K-Map

- *Definition*:

  A ***literal*** is a variable or the complement of a variable.

  A ***product term*** is a single literal or a logical product of two or more literals.

  A ***sum-of-products*** expression is a logical sum of product terms.

  A ***sum term*** is a single literal or a logical sum of two or more literals.

  A ***product-of-sums*** expression is a logical product of sum terms.

  A ***normal term*** is a product or sum term in which no variable appears more than once. A ***nonnormal term*** can always be simplified to a constant or a normal term.

  An *n*-variable ***minterm*** is a normal product term with *n* literals. There are 2^n such product terms.

  An *n*-variable ***maxterm*** is a normal sum term with *n* literals. There are 2^n such sum terms.

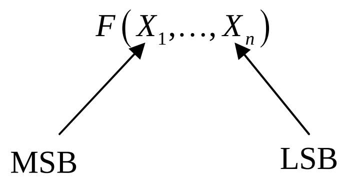
  The ***canonical sum*** of a logic function is a sum of the minterms corresponding to input combinations for which the function produces a 1 output.

  The ***canonical product*** of a logic function is a product of the maxterms corresponding to input combinations for which the function produces a 0 output.

  A ***minimal sum*** of a logic function $F\left(X_1,\ldots,X_n\right)$ is a sum-of-products expression for *F* such that no sum-of-products expression for *F* has fewer product terms, and any sum-of-products expression with the same number of product terms has at least as many literals.

  A logic function $P\left(X_1,\ldots,X_n\right)$ ***implies*** a logic function $F\left(X_1,\ldots,X_n\right)$ if for every input combination such that *P* = 1, then *F* = 1 also. (*F* can be 1 at some more places..) We may write the shorthand $P \Rightarrow F$. We may also say that "*F* includes *P*," or that "*F* covers *P*."

- Logic function:



- A minterm can be defined as a product term that is 1 in exactly one row of the truth table. Similarly, a maxterm can be defined as a sum term that is 0 in exactly one row of the truth table.

X	Y	Z	Minterm	Maxterm
0	1	1	$X'\cdot Y\cdot Z$	$X+Y'+Z'$

  Corresponding to the input combination $\left(X,Y,Z\right)=\left(0,1,1\right)$, the minterm is $\overline{X}\cdot Y\cdot Z$ = minterm 3, and the maxterm is $X+\overline{Y}+\overline{Z}$ = maxterm 3. Observe that

1) The minterm $X' \cdot Y \cdot Z$ = 1 iff $(X, Y, Z) = (0,1,1)$.

2) The maxterm $X + Y' + Z'$ = 0 iff $(X, Y, Z) = (0,1,1)$.

3) $\overline{X} \cdot Y \cdot Z = \overline{X + \overline{Y} + \overline{Z}}$.

- The following are equivalent:

  1) $F(X, Y, Z)$ = 1 iff (X, Y, Z) = 0, 3, 4, 6, 7,

  2) the minterm list $\sum_{X,Y,Z}(0,3,4,6,7)$,

  3) the canonical sum $\overline{X} \cdot \overline{Y} \cdot \overline{Z} + \overline{X} \cdot Y \cdot Z + X \cdot \overline{Y} \cdot \overline{Z} + X \cdot Y \cdot \overline{Z} + X \cdot Y \cdot Z$,

  4) $F(X, Y, Z)$ = 0 iff (X, Y, Z) = 1,2,5,

  5) the maxterm list $\prod_{X,Y,Z}(1,2,5)$,

  6) the canonical product $\left(X + Y + \overline{Z}\right) \cdot \left(X + \overline{Y} + Z\right) \cdot \left(\overline{X} + Y + \overline{Z}\right)$.

  Each one of these representations specifies exactly the same information; given any one of them, we can derive the other four using a simple mechanical process.

