Minterms (standard products) vs. maxterms (standard sums)

For a logic circuit with n input variables there are 2^n different combinations (numbers) of the input variables. If the input variables of these combinations are combined by an AND operation, they are called *minterms* or *standard products*. For the output, minterm is a combination of the input variables that produces a <u>1</u> at the output. Each variable in the minterm is unprimed if the corresponding bit in the equivalent binary number is a 1 and prime if it is a 0. In the other words, minterm is an AND term of the n variables, with each variable being unprimed if the corresponding bit in the equivalent binary number is a 1 and prime if it is a 0.

On the other hand, If the input variables of these combinations are combined by an OR operation, they are called *maxterms* or *standard sums*. For the output, maxterm is a combination of the input variables that produces a <u>0</u> at the output. Each variable in the maxterm is primed if the corresponding bit in the equivalent binary number is a 1 and unprimed if it is a 0.

Equivalent binary number	Minterms		Maxterms	
	term	designation	term	designation
0 0 0	$\bar{X} \cdot \bar{Y} \cdot \bar{Z}$	m_0	$X + Y + Z$	M_0
0 0 1	$\bar{X} \cdot \bar{Y} \cdot Z$	m_1	$X + Y + \bar{Z}$	M_1
0 1 0	$\bar{X} \cdot Y \cdot \bar{Z}$	m_2	$X + \bar{Y} + Z$	M_2
0 1 1	$\bar{X} \cdot Y \cdot Z$	m_3	$X + \bar{Y} + \bar{Z}$	M_3
1 0 0	$X \cdot \bar{Y} \cdot \bar{Z}$	m_4	$\bar{X} + Y + Z$	M_4
1 0 1	$X \cdot \bar{Y} \cdot Z$	m_5	$\bar{X} + Y + \bar{Z}$	M_5
1 1 0	$X \cdot Y \cdot \bar{Z}$	m_6	$\bar{X} + \bar{Y} + Z$	M_6
1 1 1	$X \cdot Y \cdot Z$	m_7	$\bar{X} + \bar{Y} + \bar{Z}$	M_7

$\sum$ (sum) and Π (product) notation

Any truth table or a Boolean equation description of unsimplified logic can be expressed by summing minterms $\sum m$. Sigma indicates sum and lower case "m" indicates minterms.

Example

No.	Inputs		Output
	A	B	Z
0	0	0	1
1	0	1	0
2	1	0	0
3	1	1	1

$$Z = F(A,B) = \sum(m_0, m_3)$$
$$Z = F(A,B) = \sum(0,3)$$
$$Z = F(A,B) = \overline{A}\cdot\overline{B} + A\cdot B \text{..........} \quad \text{(a)}$$

It is also can express any truth table or a Boolean equation description of unsimplified logic by multiplying maxterms ($\prod M$). Pi indicates product and upper case "M" indicates maxterms. For the same above example, Z can be expressed by using maxterms.

$$Z = F(A,B) = \prod(M_1, M_2)$$
$$Z = F(A,B) = \prod(1,2)$$
$$Z = F(A,B) = (A+\overline{B})\cdot(\overline{A}+B) \text{..} \quad \text{(b)}$$

Exercise Show that expressions (a) and (b) in the above example are equivalents.

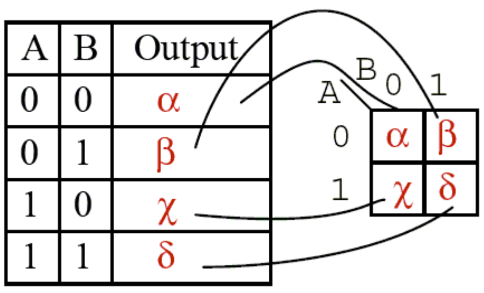
Karnaugh map

The Karnaugh map, like Boolean algebra, is a simplification tool applicable to digital logic. The Karnaugh Map will simplify logic faster and more easily in most cases. Boolean simplification is actually faster than the Karnaugh map for a task involving two or fewer Boolean variables. It is still quite usable at three variables, but a bit slower. At four input variables, Boolean algebra becomes tedious. Karnaugh maps are both faster and easier. Karnaugh maps work well for up to six input variables, are usable for up to eight variables. For more than six to eight variables, simplification should be by CAD (computer automated design).

The outputs of a truth table correspond on a one-to-one basis to Karnaugh map (simply k-map) entries.

