Unit-3
Simplification of Boolean Functions

For more notes visit:

https://collegenote.pythonanywhere.com/

Unit-3
Simplification of Boolean Functions

Different forms of Boolean Algebra

1. **Sum of Product (SOP)**
   This is an expression in which each term is a product term and all the product terms are summed together.
   a) Minimal SOP
      An expression in which each product term consist of minimum numbers of variables.
      E.g. XY+X'Y+XY'

   b) Expanded SOP
      An expression in which each product term consist of maximum numbers of variables.
      E.g. XYZ+X'YZ+WXYZ

2. **Product of Sum (POS)**
   This is an expression in which several sum terms are multiplied together.
   a) Minimal POS
      In this an expression in which there are minimum number of sum terms.
      E.g. (X+Y) (X'+Y) (X+Y')

   b) Expanded POS
      In this an expression in which there are maximum number of variables.
      E.g. (X+Y+Z)·(X'+Y'+Z')·(W+X+Y+Z)

Minterms or standard product

- Each row of a truth table can be associated with a *minterm*, which is a product (AND) of all variables in the function, in direct or complemented form.
- A function with n variables has 2^n minterms.
- A minterm has the property that it is equal to 1 on exactly one row of the truth table.

Maxterms or standard sums

- Each row of a truth table is also associated with a *maxterm*, which is a sum (OR) of all the variables in the function, in direct or complemented form.
- A function with n variables has 2^n maxterms
- A maxterm has the property that it is equal to 0 on exactly one row of the truth table.

Variable			Minterm		Maxterm	
x	y	z	Term	Designation	Term	Designation
0	0	0	x'y'z'	m_0	x+y+z	M_0
0	0	1	x'y'z	m_1	x+y+z'	M_1
0	1	0	x'yz'	m_2	x+y'+z	M_2
0	1	1	x'yz	m_3	x+y'+z'	M_3
1	0	0	xy'z'	m_4	x'+y+z	M_4
1	0	1	xy'z	m_5	x'+y+z'	M_5
1	1	0	xyz'	m_6	x'+y'+z	M_6
1	1	1	xyz	m_7	x'+y'+z'	M_7

Note: Each maxterm is the complement of its corresponding minterm and vice versa.

Expressing Boolean function by sum of minterms

A boolean function may be expressed algebraically from a given truth table by forming a minterm for each combination of the variables which produces a 1 in the function and then taking the OR of all those terms.

x	*y*	*z*	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

$$f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$$

Finding Complement of Boolean function by sum of minterms

The complement of a Boolean function can be obtained from the truth table by forming a minterm of each combination that produces a 0 in the function and then ORing those terms.

Functions of Three Variables

x	y	z	Function f_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

The Complement of f_1 is written as:

$$f_1' = x'y'z' + x'yz' + x'yz + xy'z + xyz'$$

Expressing Boolean Function by product of maxterms

A boolean function may be expressed algebraically from a given truth table by forming a maxterm for each combination of the variables which produces a 0 in the function and then taking the AND of all those terms.

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$f_1 = (x + y + z)(x + y' + z)(x + y' + z')(x' + y + z')(x' + y' + z)$$
$$= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6$$
$$f_2 = (x + y + z)(x + y + z')(x + y' + z)(x' + y + z)$$
$$= M_0M_1M_2M_4$$

Canonical and Standard forms

- Boolean functions expressed as a sum of minterms or product of maxterms are said to be in **canonical form**.
- In an expression in canonical form, every variable appears in every term.
- The two canonical forms of Boolean algebra are basic forms that one obtain from reading a function from the truth table. These forms are very seldom the ones with least number of literals, because **each minterm or maxterm must contain, by definition, all the variables either complemented or uncomplemented**.

- Another way to express Boolean functions is in **standard form**. In this configuration, the terms that form the function may contain one, two or any number of literal.

- There are two types of standard forms: the sum of products and product of sums.
  Sum of products: $F_1 = y' + xy + x'yz'$
  Product of sums: $F_2 = x(y' + z)(x' + y + z' + w)$

Expressing Boolean function as a sum of minterms and product of maxterms using Boolean algebra:

- To express the Boolean function as a sum of minterms, expand the Boolean function into a sum of products. Each term is then inspected to see if it contains all the variables. If it misses one or more variables, it is ANDed with an expression such as $x + x'$, where x is one of the missing variables.

- To express the Boolean function as a product of maxterms, expand the Boolean function into a product of sums. This may be done by using the distributive law, $x + yz = (x + y)(x + z)$. Then any missing variable x in each OR term is ORed with xx'.

Q. Express the Boolean function $F = A + B'C$ in a sum of minterms.
Soln:
The function has three variables, the first term A is missing two variables B and C.
Inclusion of variable B: $A = A(B + B') = AB + AB'$
Inclusion of variable of C: $A = AB(C + C') + AB'(C + C')$
$= ABC + ABC' + AB'C + AB'C'$

The second term $B'C$ is missing one variable A.
Inclusion of variable A: $B'C = B'C(A + A') = AB'C + A'B'C$

Combining all terms
$F = A + B'C$
$= ABC + ABC' + \boldsymbol{AB'C} + AB'C' + \boldsymbol{AB'C} + A'B'C$
$F = A'B'C + AB'C' + AB'C + ABC' + ABC$
$= m_1 + m_4 + m_5 + m_6 + m_7$

$F(A, B, C) = \sum(1, 4, 5, 6, 7)$

Q. Express the Boolean function $F = xy + x'z$ in a product of maxterm form.
Soln:
Converting the function into OR terms using distributive law:

$$F = xy + x'z = (xy + x')(xy + z)$$
$$= (x + x')(y + x')(x + z)(y + z)$$
$$= (x' + y)(x + z)(y + z)$$

Including missing with each term:

$$x' + y = x' + y + zz' = (x' + y + z)(x' + y + z')$$
$$x + z = x + z + yy' = (x + y + z)(x + y' + z)$$
$$y + z = y + z + xx' = (x + y + z)(x' + y + z)$$

Combining and avoiding the repeated terms:

$$F = (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z')$$
$$= M_0M_2M_4M_5$$
$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