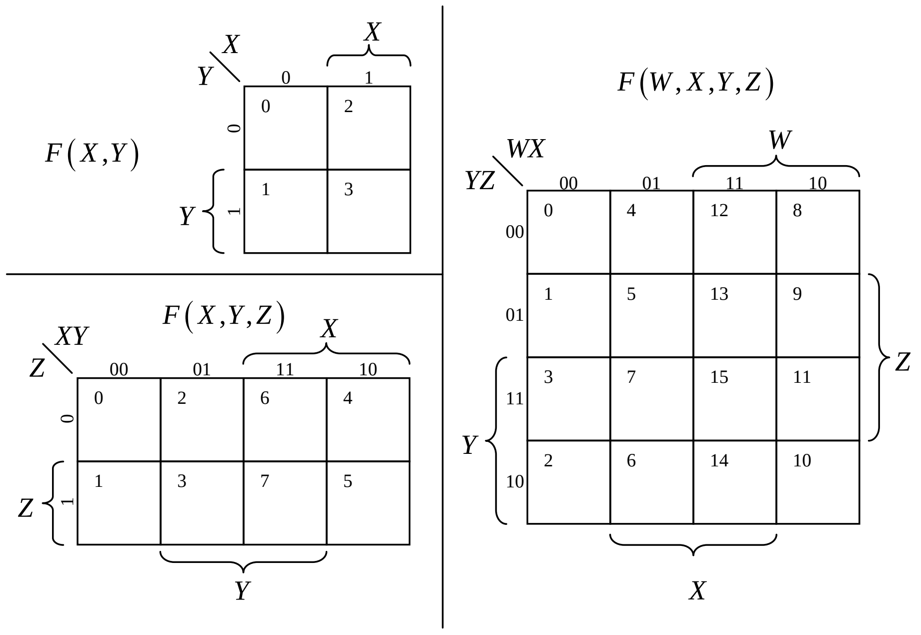
- The minterm list is also known as the ***on-set*** for the logic function. This is because each minterm "turns on" the output for exactly one input combination. On the other hand, the maxterm list is also known as the ***off-set*** for the logic function because each maxterm "turns off" the output for exactly one input combination.

  For a function of *n* variables, the possible minterm and maxterm numbers are in the set $\{0,1,\ldots,2^{n-1}\}$; a minterm or maxterm list contains a subset of these numbers. To switch between list types, take the set complement.

- The minimization method we will be consider do not consider the cost of input inverters; they assume that both true and complemented versons of all variables are available.

- A ***Karnaugh map*** is a graphical representation of a logic function's truth table.

  The map for an *n*-input logic function is an array with 2^n cells, one for each possible input combination or minterm.



The small number in side each cell is the corresponding minterm number in the truth table, assuming that the truth table inputs are labeled alphabetically from left to right (e.g. *X*, *Y*, *Z*) and the rows are numbered in binary counting order.

- Each cell corresponds to an input combination that differs from each of its immediately adjacent neighbors in only one variable.
- Combining rule for 1-cells:

  A set of 2^i cells may be combined if there are *i* variables of the logic function that take on all 2^i possible combinations within that set, while the remaining *n-i* variables have the same value throughout that set. The corresponding product term has *n-i* literals, where a variable is complemented if it appears as 0 in all of the 1-cells, and uncomplemented if it appears as 1.

  Graphically, this rule means that we can circle rectangular sets of 2^i 1s, literally as well as figuratively stretching the definition of rectangular to account for wraparound at the edges of the map. For each variable, we make the following determination

  - If a circle covers only areas of the map where variable is 0, then the variable is complemented in the product term.
  - If a circle covers only areas of the map where the variable is 1, then the variable is uncomplemented in the product term.
  - If a circle covers both areas of the map where the variable is 0 and areas where it is 1, then the variable does not appear in the product term.
- *Definition*:

  A ***prime implicant*** of a logic function $F(X_1,\ldots,X_n)$ is a normal product term $P(X_1,\ldots,X_n)$ that implies *F*, such that if any variable is removed from *P*, then the resulting product term does not imply *F*. (Note that by removing variable from *P*, we in fact expand the area of *P* (area where *P* = 1) on the K-map.)

In terms of a K-map, a prime implicant of F is a circled set of 1-cells satisfying combining rule, such that if we try to make it larger (covering twice as many cells), it covers one or more 0s.

The sum of all the prime implicants of a logic function is called the ***complete sum***. (Note that the complete sum is not always minimal.)

A ***distinguished 1-cell*** of a logic function is an input combination that is covered by only one prime implicant.

An ***essential prime implicant*** of a logic function is a prime implicant that covers one or more distinguished 1-cells. (Since an essential prime implicant is the only prime implicant that covers some 1-cell, it must be included in every minimal sum for the logic function.)

