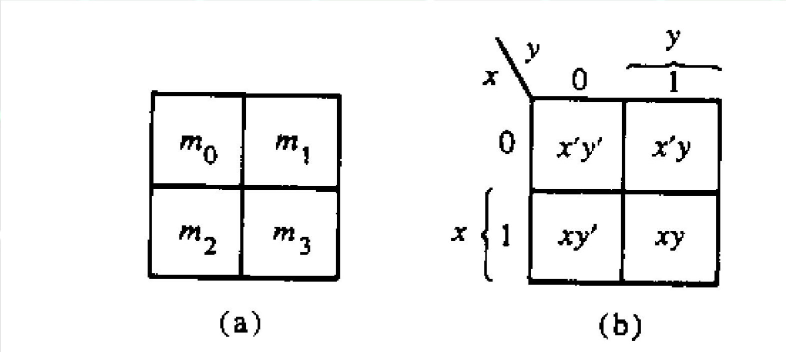
CSE260 Digital Logic Design

Karnaugh Maps

The map method

- The map method provides a simple procedure for minimizing Boolean functions.
- The map is made up of squares where each square represent a minterm.
- We represent each minterm of an equation by ‘1’s
- We groups the adjacent ‘1’s in groups of 2, groups of 4 or groups of 8 and so on.
- In those grouping try to find the common literal.
- The map method is often known as Karnaugh Map or Veitech Diagram. In short we will call it K-map



K Map Grouping Rules :

i) Group the adjacent '1's in groups of 2, 4, 8 and so on [2^n]

ii) Take maximum number of 1 if possible.

iii) You can cover a 1 multiple times only if that covered '1' can help another uncovered '1' to get covered.

iv) Our target is to group all the '1's

v) In case of " don't care " we can consider 'X' to create groups alongside '1's. Other rules stay same as always.

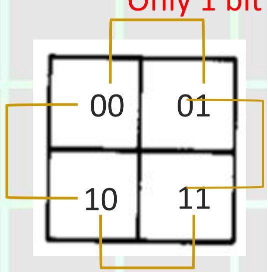
vi) However, you cannot create groups with 'X' only. 'X' can ~~be~~ only be used to create groups if it is helping any uncovered '1's.

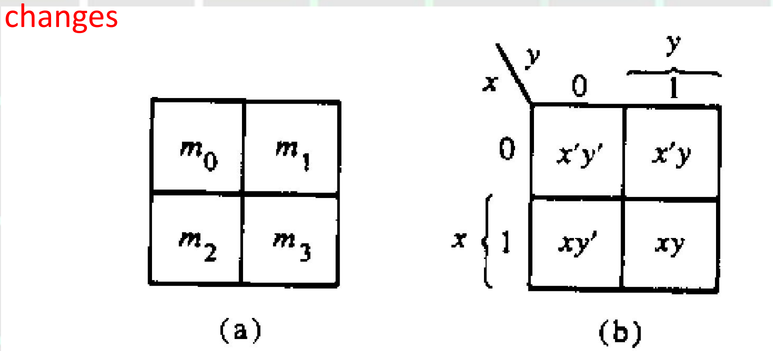
Two variable map

- There are 4 minterm of 2 variable; hence the map is four squares, one for each minterm.

Only 1 bit changes

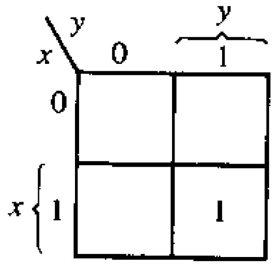
Only 1 bit changes





Representing function in K-map

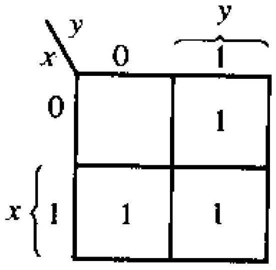
- (a) $F1=xy$;