Conversion between canonical forms

$$\boxed{m_j' = M_j}$$

Consider a function

$F(A, B, C) = \sum(1, 4, 5, 6, 7)$

Taking the complement of F

$F'(A, B, C) = \sum(0, 2, 3) = m_0 + m_2 + m_3$

Taking the complement of F'

$F = (m_0 + m_2 + m_3)' = m_0' \cdot m_2' \cdot m_3' = M_0M_2M_3 = \prod(0, 2, 3)$

Similarly,

$F(x, y, z) = \sum(1, 3, 6, 7) \leftrightarrow F(x, y, z) = \prod(0, 2, 4, 5)$

Karnaugh Map (K-Map)

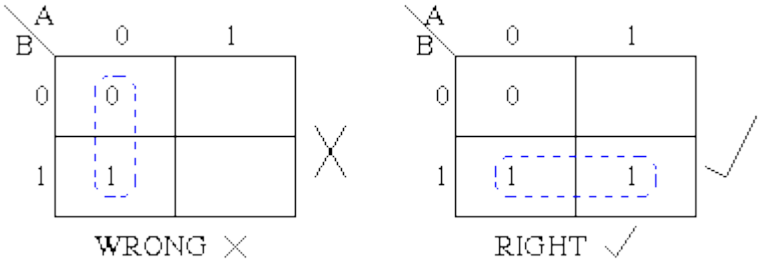
- **Karnaugh map** or **K-map** is a map of a function used in a technique used for minimization or **simplification of a Boolean expression**.
- Booleans expression can be simplified using Boolean algebraic theorems but there are no specific rules to make the most simplified expression. However, K-map can easily minimize the terms of a Boolean function.
- Unlike an algebraic method, K-map is a pictorial method and it does not need any Boolean algebraic theorems.
- K-map is basically a diagram made up of squares. Each of these squares represents a min-term of the variables. If n = number of variables then the number of squares in its K-map will be 2^n. K-map is made using the truth table.
- Karnaugh map can produce **Sum of product (SOP)** or **product of Sum(POS)** expression considering which of the two (0,1) outputs are being grouped in it. The grouping of 0's result in Product of Sum expression & the grouping of 1's result in Sum of Product expression.

Rules of minimization in K-Map

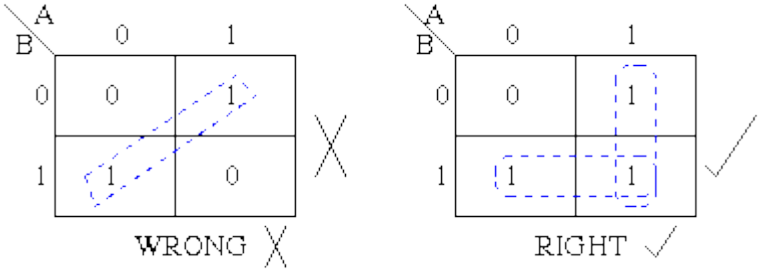
1. While grouping, you can make groups of 2^n number where n=0, 1, 2, 3…
2. You can either make groups of 1's or 0's but not both.
3. Grouping of 1's lead to Sum of Product form and Grouping of 0's lead to Product of Sum form.
4. While grouping, the groups of 1's should not contain any 0 and the group of 0's should not contain any 1.
5. The function output for 0's grouping should be complemented as F'.

Rules for grouping 1's (Sum of Product form):

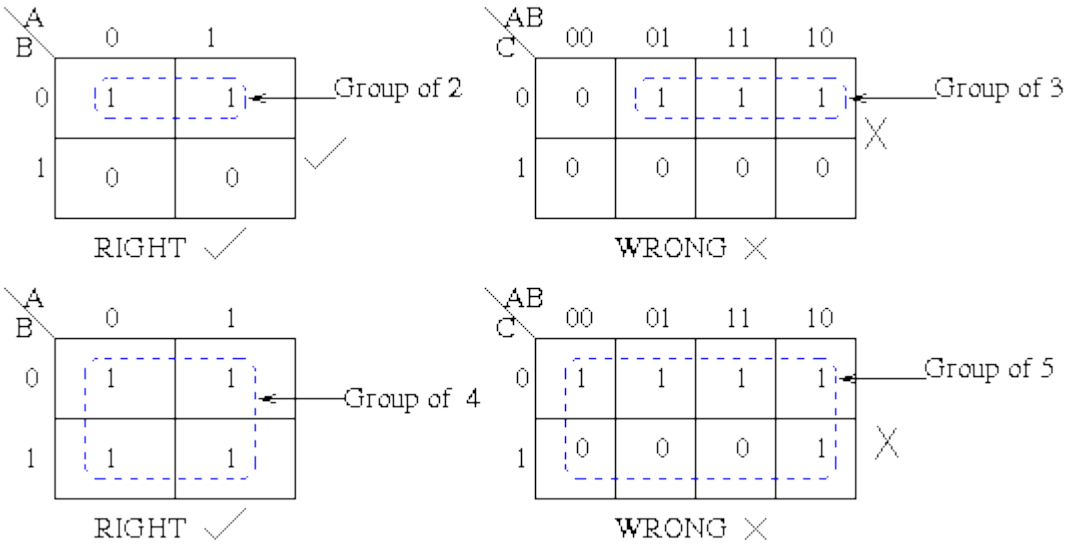
- Groups may not include any cell containing a zero.



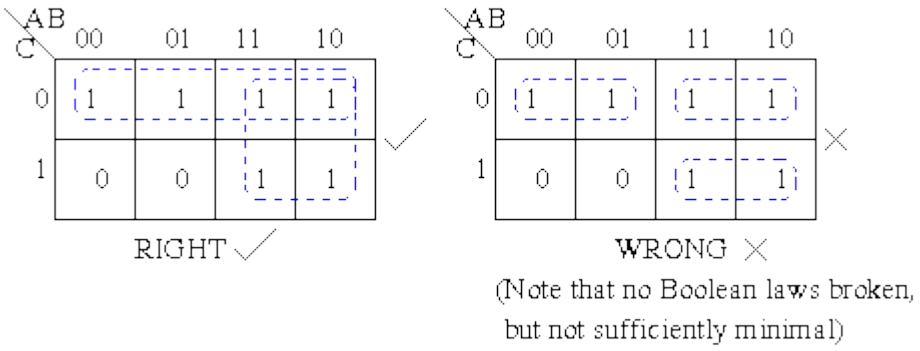
- Groups may be horizontal or vertical, but not diagonal.



