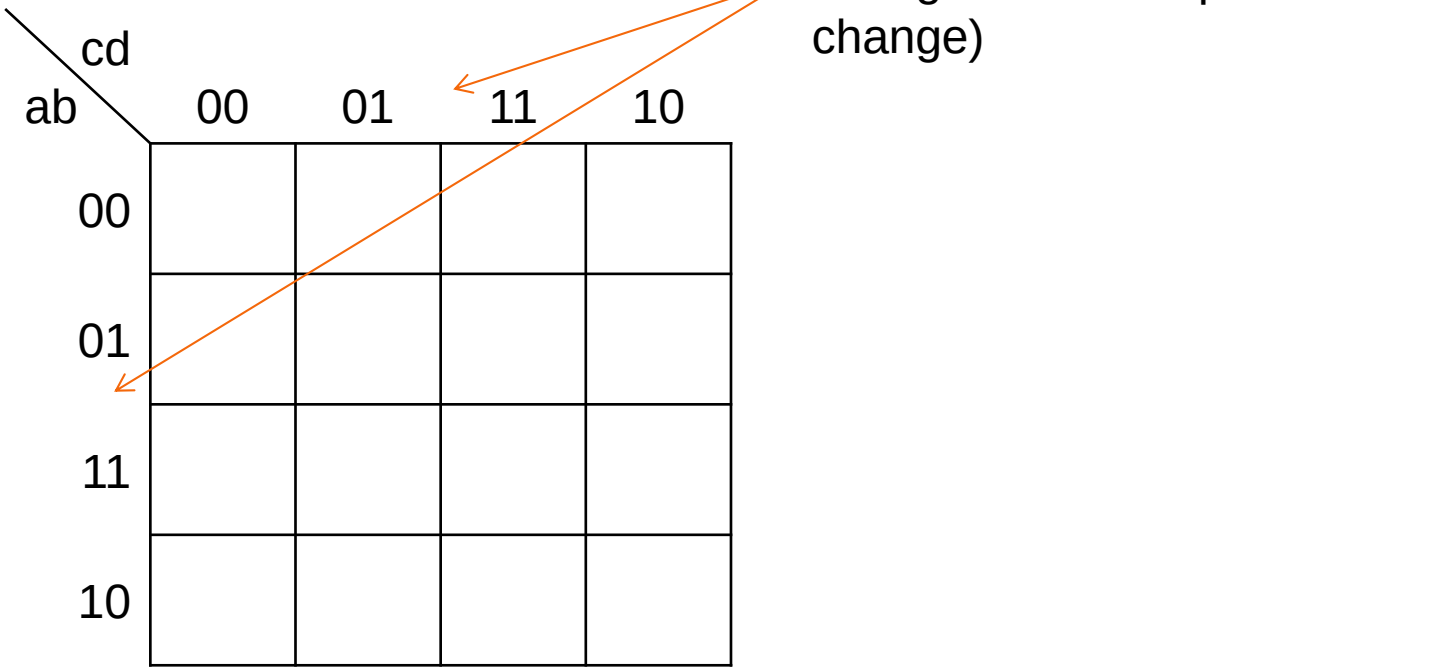
$$\overline{A}\,\overline{B}C + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}C + ABC$$

A	B	C	Y	
0	0	0	0	
0	0	1	1	$\overline{A}\,\overline{B}C$
0	1	0	1	$\overline{A}B\overline{C}$
0	1	1	1	$\overline{A}BC$
1	0	0	0	
1	0	1	1	$A\overline{B}C$
1	1	0	0	
1	1	1	1	ABC

	$\overline{C}$	C
$\overline{A}\,\overline{B}$	0	1
$\overline{A}B$	1	1
AB	0	1
$A\overline{B}$	0	1

Four variable K-map

- How do you define the four variable K-map?



- Example

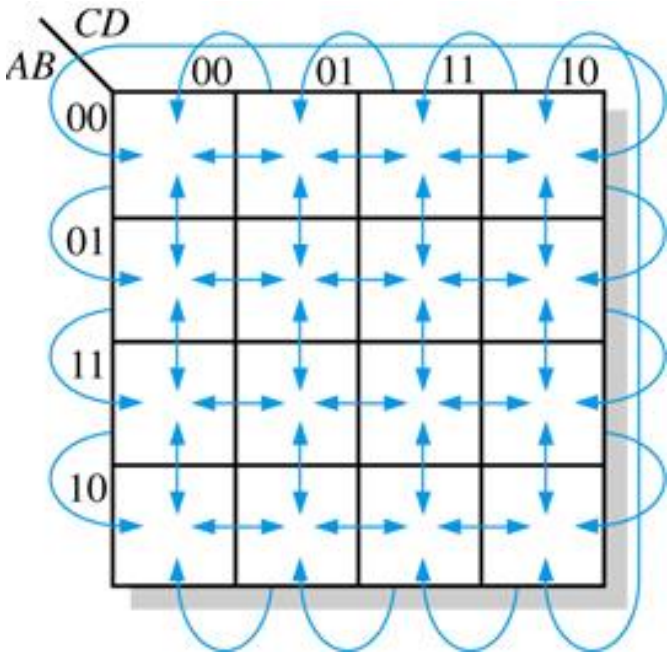
  Map the following SOP expression into its k-map form

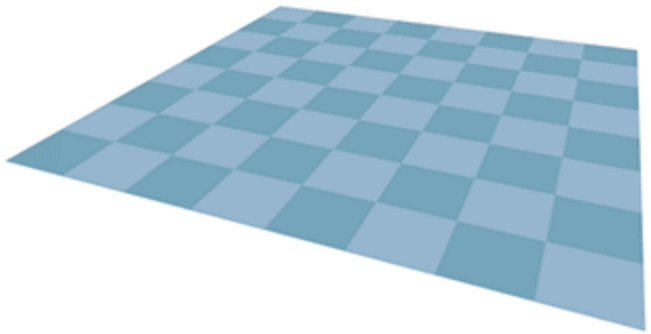
$$\overline{A}B\overline{C}\overline{D} + A\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}B\overline{C}D + AB\overline{C}D$$
$$+ A\overline{B}\overline{C}D + \overline{A}BCD + A\overline{B}CD + ABCD$$

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

K-map property

- The interesting about the k-map is that to go from one cell of the map to an adjacent cell will only require one variable (bit) to change (also known as gray coding).
- When moving on the map you can only go right, left, up or down and never on a diagonal.
- Also the map folds around its self so the going from a cell on the top right to one on the bottom right only changes one variable