(a) xy

- (b) $F2=xy'+xy+x'y$

$$=x(y'+y)+x'y$$
$$=x+x'y$$
$$=(x+x')(x+y)$$
$$=(1)(x+y)$$
$$=x+y$$



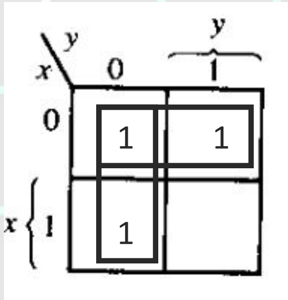
(b) $x+y$

Try it yourself

F=xy'+x'y'+x'y,
Simplify the equation using k-map

Solution

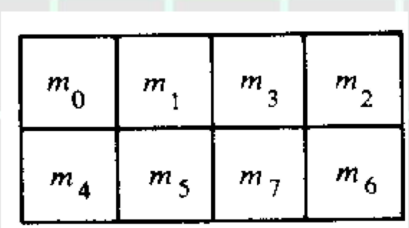
- $F=xy'+x'y'+x'y=x'+y'$

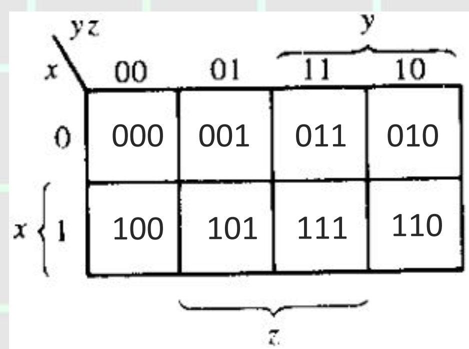


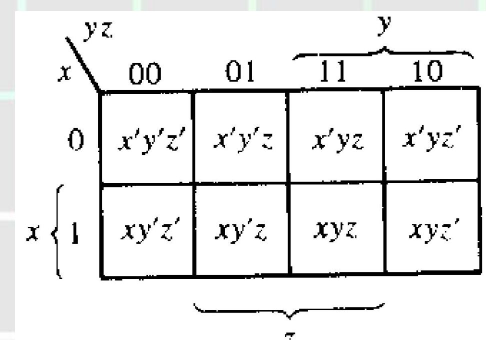
Three variable map

- Sequence in 3-variable map is not in binary sequence but in reflected code. In reflected code, at a time only 1 bit changes from 0 to 1 and 1 to 0.
- Numbering of minterms follow binary numbers. Example m5 is at 101 i.e. row 1 and column 01.

Note: Any 2 adjacent square differ by 1 variable (which is primed in one and unprimed in the other)

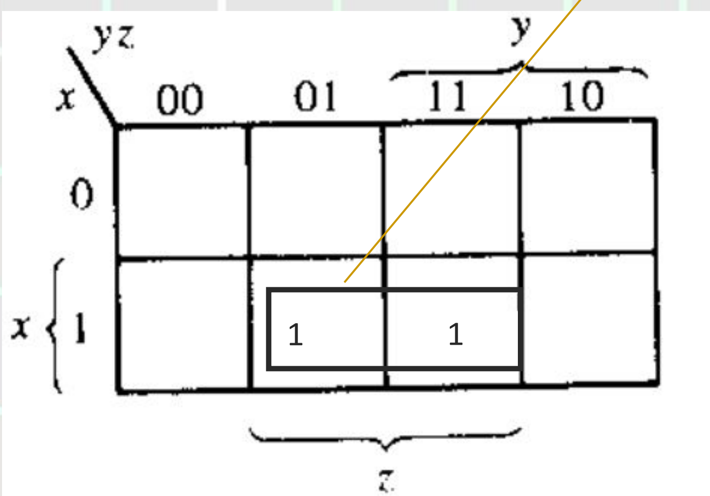


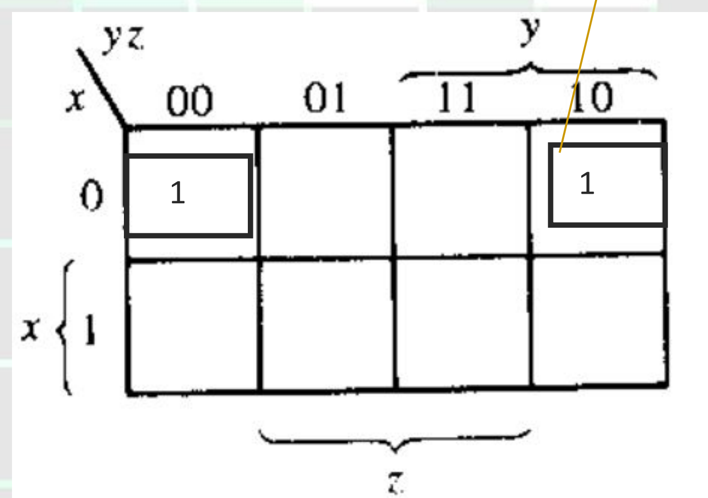




and unprimed in the other) and they are grouped, then any 2 minterms in the adjacent square are ORed, will remove the different term. As a result , we get a single ANDed term of only 2 common literals