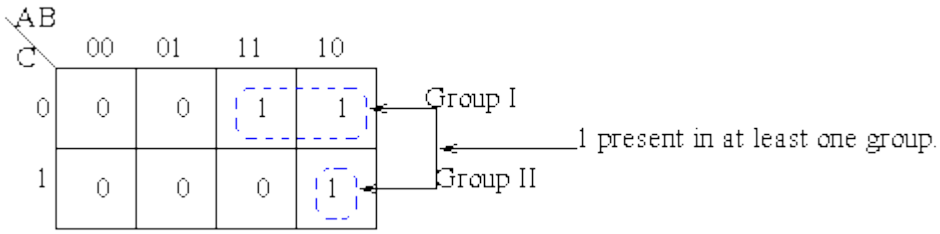
- Groups must contain 1, 2, 4, 8, or in general 2^n cells. That is if n = 1, a group will contain two 1's since $2^1 = 2$. If n = 2, a group will contain four 1's since $2^2 = 4$.



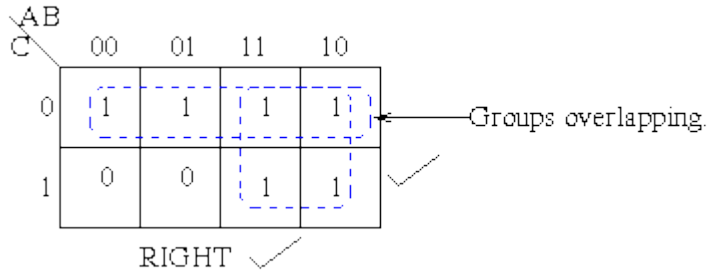
- Each group should be as large as possible.

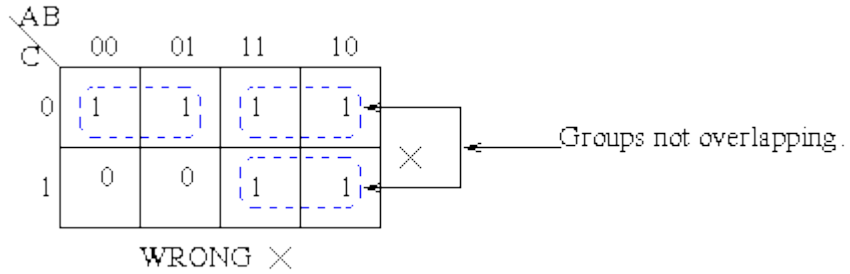


- Each cell containing a *one* must be in at least one group.

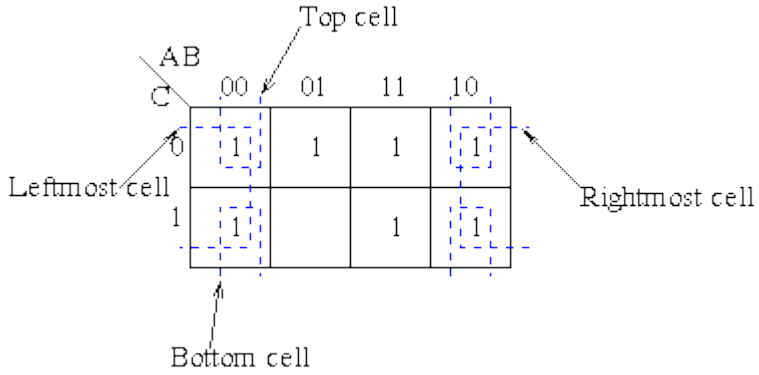


- Groups may overlap.

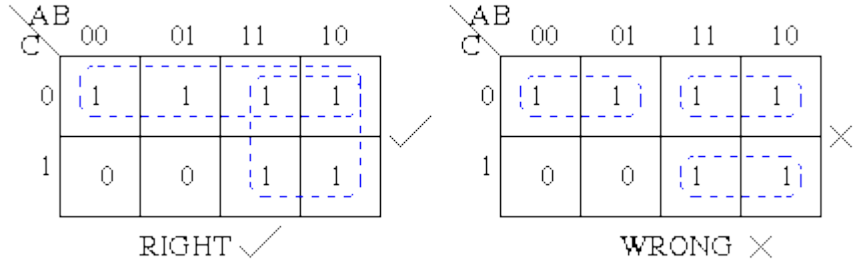




- Groups may wrap around the table. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.



- There should be as few groups as possible, as long as this does not contradict any of the previous rules.



Two Variable Map

There are four minterms for two variables; hence the map consists of fours squares, one of each minterm.

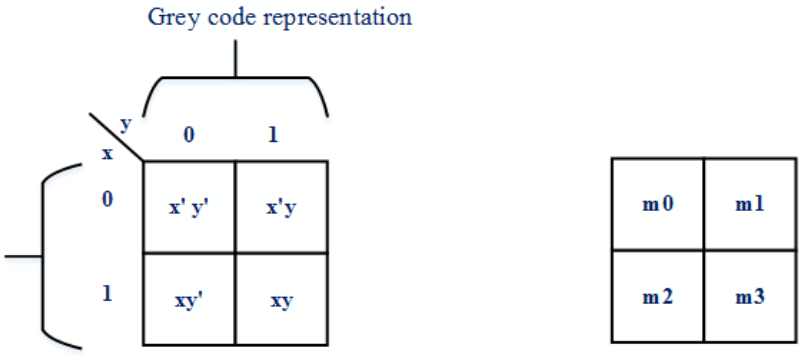
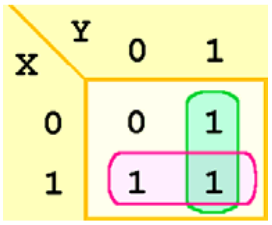


Fig: Two variable map

Q. Simplify the Boolean function $F = x'y + xy' + xy$ using K-Map.

Sol^n:

K-map for given function:



So, after simplification, $F = X + Y$

Three Variable Map

- There are eight minterms for three variables, i.e. the map consist of eight squares.
- Minterms are not arranged in binary sequence but in a sequence similar to gray code/reflected code.