Given two prime implicants P and Q in a reduced map, P is said to ***eclipse*** Q (written P … Q) if P covers at least all the 1-cells covered by Q. (P is at least as good as Q. Removing Q form consideration cannot prevent us from finding a minimal sum.)

- ***Prime-Implicant Theorem*:**

  A minimum sum is a sum of prime implicants.

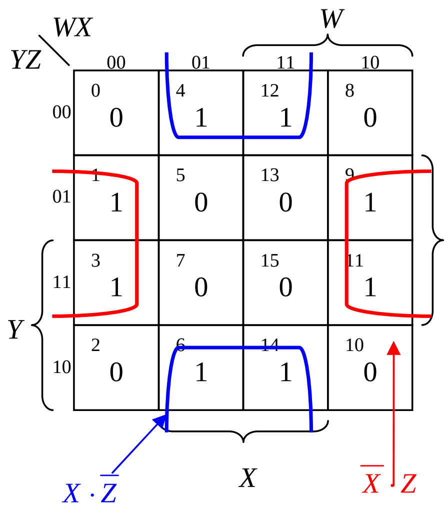
  Proof. Suppose that a product term P in a “minimal” sum is not a prime implicants. Then, according to the definition of prime implicants, if P is not one, it is possible to remove some literal from P to obtain a new product term P^* that still implies F. If we replace P with P^* in the presumed “minimal” sum, the resulting sum still equals F but has one fewer literal. Therefore, the presumed “minimal” sum was not minimal.

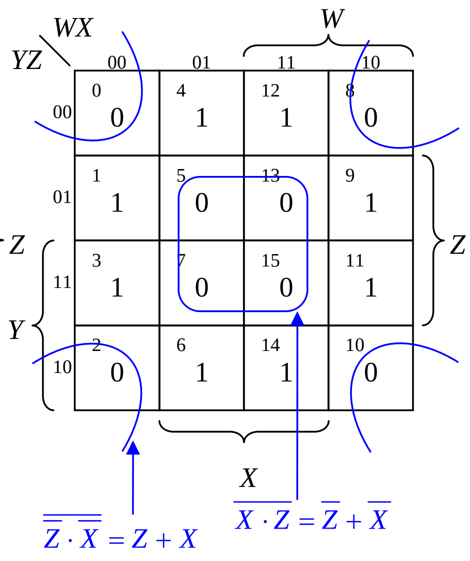
  Hence, to find a minimal sum, we need not consider any product terms that are not prime implicants.

- Branching method:

  Starting with any 1-cell, we can arbitrarily select one of the prime implicants that covers it, and we include it as if it were essential. This simplifies the remaining problem, which we can complete in the usual way to find a tentative minimal sum. We repeat this process starting with all other prime implicants that cover the starting 1-cell, generating a different minimal sum from each starting point. We may get stuck along the way and have to apply the branching method recursively. Finally, we examine all of the tentative minimal sums that we generated in this way and select one that is truly minimal.

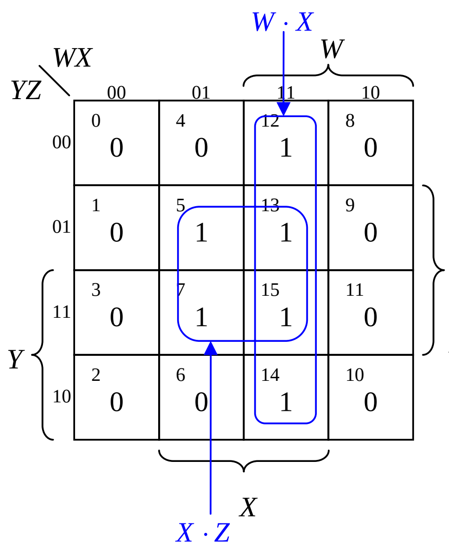
- *Example*: $F = X\overline{Z} + \overline{X}Z = X \oplus Z = \left(X + Z\right)\cdot\left(\overline{X} + \overline{Z}\right)$