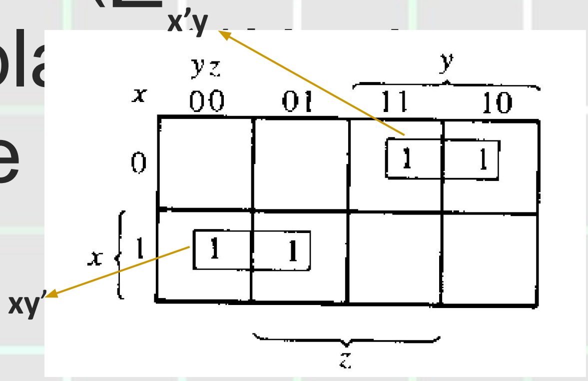
- $F1=\sum(5,7)=xy'z+xyz=xz(y'+y)=xz$
  - $F2=\sum(0,2)=x'y'z'+x'yz'=x'z'(y'+y)=x'z'$





Simplifying functions using k-map

- F=∑(2,3,4,5)=x'yz+x'yz'+xy'z'+xy'z (note: in other words, F= (∑ (010,011,100,101) so pla positions and group the



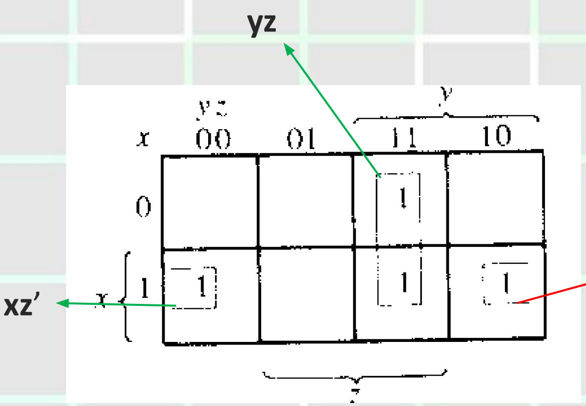
F=x'y+xy'

- Grouping of 1 can be done in groups of

two '1',four '1',eight '1'….2^n '1'.

- $F1=x'yz+xy'z'+xyz+xyz'=xz'+yz$

011 100 111 110

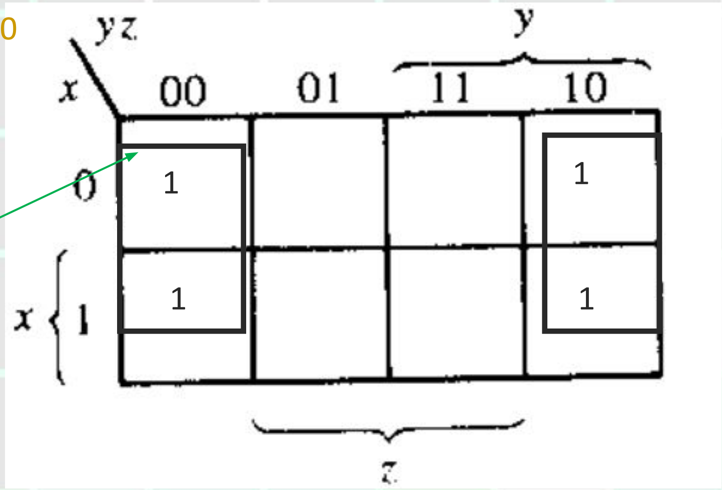


Notice even 2 opposite edges of the map are adjacent, thus they can be grouped

- If F=∑(0,2,4,6), then simplify it

000 010 100 110

=Z’

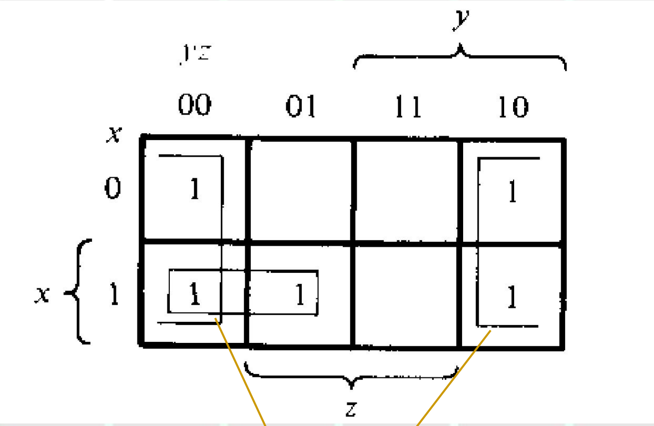


$$
\begin{aligned}
m_0 + m_2 + m_4 + m_6 &= x'y'z' + x'yz' + xy'z' + xyz' \\
&= x'z'(y' + y) + xz'(y' + y) \\
&= x'z' + xz' = z'(x' + x) = z'
\end{aligned}
$$

Try it yourself

- $F=\sum(0,2,4,5,6)$, Simplify it.