For two input variables A and B k-map is formed as follow:
Starting at the top of the truth table, the A=0, B=0 inputs produce an output α. Note that this same output α is found in the Karnaugh map at the A=0, B=0 cell address, upper left corner of K-map where the A=0 row and

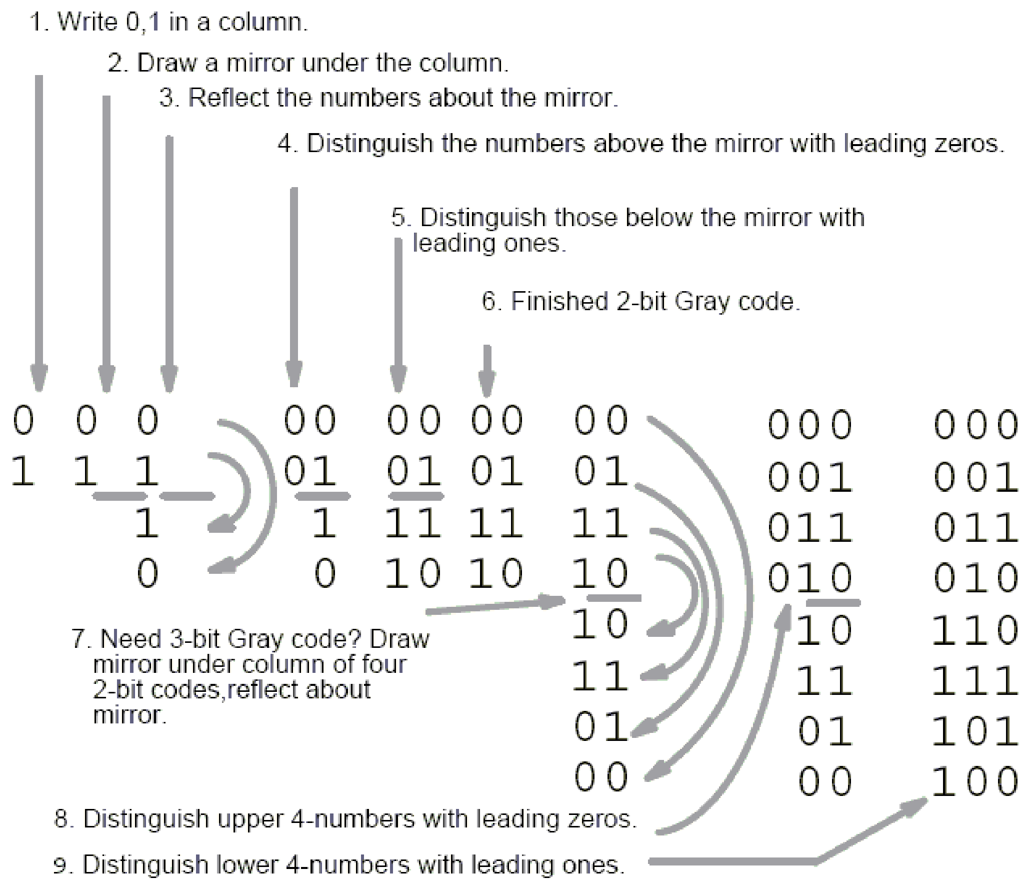
B=0 column intersect. The other truth table outputs β, χ, δ from inputs AB=01, 10, 11 are found at corresponding K-map locations.



Procedure for simplification using k-map

a) Divide input variables into two groups, such that the difference between the numbers of variables in each group doesn't exceed one.
b) Draw two dimensions grid.
c) Arrange corresponding binary numbers of the input variables combinations of the first and second groups on the top and left sides of the grid, respectively, such that the difference between any adjacent numbers is in only one digit (variable). In other words, corresponding binary number of input variables must be arranged on the top and left sides of the grid as gray code.
d) Identify the minterm (product term) term to be mapped.
e) Write the corresponding binary numeric value.
f) Use binary value as an address to place a 1 in the K-map.
g) Repeat steps (d-f) for other minterms.
h) Form largest groups of 1s possible covering all minterms. Groups must be a power of 2.
i) Write binary numeric value for groups.
j) Convert binary value to a product term.
k) Repeat steps (i-j) for other groups. Each group yields a product term within a Sum-Of-Products.

How to generate Gray code.