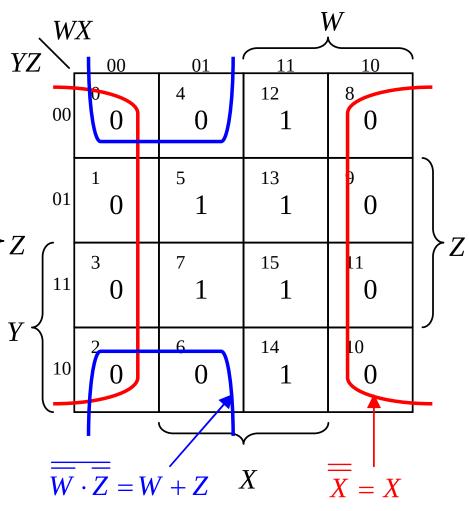




$\overline{\overline{Z} \cdot \overline{X}} = Z + X$
$\overline{X \cdot Z} = \overline{Z} + \overline{X}$

- *Example*: $F = WX + XZ = X(W + Z)$

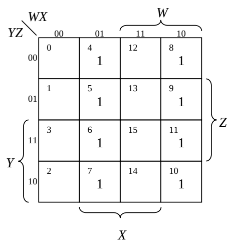
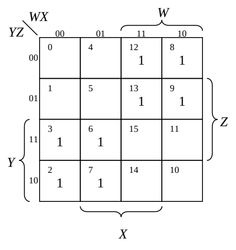
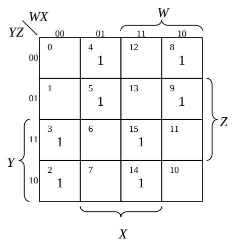
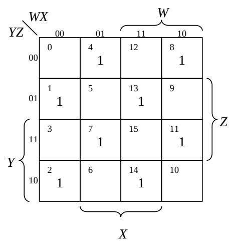




$\overline{\overline{W} \cdot \overline{Z}} = W + Z$
$\overline{\overline{X}} = X$

- *Example*:

$W \oplus X$ $\quad W \oplus Y$ $\quad W \oplus X \oplus Y$ $\quad W \oplus X \oplus Y \oplus Z$

- Using the principle of duality, we can minimize product-of-sums expressions by looking at the 0s on a K-map.
- Alternatively, to obtain the minimal product:

1) Complement *F*. (Switch 1 and 0.)
2) Find a minimal sum for $\overline{F}$.
3) Complement the result using the generalized DeMorgan's theorem, which yields a minimal product for $\overline{\overline{F}} = F$.

- In general, to find the lowest-cost two-level realization of a logic function, we have to find *both* a minimal sum and a minimal product and compare them.
- For "Don't-Care" input combinations, we modify the procedure for circling sets of 1s (prime implicants) as follows:
    - Allow d's to be included when circling sets of 1s, to make the sets as large as possible.
    - Do not circle any sets that contain only d's. (Including the corresponding product term in the function would unnecessarily increase its cost.)

  Note that we still
    - do not circle any 0s,
    - look for distinguished 1-cells, and include only the corresponding essential prime implicants and ony others that are needed to cover all the 1s on the map.
- ***Multiple output minimization***:

  The key to successful multiple-output minimization of a set of *n* functions is to consider not only the *n* original single-output functions, but also "product functions."

  An ***m-product function*** of a set of *n* functions is the product of *m* of the functions, where $2 \le m \le n$. There are $\sum_{2\le m\le n} \binom{n}{m} = 2^n - \binom{n}{0} - \binom{n}{1} = 2^n - n - 1$ such functions. In general, the map for an *m*-product function is obtained by ANDing the maps of its *m* components.

  A ***multiple-output prime implicant*** of a set of *n* functions is a prime implicant of one of the *n* functions or of one of the product functions.

  The first step in minimization is to find all the multiple-output prime implicants. Each prime implicant of an *m*-product function is a possible term to include in the corresponding *m* outputs of the circuit. Then, we identify the essential ones.

  A ***distinguished 1-cell*** of a particular single-output function *F* is a 1-cell that is covered by exactly one prime implicant of *F* or of the product functions involving *F*. An ***essential prime implicant*** of a particular single-output function is one that contains a distinguished 1-cell. The essential prime implicants must be included in a minimum-cost solution.

  Only the 1-cells that are not covered by essential prime implicants are considered in the remainder of the algorithm.

  The final step is to select a minimal set of prime implicants to cover the remaining 1-cells. In this step, we must consider all *n* functions simultaneously, including the possibility of sharing. (This will not be discussed here.)