Solution



F= xy' + z'

Try it yourself:

- $F=A'C+A'B+AB'C+BC$

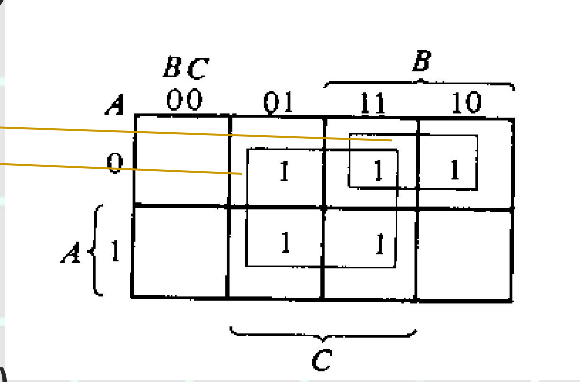
a) Express it as SOP (i.e. $\sum$)

b) Simplify it using Kmap

Solution

- (a) $F= \sum(1,2,3,5,7)$

(b)F=C + A'B



To find A'C, coincide A' (first row) with C(middle 2 column)
To find A'B, coincide A' (first row) with B(last 2 column)
To find BC, coincide B (last 2 column) with C(middle 2 column)
To find AB'C, is m5(2nd row 2nd column)

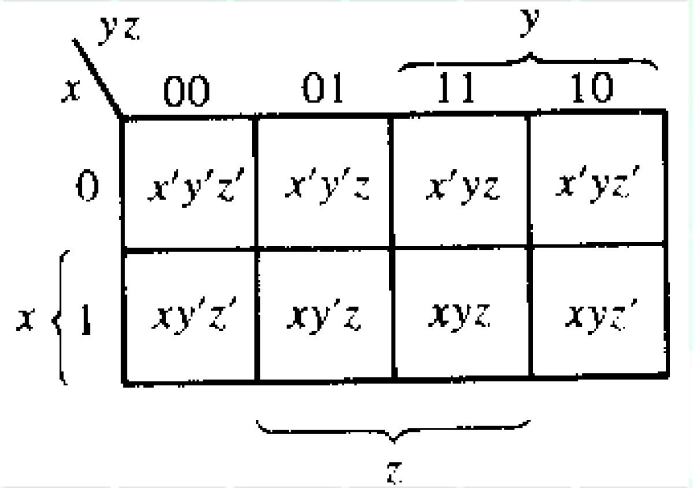
Summary of grouping in 3-variable map

One square represents one minterm, giving a term of three literals.

Two adjacent squares represent a term of two literals.

Four adjacent squares represent a term of one literal.

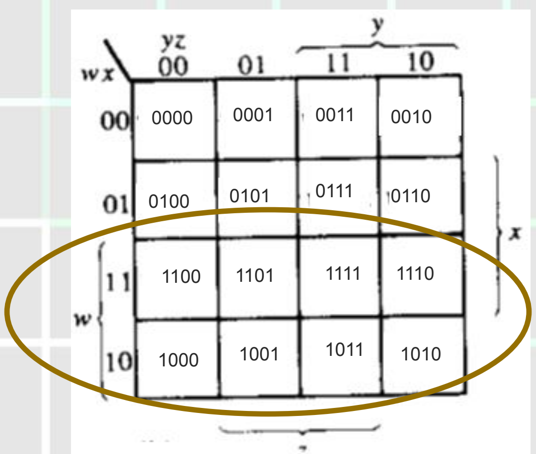
Eight adjacent squares encompass the entire map and produce a function that is always equal to 1.