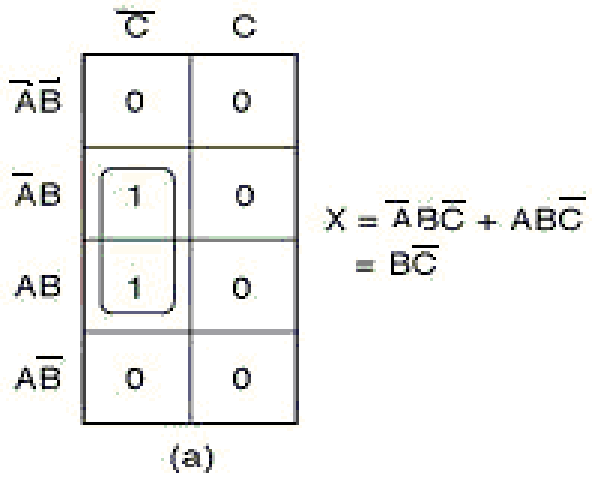


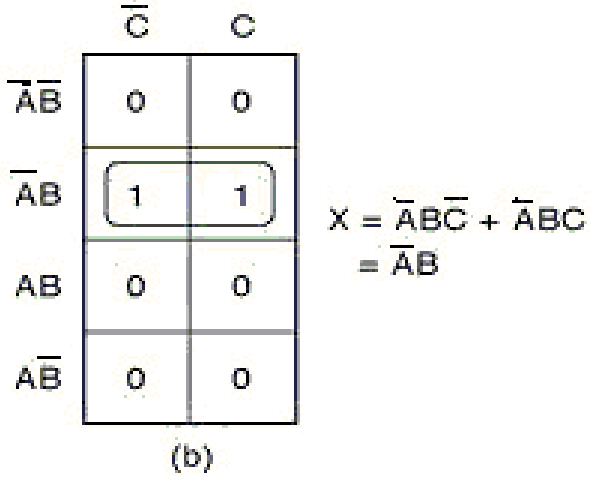


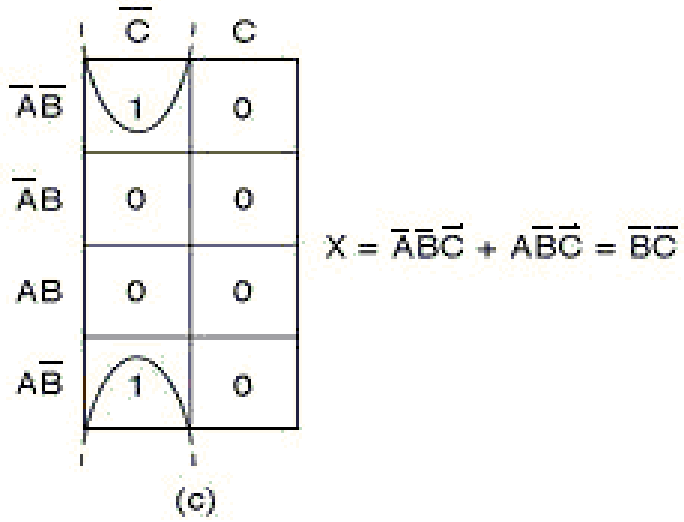
K-map looping

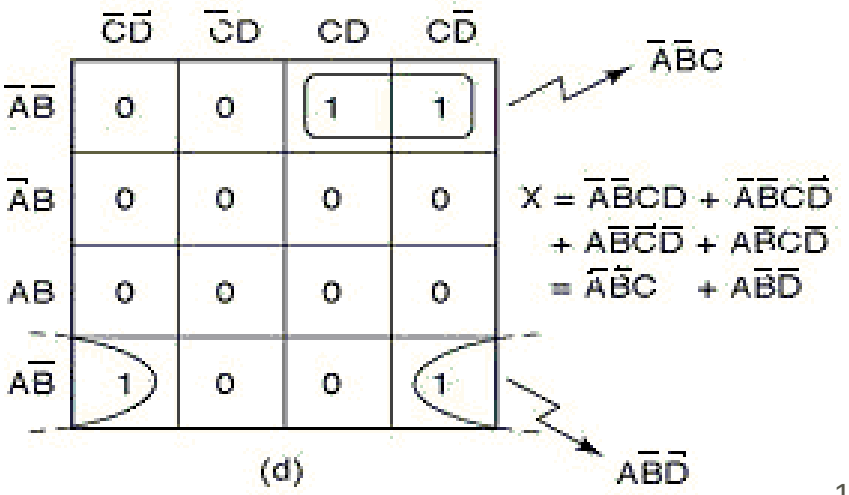
- The next stage with K-maps is to simplify the equation
- The expression for output Y can be simplified by properly combining those squares in the K map which contain 1s.
- The process for combining these 1s is called looping.
- Loops must be in a power of two, e.g.
  - Looping groups of two, groups of four, groups of eight etc
- Loops (sometimes called groups) should be as large as possible for maximum simplification. To be exact
  - loop pairs eliminate one variable
  - quad loops eliminate two variables
  - octet loops eliminate three variables
- Overlaps are allowed in loops provided every loop has at least one ‘1’ that is unique

Looping groups of two (pairs)