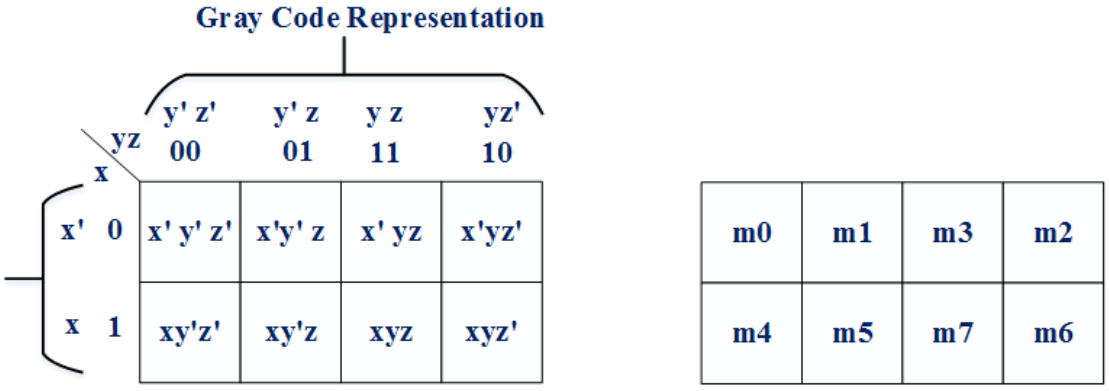
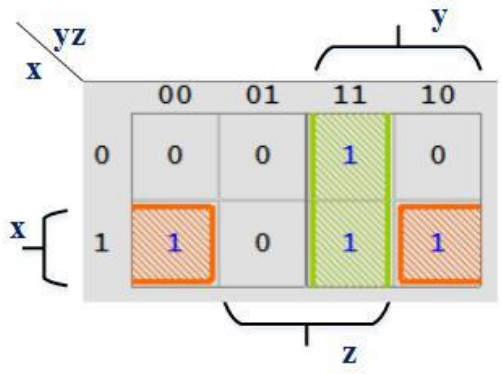


Fig: Three variable map

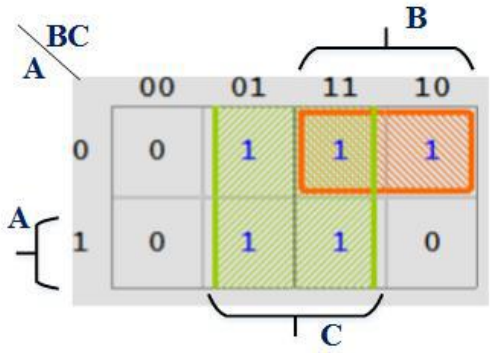
Q. Simplify the Boolean function, $F = X'YZ + XY'Z' + XYZ + XYZ'$ using K-Map.
Sol^n:
K-map for given function:



After simplification, $F = YZ + XZ'$

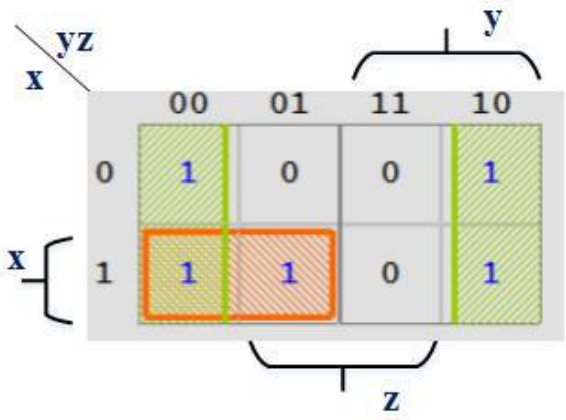
Q. Simplify the Boolean function, $F = A'C + A'B + AB'C + BC$ using K-Map.
Sol^n:
$F = A'C + A'B + AB'C + BC = \boldsymbol{A'BC} + A'B'C + \boldsymbol{A'BC} + A'BC' + AB'C + ABC + \boldsymbol{A'BC}$

K-map for given function:



After simplification, $F = C + A'B$

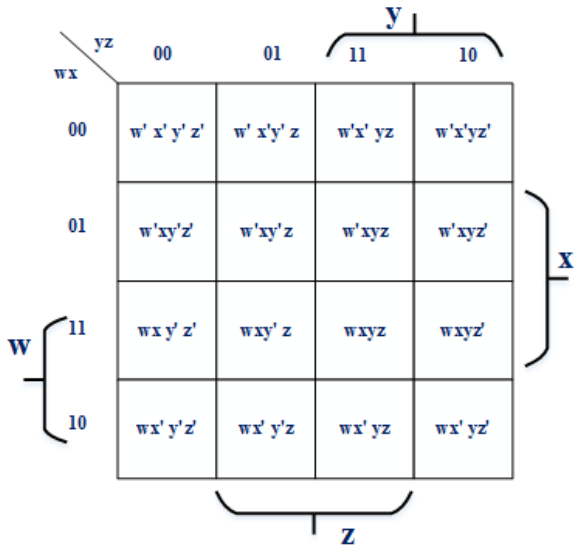
Q. Simplify the Boolean function: $F(X,Y,Z) = \sum(0,2,4,5,6)$ using three variable K-map.
Sol^n:
K-map for given function:



After simplification, $F = Z' + XY'$

Four Variable Map

The map for Boolean function of four binary variables has 16 minterms and the squares assigned to each.



m0	m1	m3	m2
m4	m5	m7	m6
m12	m13	m15	m14
m8	m9	m11	m10

Fig: Four variable map

- The rows and columns are numbered in a reflected code sequence, with only one digit changing value between two adjacent rows and columns.
- The minterm corresponding to each square can be obtained from the concatenation of the row number with the column number.