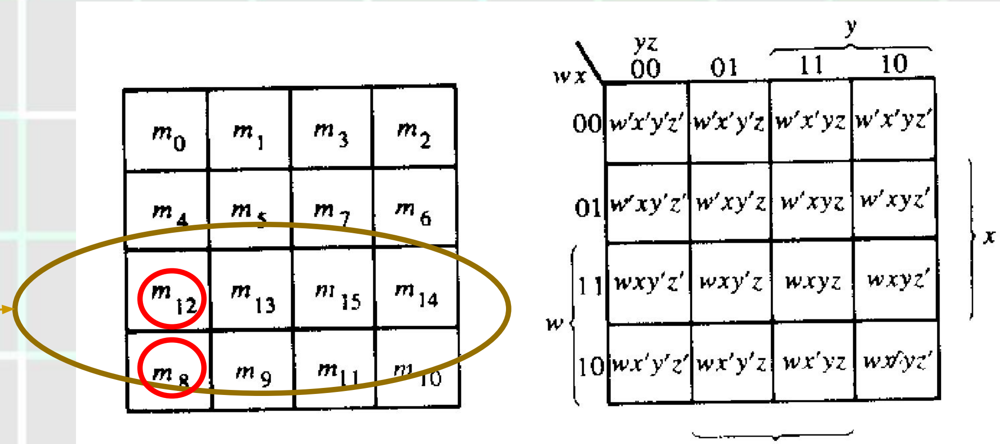


Four variable map

- Adjacent squares can be side by side or lie in 4 corner or top-bottom or left-right edge condition. Example, m0 -m2 are adjacent and similarly m3-m11 are adjacent

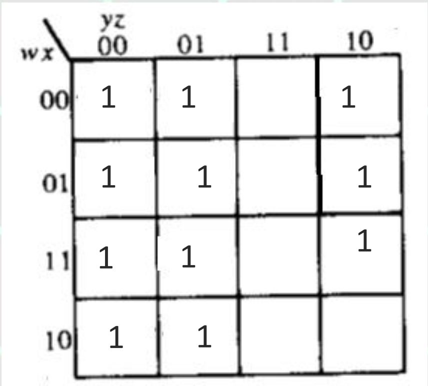
One square represents one minterm, giving a term of four literals.
Two adjacent squares represent a term of three literals.
Four adjacent squares represent a term of two literals.
Eight adjacent squares represent a term of one literal.
Sixteen adjacent squares represent the function equal to 1.

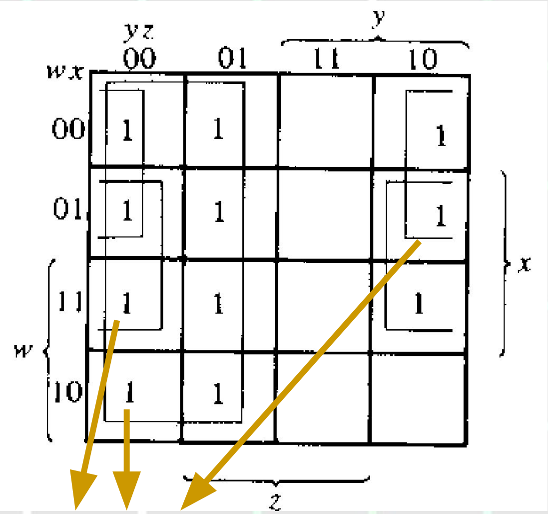




(a) (b)

- Simplify F=∑(0,1,2,4,5,6,8,9,12,13,14)
- Note: it is allowed to use the same ‘1’ more than once.

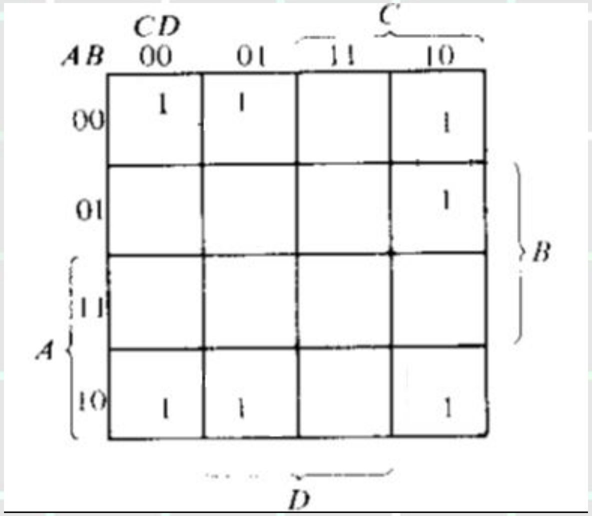




F= xz’+ y’+ w’z’

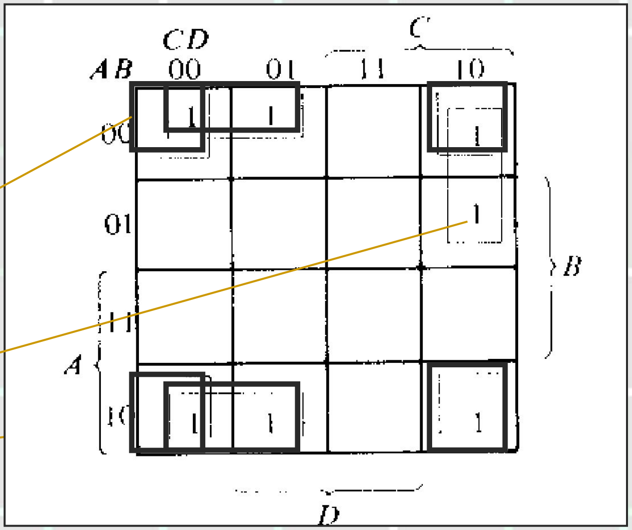
Try it yourself

- F=A'B'C'+B'CD'+A'BCD'+AB'C'