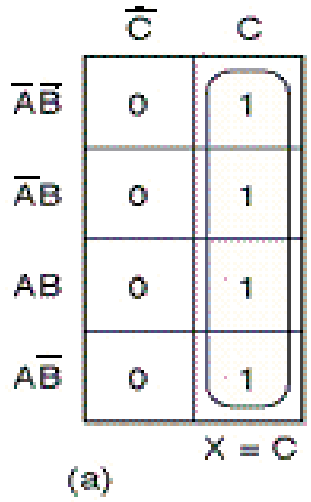


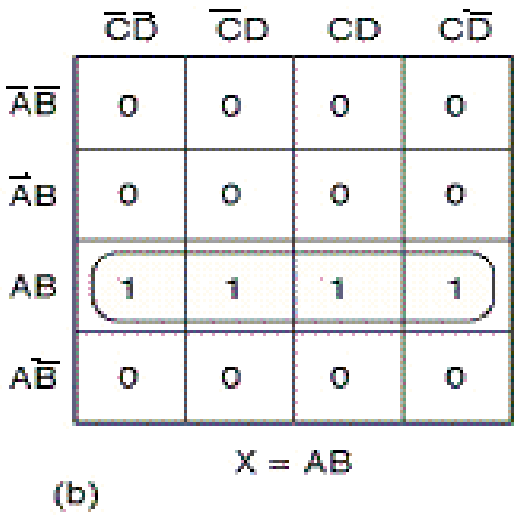


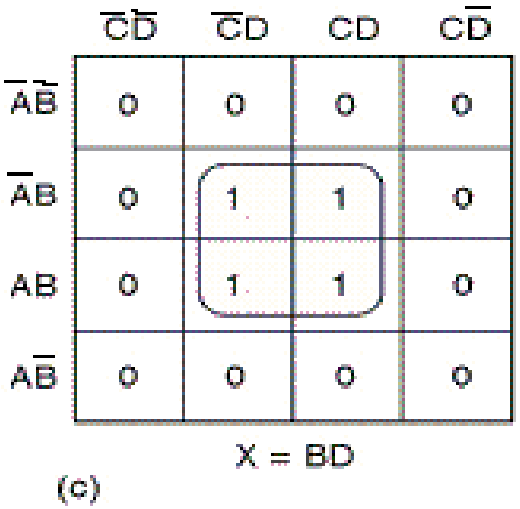


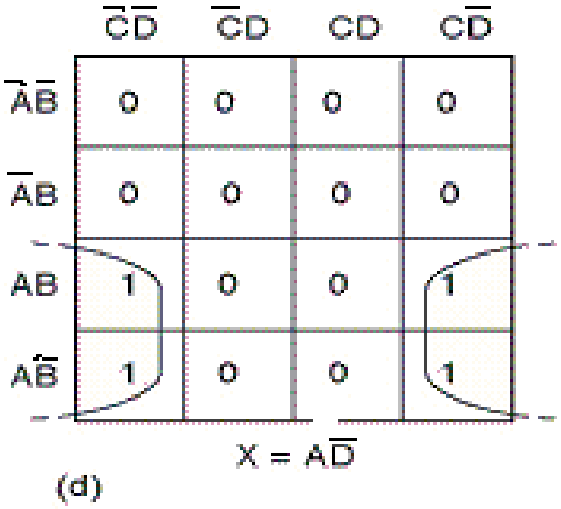


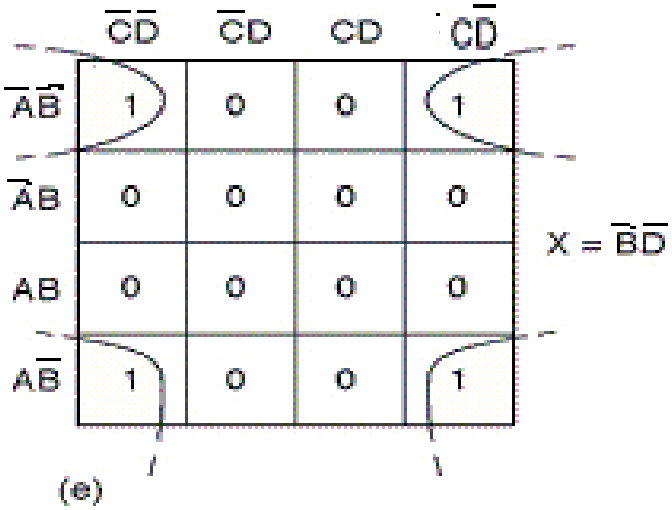
Looping groups of four (quads)



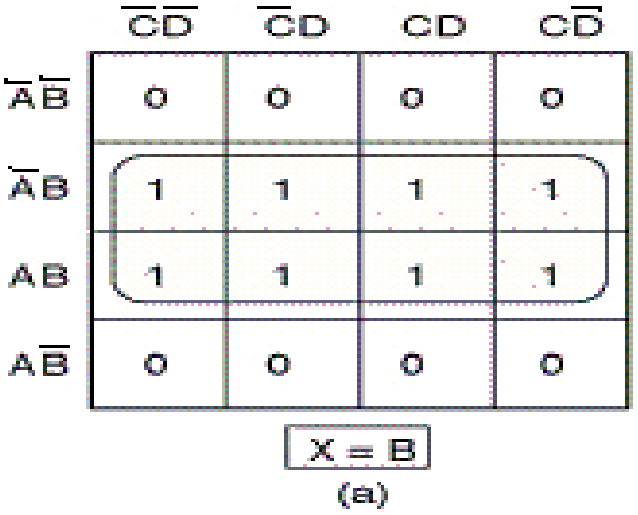


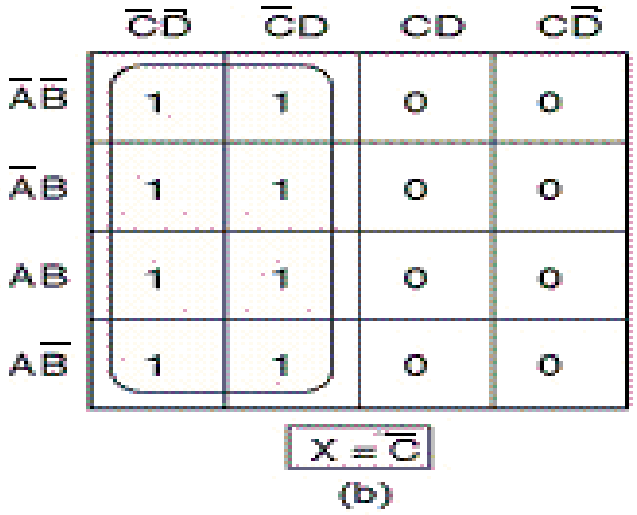


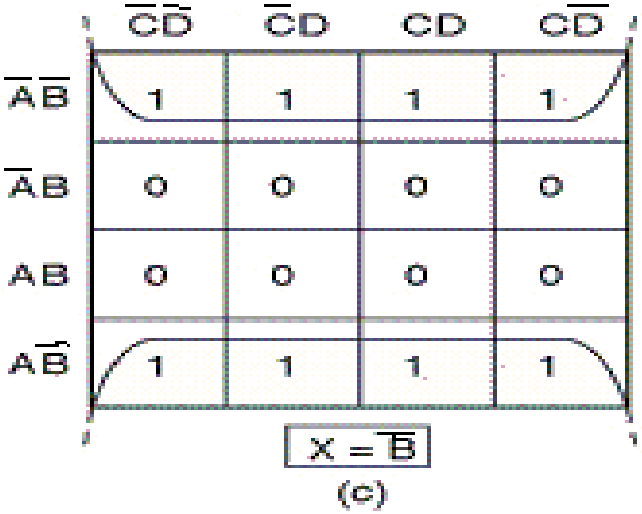


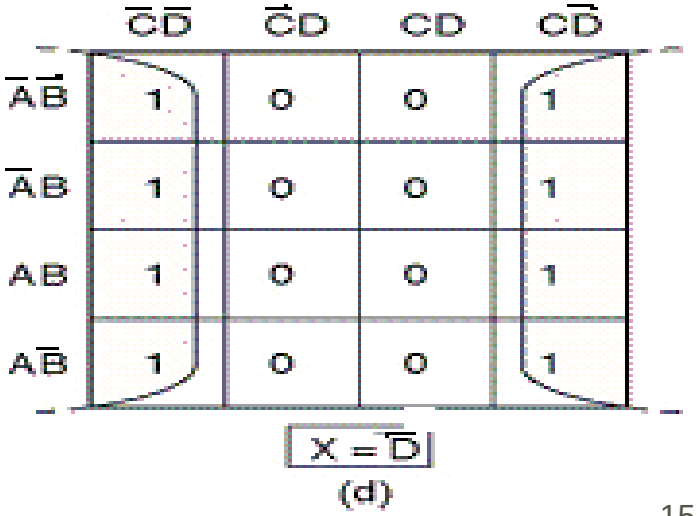


Looping groups of eight (octets)