Q. Simplify the Boolean function $F = A'B'C' + B'CD' + A'BCD' + AB'C'$ using K-map.

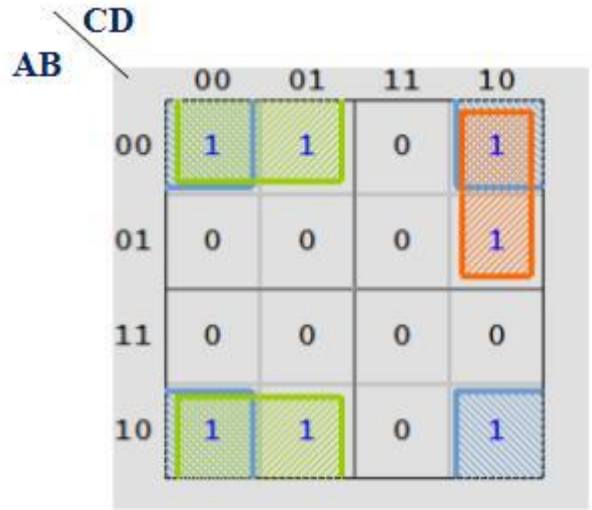
Sol^n:

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

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

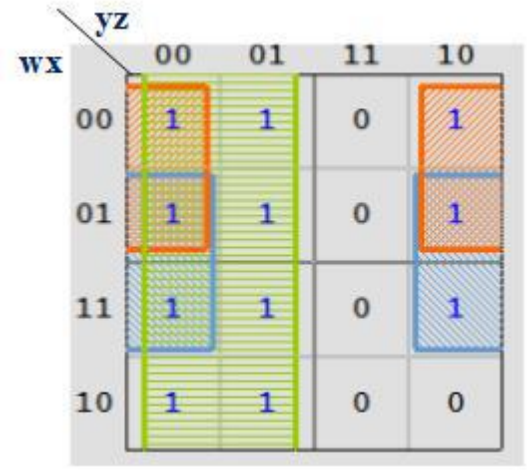
$= A'B'C'D + A'B'C'D' + AB'CD + A'B'CD + A'BCD' + AB'C'D + AB'C'D'$

K-map for given function:



After simplification, $F = B'D' + B'C' + A'CD'$

Q. Simplify the Boolean function $F(W,X,Y,Z) = \sum(0,1,2,4,5,6,8,9,12,13,14)$.
Sol^n:
K-map for given function:



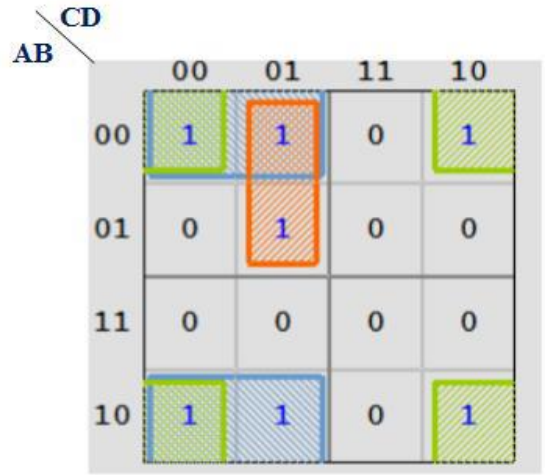
After simplification, $F = Y' + W'Z' + XZ'$

Product of sum simplification

- All previous examples are in sum-of-products form.
- To obtain the product-of-sum form:
  - Simplify F' in the form of sum of products. [If we mark the empty squares by 0's and combine them into valid rectangles, we obtained a simplified expression of the complement of the function, i.e. of F']
  - Apply DeMorgan's theorem $F = (F')'$
  - ***F'*: sum of products ⇒ *F* :** product of sums

Q. Simplify the Boolean function: $F = \sum(0,1,2,5,8,9,10)$ in (a) sum of product (SOP) and (b) product of sum (POS).
Sol^n:

(a) Sum of products simplification

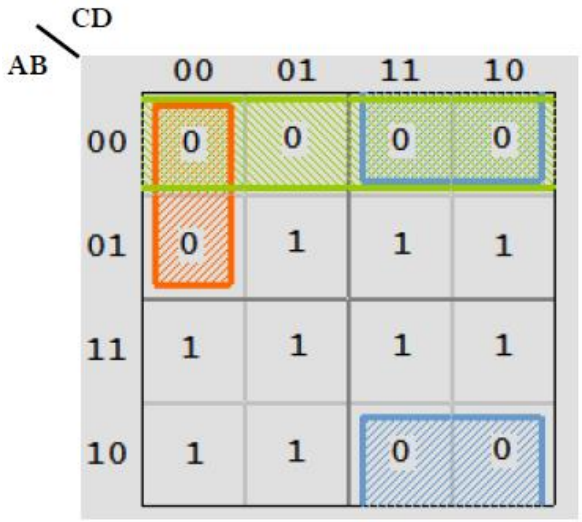


$F = B'D' + B'C' + A'C'D$

(b) Product of sums simplification

$F' = AB + CD + BD'$
So, $F = (A' + B')(C' + D')(B' + D)$

Q. Simply the Boolean function $F(A,B,C,D) = \prod(0,1,2,3,4,10,11)$ in POS.
Sol^n:
K-map for given function:



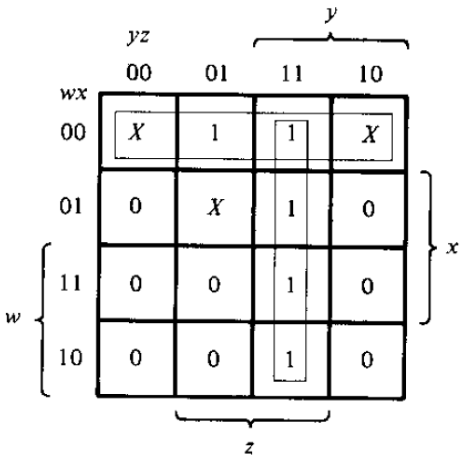
From Map,
$F' = A'B' + B'C + A'C'D'$

So, $F = (A + B)(B + C')(A + C + D)$

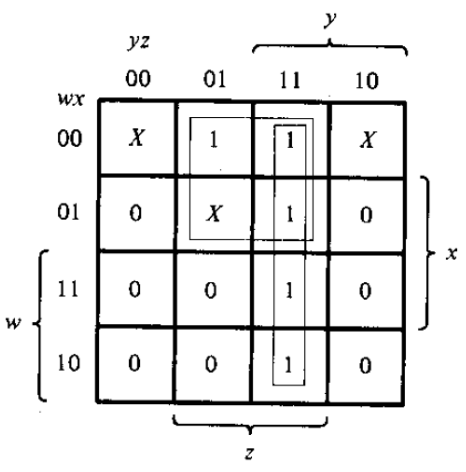
Don't Care Conditions

- There are certain situations where a function may not be completely specified, meaning there may be some input combination that are **undefined** for the function.
- **Real circuits don't** always need to have an **output** defined for every possible input.
- The unused combinations is known as don't care conditions and can be used on the map to provide further simplification of the boolean expression.
- It should be realized that a don't care minterm is a combination of variables whose logical value is not specified. It cannot be marked with a 1 or, 0 in the map as it is not specified as 0 or 1. To distinguish the don't care condition from 1's and 0's, an X is used. Thus, an X inside a square in the map indicates that we don't care whether the value of 0 or 1 is assigned to F for the particular minterms.

Q. Simplify $F(w,x,y,z) = \sum(1,3,7,11,15)$ which has the don't care conditions $d(w,x,y,z) = \sum(0,2,5)$.
Sol^n:



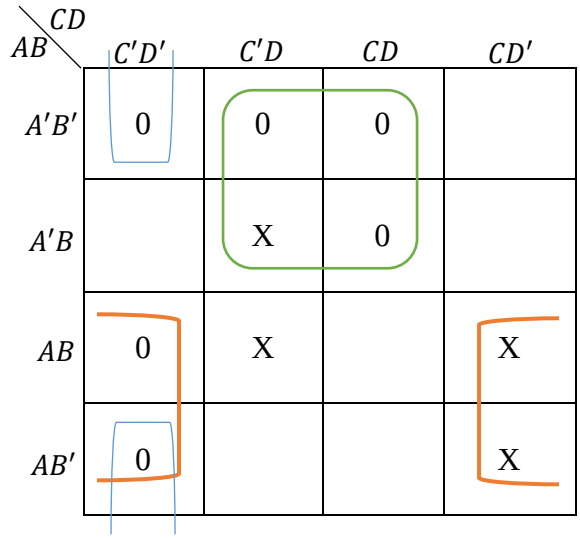
(a) $F = yz + w'x'$



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

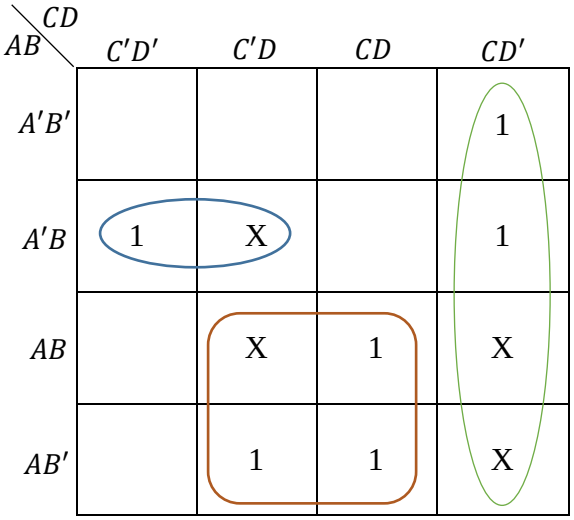
Q. $F(A,B,C,D) = \prod(0,1,3,7,8,12)$ and $\prod d(5,10,13,14)$. Obtain SOP and POS.
Sol^n:

For POS:



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

For SOP:



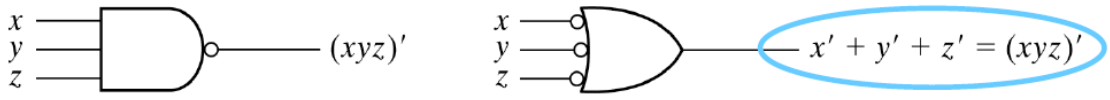
$F = CD' + AD + A'BC'$

NAND and NOR Implementation

Why NAND and NOR implementation?

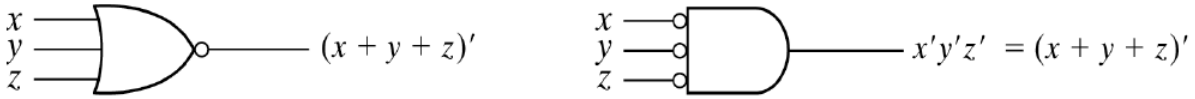
- Digital circuits are frequently constructed with NAND/NOR rather than with AND/OR gates.
- NAND and NOR gates are easier to fabricate with electronic components than AND/OR.
- Cheaper (lower cost) and faster(less delay).
- Any Boolean function can be constructed using only NAND or only NOR gates. That's why NAND and NOR are known as universal gates.

Two graphic symbols for NAND gate:



(a) AND–invert (b) Invert–OR

Two graphic symbols for NOR gate:

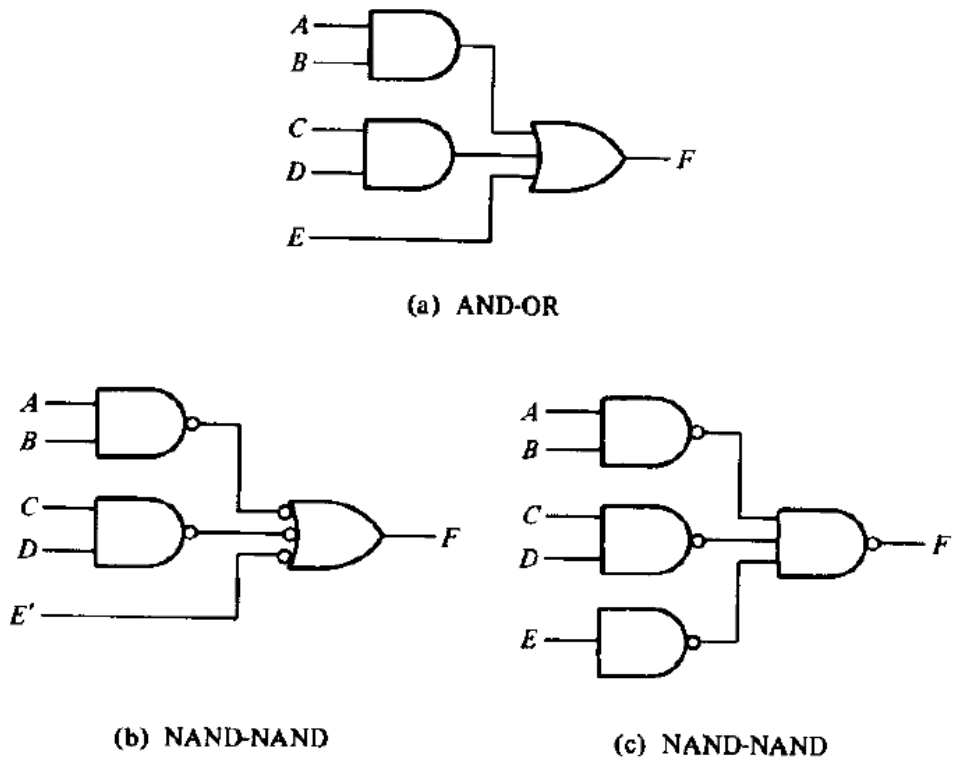


(a) OR–invert (a) Invert–AND

NAND Implementation

The implementation of a Boolean function with NAND gates requires that the function be simplified in the sum of products form.

E.g. Implementation of $F = AB + CD + E$ by NAND gate only.



(a) AND-OR

(b) NAND-NAND

(c) NAND-NAND

Fig: Three ways to implement $F = AB + CD + E$

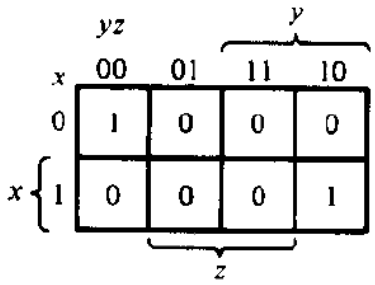
Rules for obtaining the NAND logic diagram from a Boolean function:

1) Simplify the function and express it in sum of products.
2) Draw a NAND gate for each product term of the function that has at least **two literals**. The inputs to each NAND gate are the literals of the term. This constitutes a group of first-level gates.
3) Draw a single NAND gate (using the AND-invert or invert-OR graphic symbol) in the second level, with inputs coming from outputs of the 1st level.
4) A term with a single literal requires an inverter in the first level or may be complemented and applied as an input to the second-level NAND gate.

Note: If we simplify the function combining 0's in a map, we obtain the simplified expression of the complement of the function in sum of product. The complement of the function can then be implemented with two levels of NAND gates using the rules stated above. If the normal output is desired, it would be necessary to insert a one-input NAND gate.

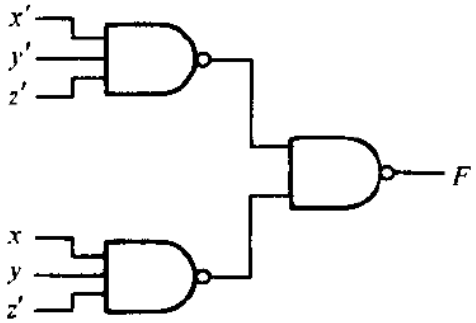
Q. Implement the following function with NAND gates.
$F(x, y, z) = \sum(0, 6)$

Sol^n:

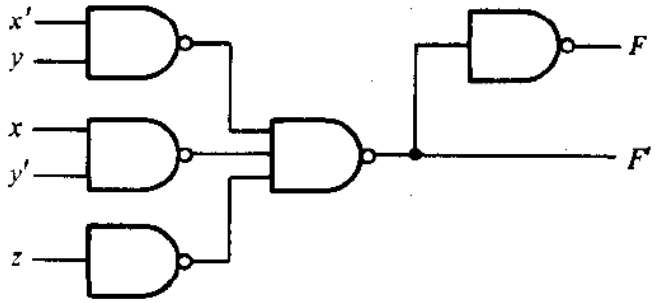


$F = x'y'z' + xyz'$
$F' = x'y + xy' + z$

(a) Map simplification in sum of products.



(b) $F = x'y'z' + xyz'$



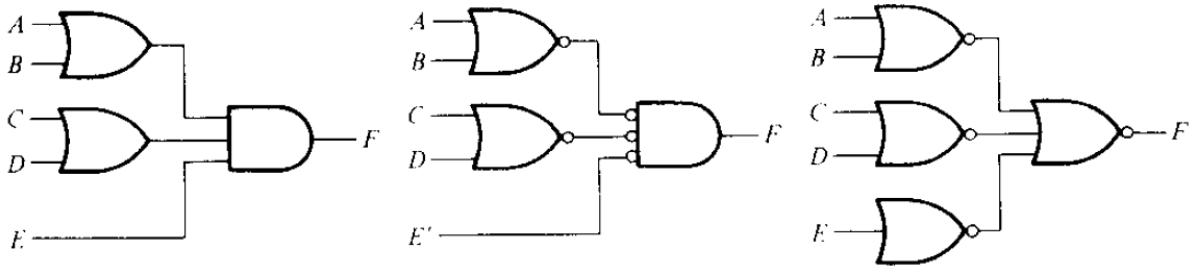
(c) $F' = x'y + xy' + z$

In Fig(c): If output F is required, it is necessary to add a one input NAND gate to invert the function. This gives a three-level implementation.

NOR Implementation

The implementation of a Boolean function with NOR gates requires that the function be simplified in product of sums form.

E.g. Implementation of the function $F = (A + B)(C + D)E$



(a) OR-AND (b) NOR-NOR (c) NOR-NOR

Fig: Three ways to implement $F = (A + B)(C + D)E$

Rules for obtaining the NAND logic diagram from a Boolean function:

1) Simplify the function and express it in POS.
2) Draw a NOR gate for each sum term of the function that has at least **two literals**.
3) Draw a single NOR gate (using the OR-invert or invert-AND) in the second level, with inputs coming from outputs of the 1st level.
4) A term with a single literal requires a one input NOR or inverter gate in the first level or may be complemented and applied as an input to the second-level NOR gate.