Solution

- F=A’B’C’+B’CD’+A’BCD’+AB’C’



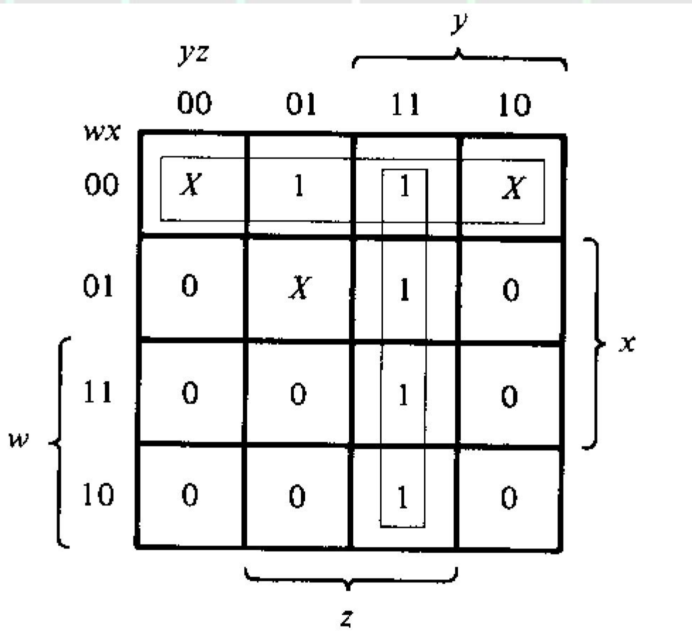
F=B’D’+B’C’+A’CD’

Don’t Care condition

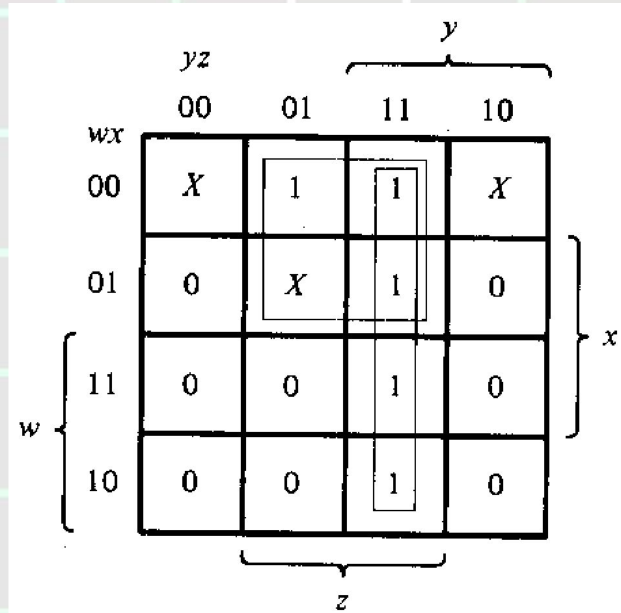
- Don't cares in a Karnaugh map, or truth table, may be either **1**s or **0**s, as long as we don't care what the output is for an input condition we never expect to see.
- We plot these cells with a special sign, ‘X’, among the normal **1**s and **0**s.
- When forming groups of cells, treat the don't care cell as either a **1** or a **0**, or ignore the don't cares. 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. (i.e. do not make any group only consisting the don’t cares). Only use them in a group if it simplifies the logic.

Simplify F(w,x,y,z)= ∑(1,3,7,11,15) and don’t care d(w,x,y,z)= ∑(0,2,5)

Solution: Even though 2 solutions are not same, but either 1 is acceptable. Such scenario only applicable for Don’t care scenarios!