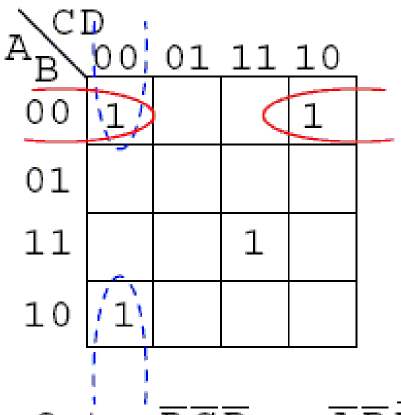


1. Write 0,1 in a column.
2. Draw a mirror under the column.
3. Reflect the numbers about the mirror.
4. Distinguish the numbers above the mirror with leading zeros.
5. Distinguish those below the mirror with leading ones.
6. Finished 2-bit Gray code.
7. Need 3-bit Gray code? Draw mirror under column of four 2-bit codes,reflect about mirror.
8. Distinguish upper 4-numbers with leading zeros.
9. Distinguish lower 4-numbers with leading ones.
0 0 0
1 1 1
1
0
00
01
1
0
00 00
01 01
11 11
10 10
00
01
11
10
10
11
01
00
000
001
011
010
10
11
01
00
000
001
011
010
110
111
101
100

Examples

Binary number	Inputs ABCD	Output Z
0000	$\overline{A}\overline{B}\overline{C}\overline{D}$	1
0001	$\overline{A}\overline{B}\overline{C}D$	0
0010	$\overline{A}\overline{B}C\overline{D}$	1
0011	$\overline{A}\overline{B}CD$	0
0100	$\overline{A}B\overline{C}\overline{D}$	0
0101	$\overline{A}B\overline{C}D$	0
0110	$\overline{A}BC\overline{D}$	0
0111	$\overline{A}BCD$	0
1000	$A\overline{B}\overline{C}\overline{D}$	1
1001	$A\overline{B}\overline{C}D$	0
1010	$A\overline{B}C\overline{D}$	0
1011	$A\overline{B}CD$	0
1100	$AB\overline{C}\overline{D}$	0
1101	$AB\overline{C}D$	0
1110	$ABC\overline{D}$	0
1111	$ABCD$	1

$$out = \sum(0,2,8,15)$$

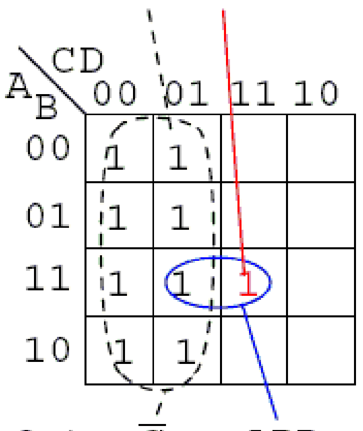


$$Out = \overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{D} + ABCD$$

Binary number	Inputs ABCD	Output Z
0000	$\overline{A}\overline{B}\overline{C}\overline{D}$	1
0001	$\overline{A}\overline{B}\overline{C}D$	1
0010	$\overline{A}\overline{B}C\overline{D}$	0
0011	$\overline{A}\overline{B}CD$	0
0100	$\overline{A}B\overline{C}\overline{D}$	1
0101	$\overline{A}B\overline{C}D$	1
0110	$\overline{A}BC\overline{D}$	0
0111	$\overline{A}BCD$	0
1000	$A\overline{B}\overline{C}\overline{D}$	1
1001	$A\overline{B}\overline{C}D$	1
1010	$A\overline{B}C\overline{D}$	0
1011	$A\overline{B}CD$	0
1100	$AB\overline{C}\overline{D}$	1
1101	$AB\overline{C}D$	1
1110	$ABC\overline{D}$	0
1111	$ABCD$	1