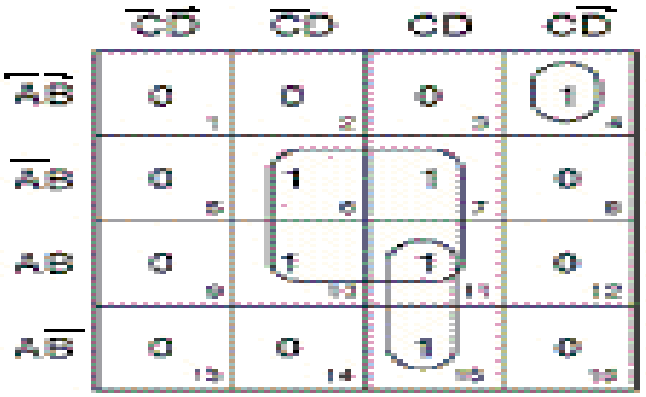






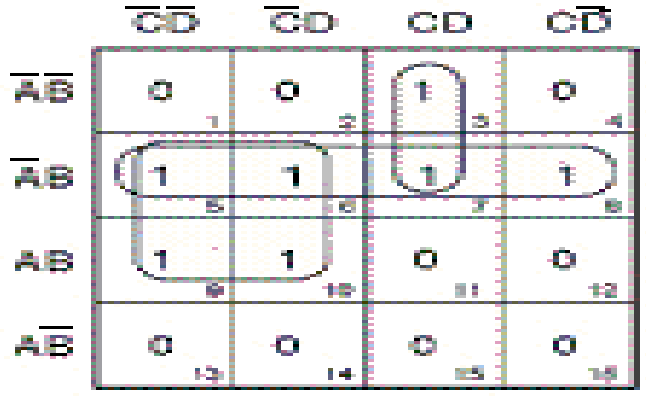
The k-map simplification process

- Steps to follow in using the K-map method for simplifying a SOP standard form boolean expression :
  - Construct the K-map and place 1s in those squares corresponding to the 1s in the truth table. Place 0s in the other squares
  - Examine the map for adjacent 1s and loop those 1s which are not adjacent to any other 1s. These are isolated 1s
  - Next, look for 1s which are adjacent to only one other. Loop any pair containing such 1s
  - Loop any quad that contains one or more 1s that have not already been looped, making sure to use the minimum number of loops
  - Loop any octet even if it contains some 1s that already been looped
    - Loop any pairs necessary to include any 1s that have not yet been looped, making sure to use the minimum of loops
  - Form the OR sum of all the terms generated by each loop



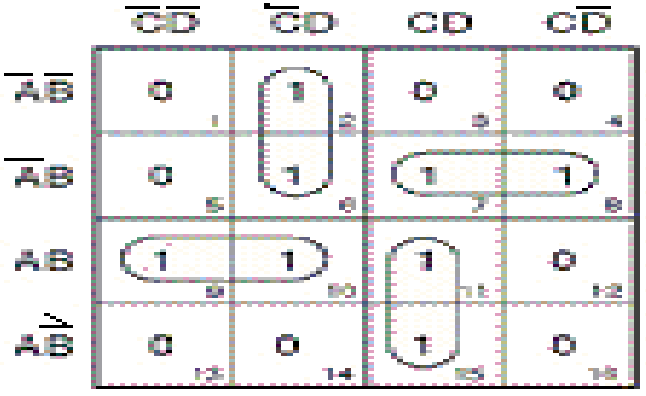
$$X = \underbrace{\overline{A}\overline{B}C\overline{D}}_{\text{loop 4}} + \underbrace{ACD}_{\text{loop 11, 15}} + \underbrace{BD}_{\text{loop 6, 7, 10, 11}}$$

(a)



$$X = \underbrace{\overline{A}B}_{\text{loop 5, 6, 7, 8}} + \underbrace{B\overline{C}}_{\text{loop 5, 6, 9, 10}} + \underbrace{\overline{A}CD}_{\text{loop 3,7}}$$

(b)



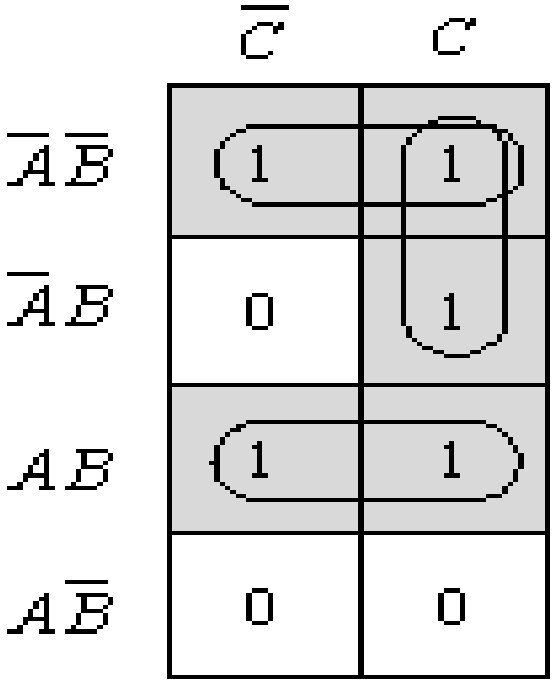
$$X = \underbrace{AB\overline{C}}_{9,\ 10} + \underbrace{\overline{A}\overline{C}D}_{2,\ 6} + \underbrace{\overline{A}BC}_{7,\ 8} + \underbrace{ACD}_{11,\ 15}$$

(c)

Example 1

- Use the K-map method to minimize the following SOP boolean expression

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



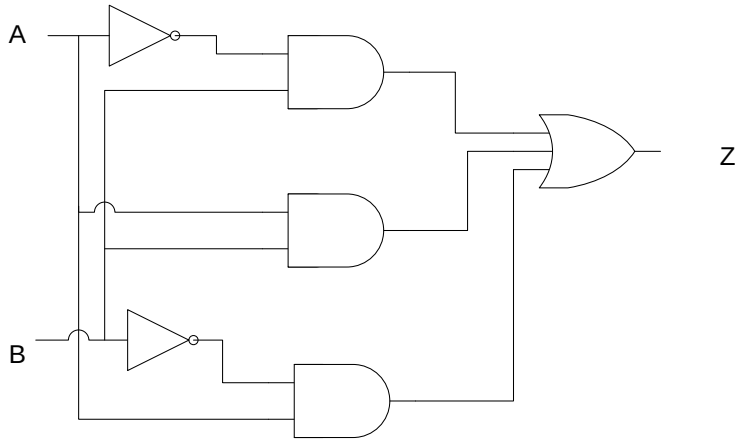
$$Y = AB + \overline{A}C + \overline{A}\,\overline{B}$$

Example 2

- Construct the true table for the Boolean expression below and draw the resulting logic circuit.