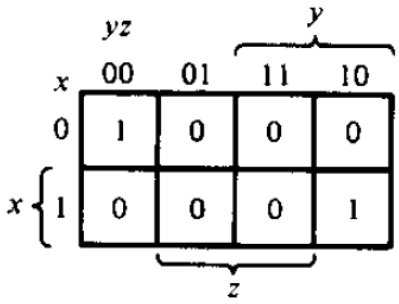
A second way to implement a function with NOR gates would be to use the expression for the complement of the function of the function in product of sums. This will give a two-level implementation for F' and a three-level implementation if the normal output F is required.

To obtain the simplified product of sums from a map, it is necessary to combine the 0's in the map and then complement the function. To obtain the simplified product of sums expression for the complement of the function, it is necessary to combine the 1's in the map and then complement the function.

Q. Implement the following function with NOR gates.

$$F(x, y, z) = \sum(0, 6)$$

Sol^n:

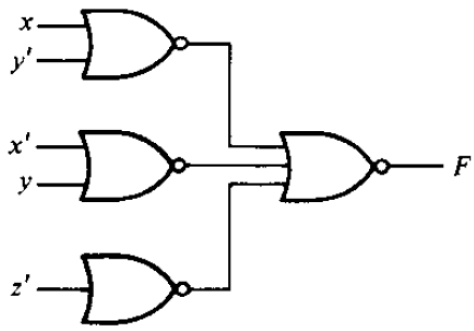


$F = x'y'z' + xyz'$

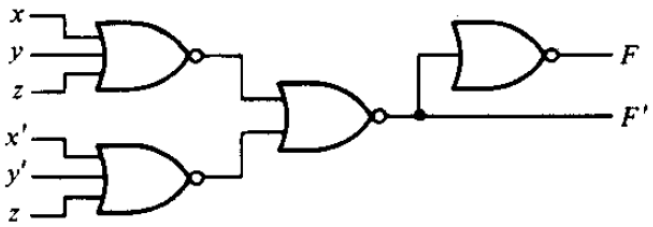
$F' = x'y + xy' + z$

Combine 0's in map:
Sum of product: $F' = x'y + xy' + z$
Product of sums: $F = (x + y')(x' + y)z'$

Combine 1's in map:
Sum of product: $F = x'y'z' + xyz'$
Product of sums: $F' = (x + y + z)(x' + y' + z)$



(a) $F = (x + y')(x' + y)z'$

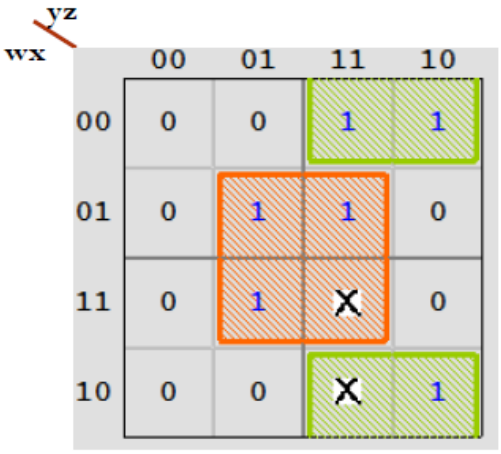


(b) $F' = (x + y + z)(x' + y' + z)$

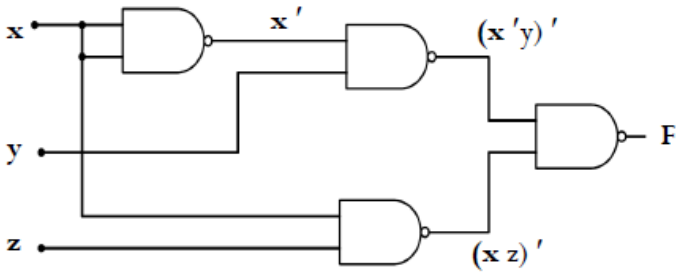
Q. Implement the following function with NAND and NOR gates. Use only four gates.
$$F = w'xz + w'yz + x'yz' + wxy'z \text{ and } d = wyz$$

Sol^n:

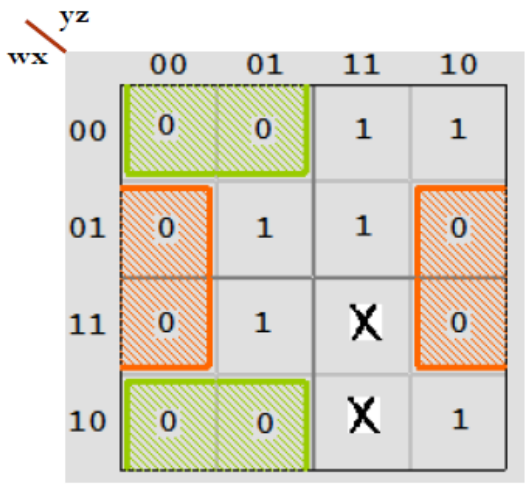
NAND implementation



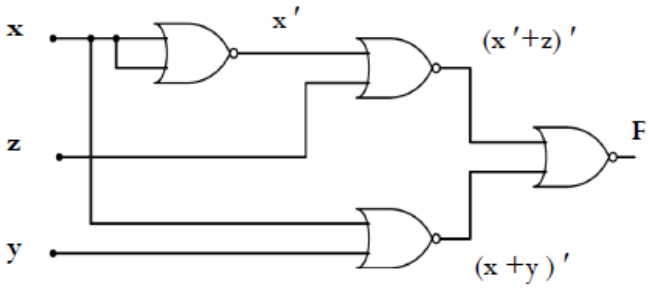
Sum of products:
Combine 1's and some of X's
F= x ′y+xz



NOR implementation



Products of sums:
Combine 0's and some of X's
F ′= x ′y ′ +xz ′
So, F= (x+y) (x ′+z)



References:

- ***M. Morris Mano, "Digital Logic & Computer Design"***