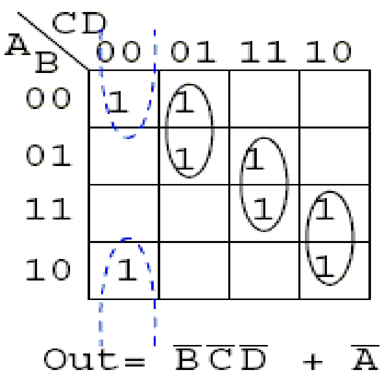
$$out = \sum(0,1,4,5,8,9,12,13,15)$$

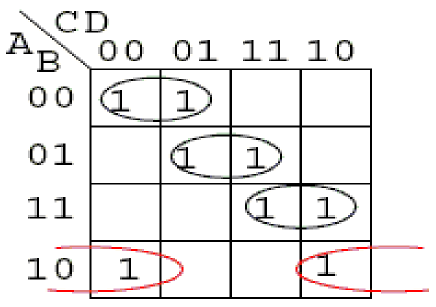
$$Out = \overline{C} + ABCD$$



$$Out = \overline{C} + ABD$$

Binary number	Inputs ABCD	Output Z	Binary number	Inputs ABCD	Output Z
0000	$\overline{A}\overline{B}\overline{C}\overline{D}$	1	1000	$A\overline{B}\overline{C}\overline{D}$	1
0001	$\overline{A}\overline{B}\overline{C}D$	1	1001	$A\overline{B}\overline{C}D$	0
0010	$\overline{A}\overline{B}C\overline{D}$		1010	$A\overline{B}C\overline{D}$	1
0011	$\overline{A}\overline{B}CD$		1011	$A\overline{B}CD$	0
0100	$\overline{A}B\overline{C}\overline{D}$		1100	$AB\overline{C}\overline{D}$	0
0101	$\overline{A}B\overline{C}D$	1	1101	$AB\overline{C}D$	0
0110	$\overline{A}BC\overline{D}$		1110	$ABC\overline{D}$	1
0111	$\overline{A}BCD$	1	1111	$ABCD$	1



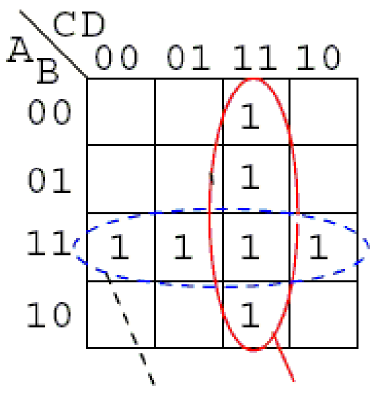


$\text{Out} = \overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{C}D + BCD + AC\overline{D}$

$\text{Out} = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}BD + ABC + A\overline{B}\,\overline{D}$

Binary number	Inputs ABCD	Output Z
0000	$\overline{A}\overline{B}\overline{C}\overline{D}$	0
0001	$\overline{A}\overline{B}\overline{C}D$	0
0010	$\overline{A}\overline{B}C\overline{D}$	0
0011	$\overline{A}\overline{B}CD$	1
0100	$\overline{A}B\overline{C}\overline{D}$	0
0101	$\overline{A}B\overline{C}D$	0
0110	$\overline{A}BC\overline{D}$	0
0111	$\overline{A}BCD$	1
1000	$A\overline{B}\overline{C}\overline{D}$	0
1001	$A\overline{B}\overline{C}D$	0
1010	$A\overline{B}C\overline{D}$	0
1011	$A\overline{B}CD$	1
1100	$AB\overline{C}\overline{D}$	1
1101	$AB\overline{C}D$	1
1110	$ABC\overline{D}$	1
1111	$ABCD$	1

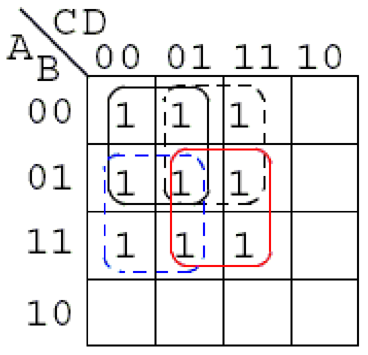
$$out = \sum(3,7,11,12,13,14,15)$$



$\text{Out} = AB + CD$

Binary number	Inputs ABCD	Output Z
0000	$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$	1
0001	$\overline{A}\,\overline{B}\,\overline{C}D$	1
0010	$\overline{A}\,\overline{B}C\overline{D}$	0
0011	$\overline{A}\,\overline{B}CD$	1
0100	$\overline{A}B\overline{C}\,\overline{D}$	1
0101	$\overline{A}B\overline{C}D$	1
0110	$\overline{A}BC\overline{D}$	0
0111	$\overline{A}BCD$	1
1000	$A\overline{B}\,\overline{C}\,\overline{D}$	0
1001	$A\overline{B}\,\overline{C}D$	0
1010	$A\overline{B}C\overline{D}$	0
1011	$A\overline{B}CD$	0
1100	$AB\overline{C}\,\overline{D}$	1
1101	$AB\overline{C}D$	1
1110	$ABC\overline{D}$	0
1111	$ABCD$	1