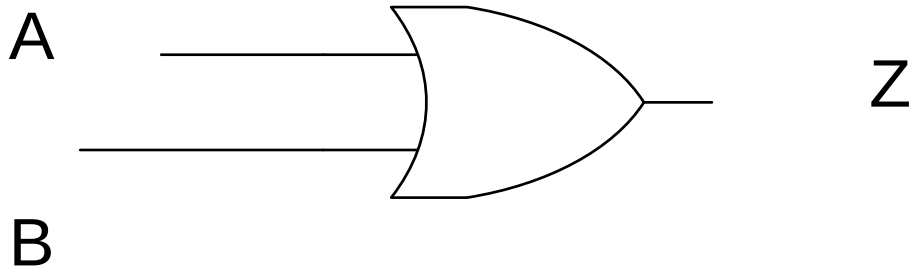
$$\bar{A}B + AB + A\bar{B}$$

A	B	A’	B’	AB	A’B	AB’	A’B+AB+AB’
0	0	1	1	0	0	0	0
0	1	1	0	0	1	0	1
1	0	0	1	0	0	1	1
1	1	0	0	1	0	0	1



- Using the K-map method, simplify the boolean equation from the previous slide and draw the simplified circuit

	$\overline{B}$	B
$\overline{A}$	0	1
A	1	1

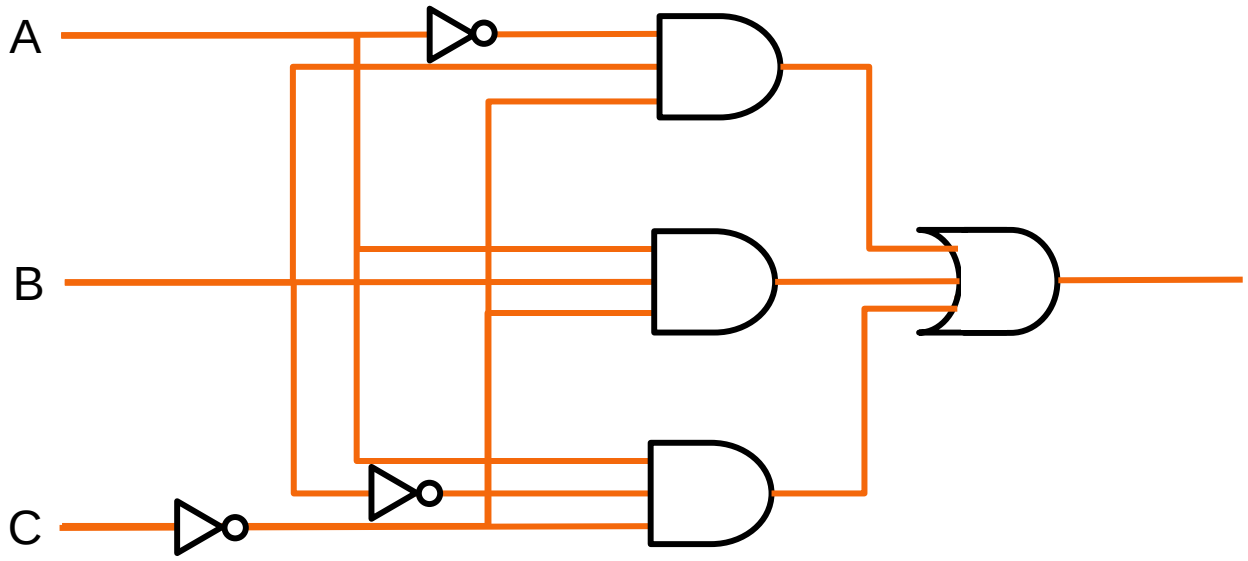


Exercise

- Draw the circuit based on SOP Boolean expression below

$$\bar{A}B\bar{C} + AB\bar{C} + A\bar{B}\,\bar{C}$$

- Use K-map to minimize the expression and redraw the simplified circuit



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

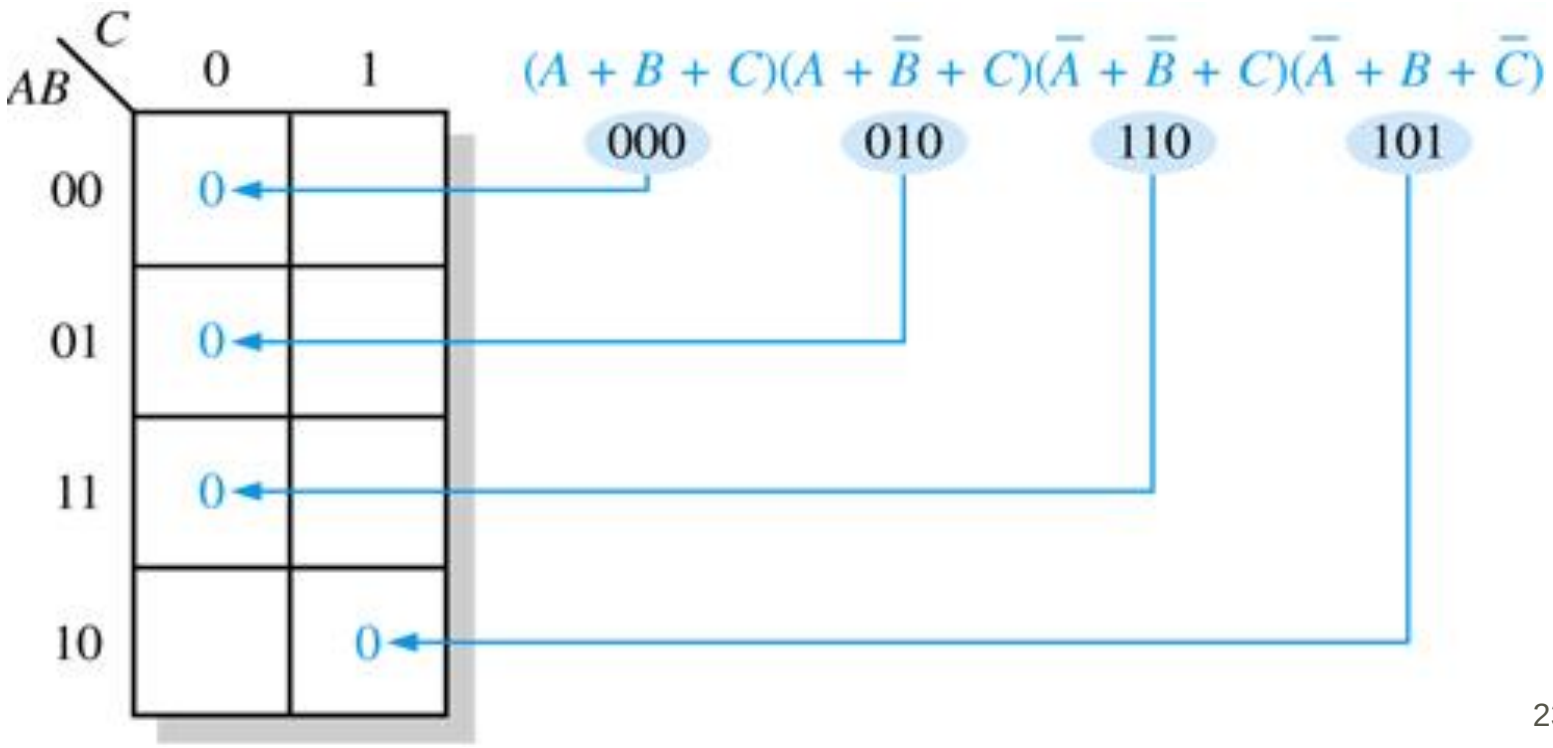
Exercise

- Use a K-map to simplify the following expression

$$Y = AB + \overline{B}C + AC$$

The k-map simplification process 2

- For a ***POS expression in standard form***, a 0 is placed on the Karnaugh map for each sum term in the expression.
- Each 0 is placed in a cell corresponding to the value of a sum term.
- The cells that do not have a 0 are the cells for which the expression is 1.