(a) $F = yz + w'x'$



(b) $F = yz + w'z$

APPLICATIONS OF K-MAP

Example 1: Design and draw the circuit diagram of BCD to Excess-3 code converter

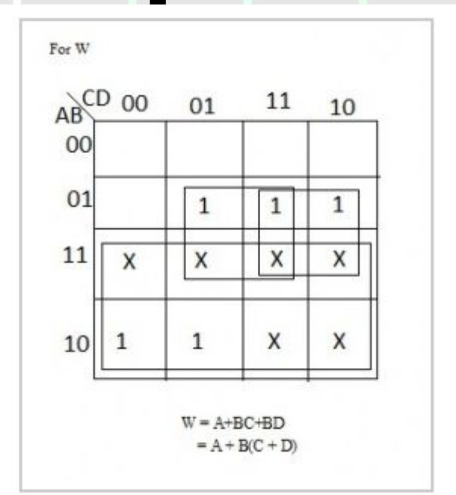
BCD Input				Excess-3 Output			
B3	B2	B1	B0	W	X	Y	Z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	x

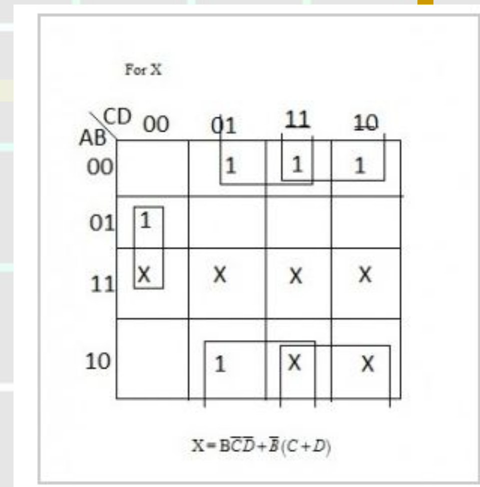
Q1.Treat each W, X, Y and Z as a function and write them as SOP

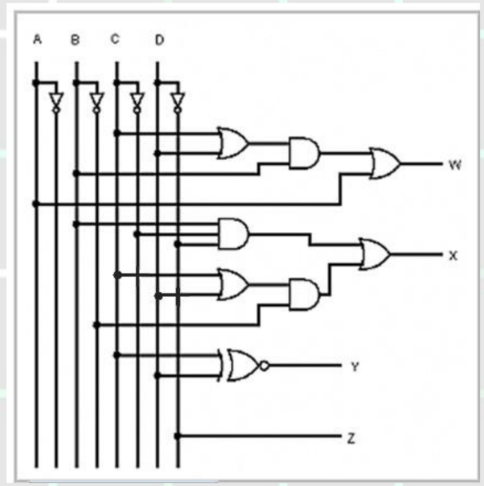
Q2. Based on the SOP, draw the circuit diagram of BCD to Excess-3 code converter

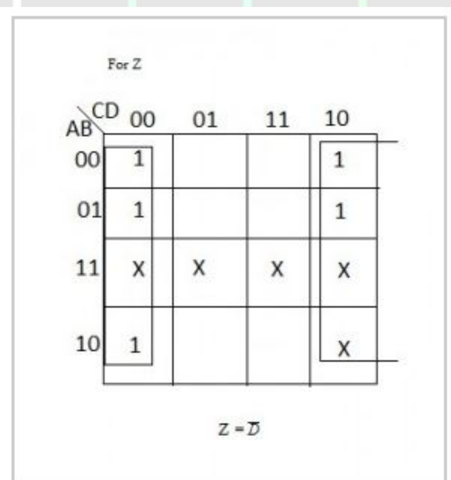
W X
Y Z

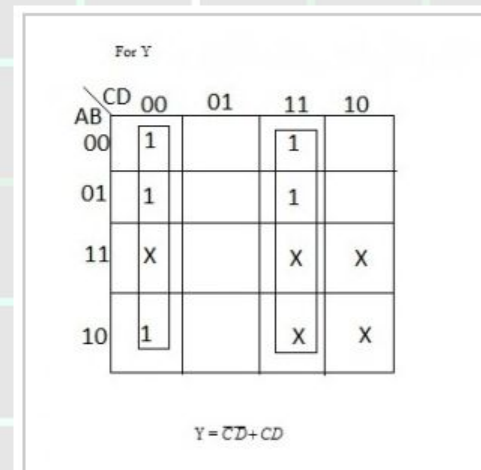
Any value higher than 9 will result into don't care output