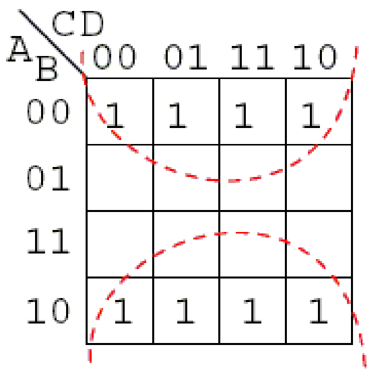
$$out = \sum(0,1,3,4,5,7,12,13,15)$$



$\text{Out} = \overline{A}\,\overline{C} + \overline{A}D + B\overline{C} + BD$

Binary number	Inputs ABCD	Output Z
0000	$\overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$	1
0001	$\overline{A}\,\overline{B}\,\overline{C}D$	1
0010	$\overline{A}\,\overline{B}C\overline{D}$	1
0011	$\overline{A}\,\overline{B}CD$	1
0100	$\overline{A}B\overline{C}\,\overline{D}$	0
0101	$\overline{A}B\overline{C}D$	0
0110	$\overline{A}BC\overline{D}$	0
0111	$\overline{A}BCD$	0
1000	$A\overline{B}\,\overline{C}\,\overline{D}$	1
1001	$A\overline{B}\,\overline{C}D$	1
1010	$A\overline{B}C\overline{D}$	1
1011	$A\overline{B}CD$	1
1100	$AB\overline{C}\,\overline{D}$	0
1101	$AB\overline{C}D$	0
1110	$ABC\overline{D}$	0
1111	$ABCD$	0

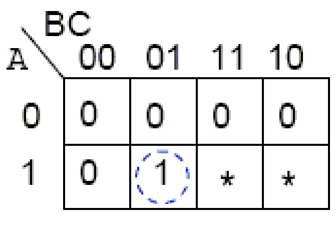
$$out = \sum(0,1,2,3,8,9,10,11)$$



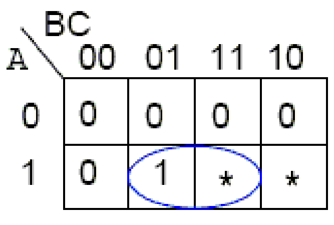
$\text{Out} = \overline{B}$

Don't care cells in the Karnaugh map

Up to this point we have considered logic reduction problems where the input conditions were completely specified. That is, a 3-variable truth table or Karnaugh map had $2^n = 2^3$ or 8-entries, a full table or map. It is not always necessary to fill in the complete truth table for some real-world problems. We may have a choice to not fill in the complete table. For example, when dealing with BCD (Binary Coded Decimal) numbers encoded as four bits, we may not care about any codes above the BCD range of (0, 1, 2...9). The 4-bit binary codes for the hexadecimal numbers (Ah, Bh, Ch, Dh, Eh, Fh) are not valid BCD codes. Thus, we do not have to fill in those codes at the end of a truth table, or K-map, if we do not care to. We would not normally care to fill in those codes because those codes (1010, 1011, 1100, 1101, 1110, 1111) will never exist as long as we are dealing only with BCD encoded numbers. These six invalid codes are don't cares as far as we are concerned. That is, we do not care what output our logic circuit produces for these don't cares. Don't cares in a Karnaugh map, or truth table, may be either 1s or 0s, 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 an asterisk, *, among the normal 1s and 0s. 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. Only use them in a group if it simplifies the logic.



Out = $A\overline{B}C$

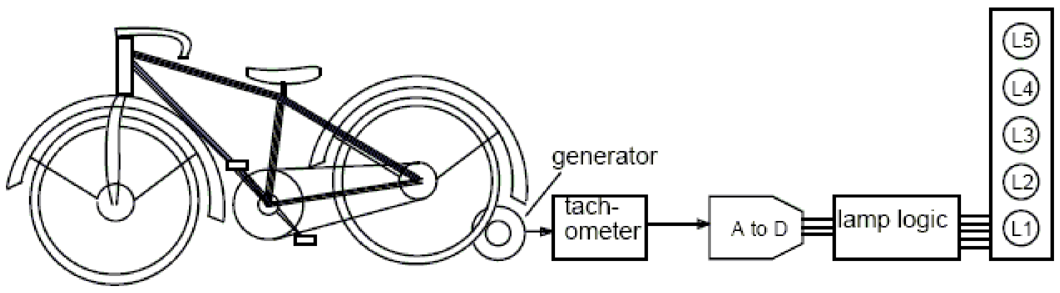


Out = AC

<u>Example</u>

Design a lamp logic circuits that drives five lamps (L1-L5), as shown in the figure below. These lights work as follows:
No lamps will light for no motion of the bicycle. As speed increases, the lower lamp, L1 lights, then L1 and L2, then, L1, L2, and L3, until all lamps light at the highest speed. Once all the lamps illuminate, no further increase in speed will have any effect on the display. A small DC generator coupled to the bicycle tire outputs a voltage proportional to speed. It drives a