- E.g. mapping a standard POS expression



- The process for minimizing a POS expression is basically the same as for an SOP expression except that grouping is based on 0s to produce minimum sum terms instead of grouping 1s to produce minimum product terms.
- The steps for grouping the 0s are the same as those for grouping the 1s.

Example

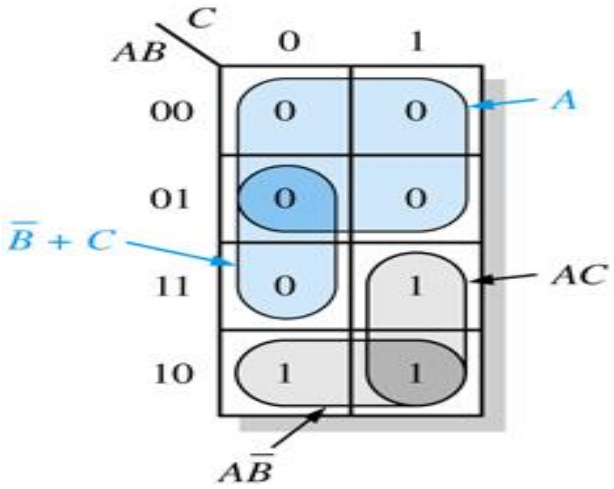
- Use a Karnaugh map to minimize the following standard POS expression

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

- The combination of binary values of the expression are

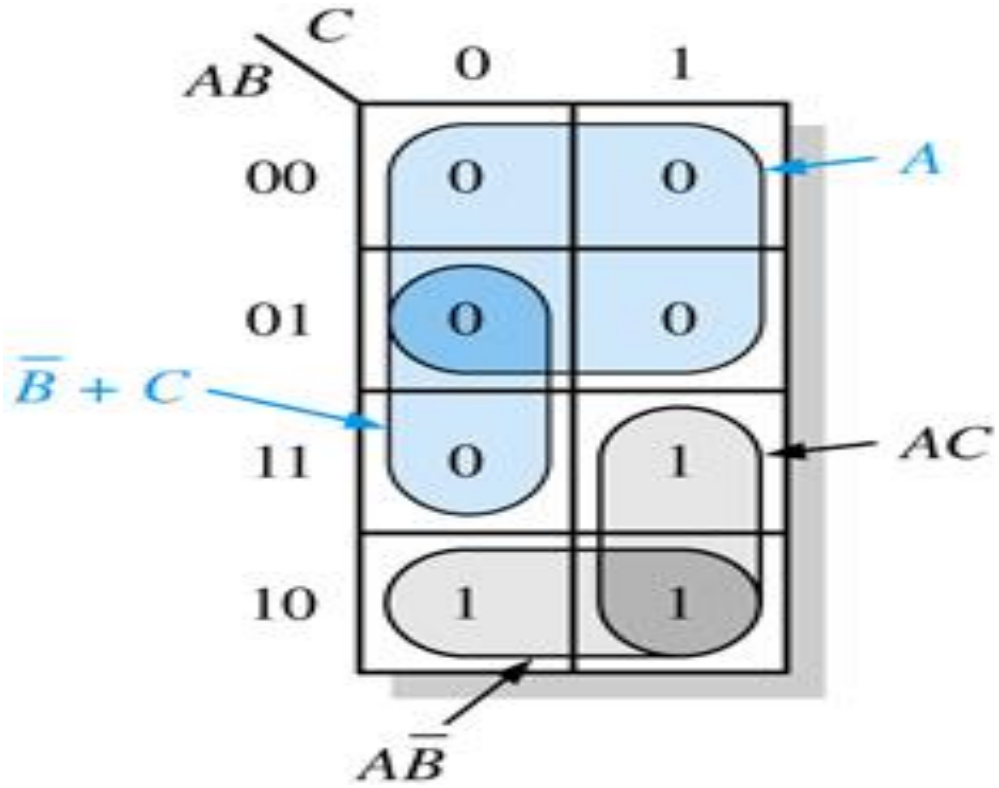
(0+0+0)(0+0+1)(0+1+0)(0+1+1)(1+1+0)

- When mapped onto the k-map graph and 0s are grouped



- The sum term for each blue group is shown in the figure and the resulting minimum POS expression is $A(\overline{B}+C)$

- Grouping the 1s as shown by gray areas yields an SOP expression that is equivalent to grouping the 0s



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

Example 2

- Write Y in POS Boolean expression for the given truth table.