For W
$W = A + BC + BD$
$= A + B(C + D)$


For X
$X = B\overline{C}\overline{D} + \overline{B}(C + D)$

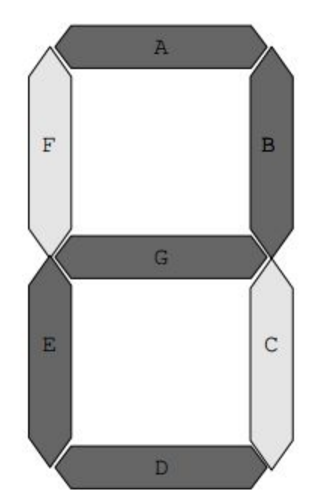

A
B
C
D
W
X
Y
Z


For Z
$Z = \overline{D}$


For Y
$Y = \overline{C}\overline{D} + CD$

Example 2: 7 segment display board

Display digit	b8	b4	b2	b1	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	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1
off	1	0	1	0	×	×	×	×	×	×	×
off	1	0	1	1	×	×	×	×	×	×	×
off	1	1	0	0	×	×	×	×	×	×	×
off	1	1	0	1	×	×	×	×	×	×	×
off	1	1	1	0	×	×	×	×	×	×	×
off	1	1	1	1	×	×	×	×	×	×	×



Segment a

		!C !D	!C D	C D	C !D
!A	!B	1	0	1	1
!A	B	0	1	1	0
A	B	D	D	D	D
A	!B	1	1	D	D

A + CD + B D + !B !D

Segment b

		!C !D	!C D	C D	C !D
!A	!B	1	1	1	1
!A	B	1	0	1	0
A	B	D	D	D	D
A	!B	1	1	D	D

!B + !C !D + C D

Segment c

		!C !D	!C D	C D	C !D
!A	!B	1	1	1	0
!A	B	1	1	1	1
A	B	D	D	D	D
A	!B	1	1	D	D

B + !C + D

Segment d

		!C !D	!C D	C D	C !D
!A	!B	1	0	1	1
!A	B	0	1	0	1
A	B	D	D	D	D
A	!B	1	1	D	D

A + C !D + !B !D + !B C + B !C D

Segment e

		!C !D	!C D	C D	C !D
!A	!B	1	0	0	1
!A	B	0	0	0	1
A	B	D	D	D	D
A	!B	1	0	D	D

C !D + !B !D

Segment f

		!C !D	!C D	C D	C !D
!A	!B	1	0	0	0
!A	B	1	1	0	1
A	B	D	D	D	D
A	!B	1	1	D	D

A + B !C + !C !D + B !D

Segment g

		!C !D	!C D	C D	C !D
!A	!B	0	0	1	1
!A	B	1	1	0	1
A	B	D	D	D	D
A	!B	1	1	D	D

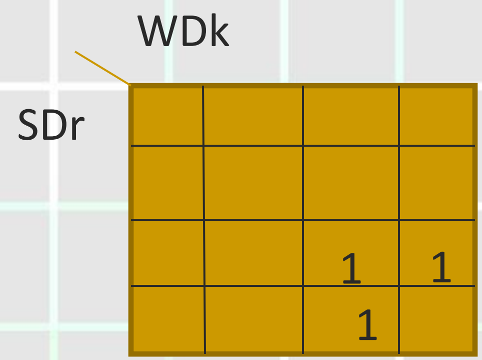
A + B !C + B !D + !B C

Try it by your self

- A **car garage** has a front door and one window, each of which has a sensor to detect whether it is open. A third sensor detects whether it is dark outside. A security system for the garage follows this rule: the alarm rings if the **alarm switch is turned on** and **either the front door is not closed or it is dark** and **the side window is not closed**.

Solution

Switch On	Door	Window	Dark	AlarmRing
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1



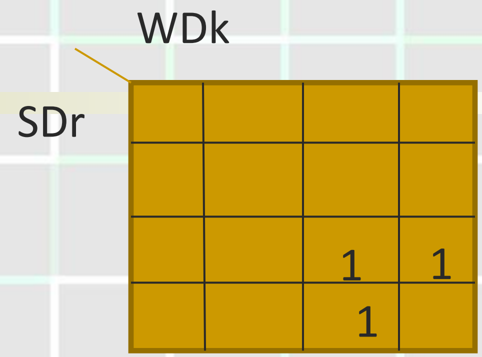
AlarmRing=SWDk+SDrW

Assuming

AlarmRing = 1 if Alarm rings, 0 otherwise.
Switch On = 1 if alarm switch is on, 0 otherwise
Door = 1 if door is not closed, 0 otherwise.
Window = 1 if window is not closed.
Dark = 1 if it is dark outside, 0 otherwise.

Solution

Switch On	Door	Window	Dark	AlarmRing
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1



AlarmRing=SWDk+SDrW

Next Step

- **You do have to draw the circuit for the above function**
- **You may be asked to draw the circuit using basic gates or universal gates**
- **By now, you should be able to do it yourself. So please try.**