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

- Draw the logic circuit using Basic logic gates (AND, OR, NOT)
- Simplify the Boolean expression using K-map
- Draw the simplified circuit

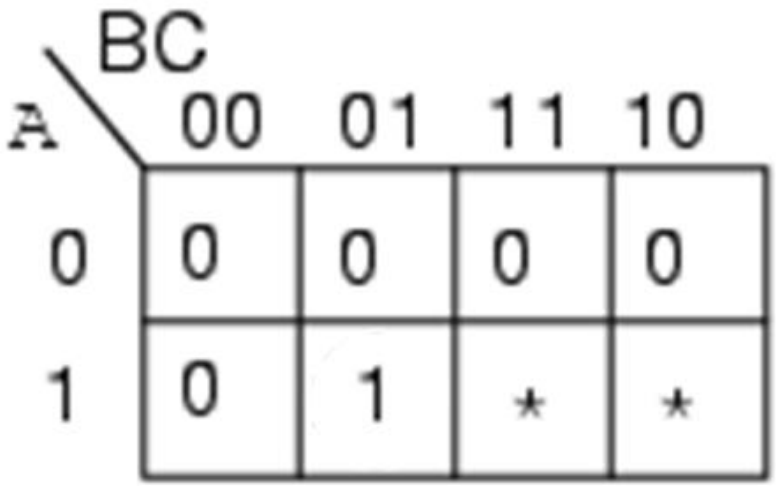
K-map *don't cares*

- Some logic circuits can be designed so that there are certain input conditions for which there are no specific output levels, usually because these input conditions will never occur
  - These will be the don't care conditions where we literally "don't care" what the output is no matter the input level
- Example
  BCD encodes decimal digits (0-9) as 4-bit binary numbers – but 4 bits equates to 16 possible numbers
  Binary values of 1010, 1011, 1100, 1101, 1110 and 1111 will never be used in BCD and thus these six values are *don't cares*

- In a k-map, don't cares are indicated with asterisks * and can be treated as either a 1 or 0 (for simplification purposes) or just ignored entirely.
- This is helpful if it allows us to form a larger group than would otherwise be possible without the don't cares
- There is no requirement to group all or any of the don't cares. Only use them in a group if it simplifies the logic

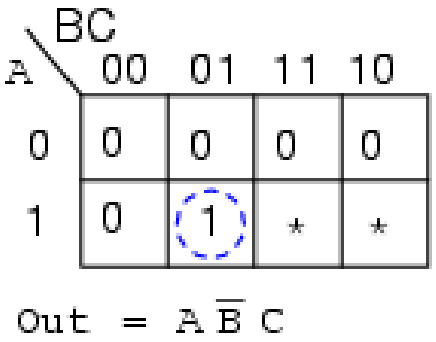
Don’t care example

- Assume a logic function where the desired output is 1 for input ABC = 101 over the range from 000 to 101.
- We *don’t care* what the output is for the other possible inputs (110, 111).
- When mapped to a three variable k-map, the resulting graph is

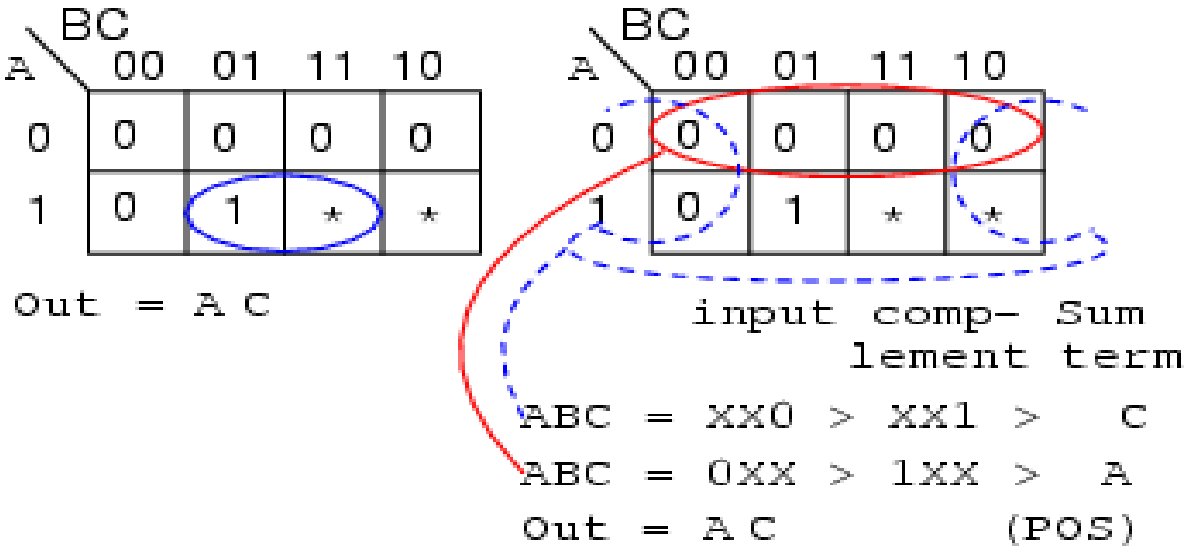


Source adapted from :http://www.allaboutcircuits.com/vol_4/chpt_8/10.html

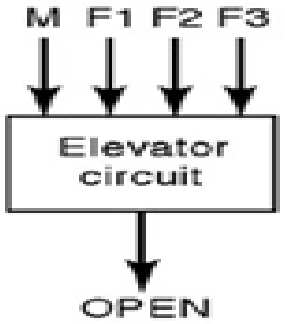
- Simplest solution is just to loop the singular 1 to result in output AB'C



- But if don't care cells are taken into consideration two possible loops can occur (with same conclusion) – one using SOP the other with POS



Example 2



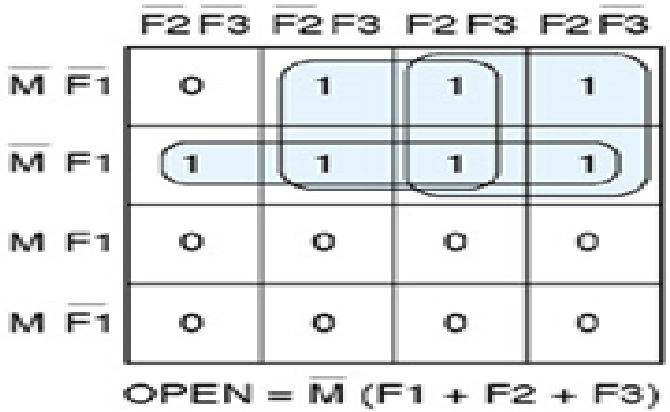
(a)

M	F1	F2	F3	OPEN
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	X
0	1	0	0	1
0	1	0	1	X
0	1	1	0	X
0	1	1	1	X
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	X
1	1	0	0	0
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

(b)

	$\overline{F2}\,\overline{F3}$	$\overline{F2}F3$	$F2F3$	$F2\,\overline{F3}$
$\overline{M}\,\overline{F1}$	0	1	X	1
$\overline{M}F1$	1	X	X	X
$MF1$	0	X	X	X
$M\,\overline{F1}$	0	0	X	0

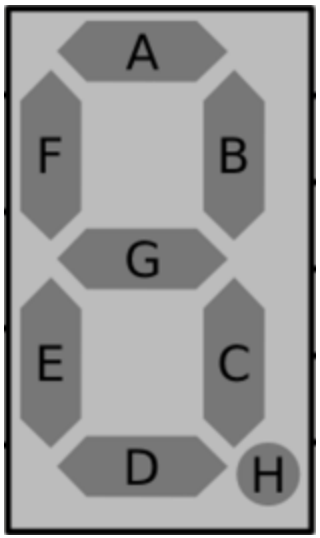
(c)

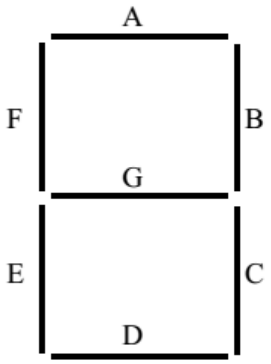


(d)

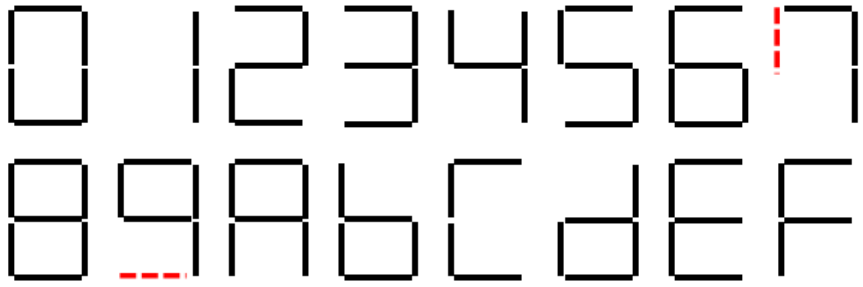
Exercise

- If time permits, design a logic circuit to convert a 4-bit hex input into its equivalents for use in a 7-segment display. Make use of *don't cares* to simplify.



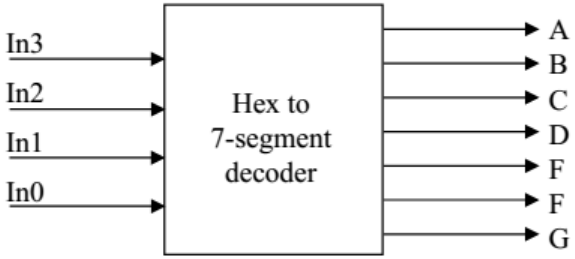


7 segments

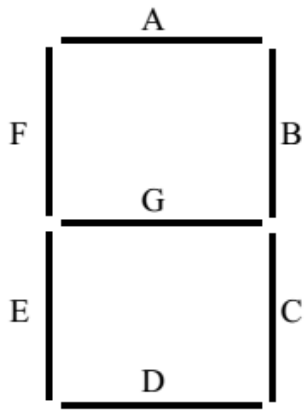


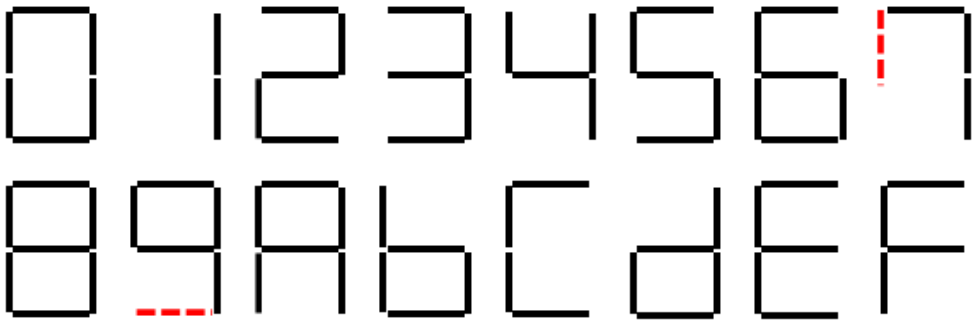
active (on) segments for a given HEX value

= don't care



Circuit block diagram

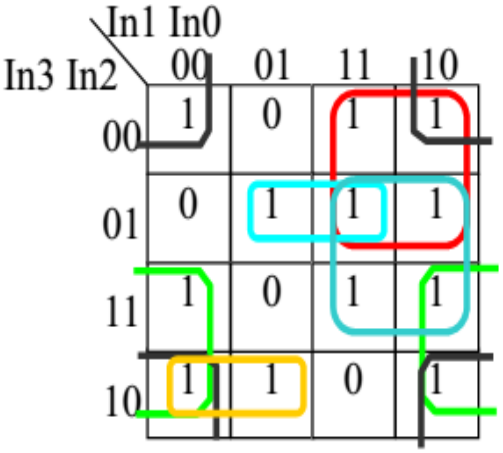




--- = don't care

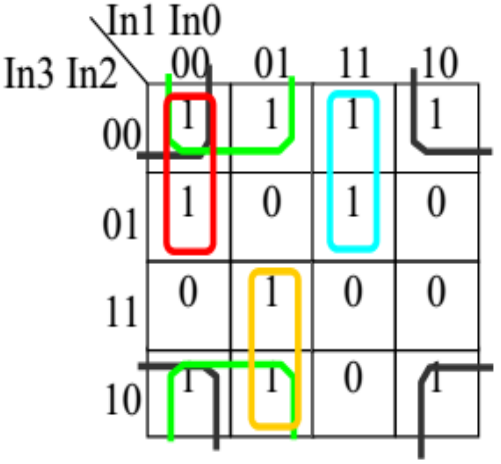
In3	In2	In1	In0	A	B	C	D	E	F	G
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	X	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	X	0	1	1
1	0	1	0	1	1	1	0	1	1	1
1	0	1	1	0	0	1	1	1	1	1
1	1	0	0	1	0	0	1	1	1	0
1	1	0	1	0	1	1	1	1	0	1
1	1	1	0	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	1	1	1

In3	In2	In1	In0	A	B	C	D	E	F	G
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	0	1	1	1	0	1	1	1
1	0	1	1	0	0	1	1	1	1	1
1	1	0	0	1	0	0	1	1	1	0
1	1	0	1	0	1	1	1	1	0	1
1	1	1	0	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	1	1	1



K-map for A output

A = In2'In0' + In3'In1 + In2 In1
+ In3 In0' + In3'In2 In0 + In3 In2'In1'



K-map for B output

B = In2'In0' + In2'In1' + In3'In1'In0'
+ In3 In1'In0 + In3'In1 In0

etc.….

Summary

- K-maps are an alternative to simplifying boolean expressions of limited variable length
- SOP expressions can be simplified by looping/grouping 1s indicated in the K-map graph
- POS expressions can be simplified by looping/grouping 0s in the K-map graph
- Larger loops allow for a more simplified final form
- Don’t cares can be taken into consideration in SOP/POS looping to improve the simplification