tachometer board which limits the voltage at the high end of speed where all lamps light. No further increase in speed can increase the voltage beyond this level. This is crucial because the downstream A to D (Analog to Digital) converter puts out a 3-bit code, ABC, 2^3 or 8-codes, but we only have five lamps. A is the most significant bit, C the least significant bit. The lamp logic needs to respond to the six codes out of the A to D. For ABC=000, no motion, no lamps light. For the five codes (001 to 101) lamps L1, L1&L2, L1&L2&L3, up to all lamps will light, as speed, voltage, and the A to D code (ABC) increases. We do not care about the response to input codes (110, 111) because these codes will never come out of the A to D due to the limiting in the tachometer block. We need to design five logic circuits to drive the five lamps.



Solution

A/D	Inputs ABC	Output L1	L2	L3	L4	L5
000	$\bar{A}\bar{B}\bar{C}$	0	0	0	0	0
001	$\bar{A}\bar{B}C$	1	0	0	0	0
010	$\bar{A}B\bar{C}$	1	1	0	0	0
011	$\bar{A}BC$	1	1	1	0	0
100	$A\bar{B}\bar{C}$	1	1	1	1	0
101	$A\bar{B}C$	1	1	1	1	1
110	$AB\bar{C}$	*	*	*	*	*
111	ABC	*	*	*	*	*

L1 A \ BC	00	01	11	10
0	0	1	1	1
1	1	1	*	*

L1 = A + B + C

L2 A \ BC	00	01	11	10
0	0	0	1	1
1	1	1	*	*

L2 = A + B

L3 A \ BC	00	01	11	10
0	0	0	1	0
1	1	1	*	*

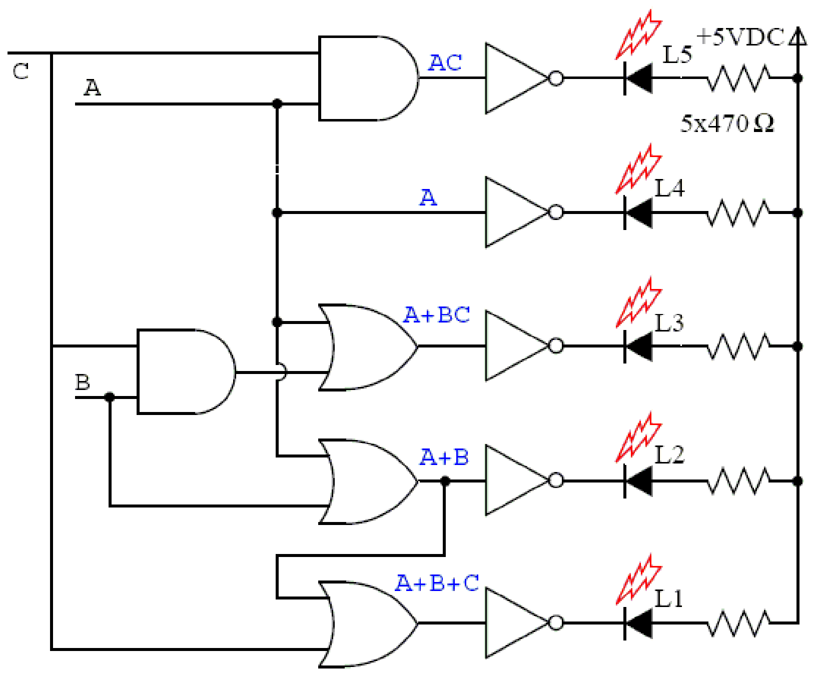
L3 = A + B C

L4 A \ BC	00	01	11	10
0	0	0	0	0
1	1	1	*	*

L4 = A

L5 A \ BC	00	01	11	10
0	0	0	0	0
1	0	1	*	*

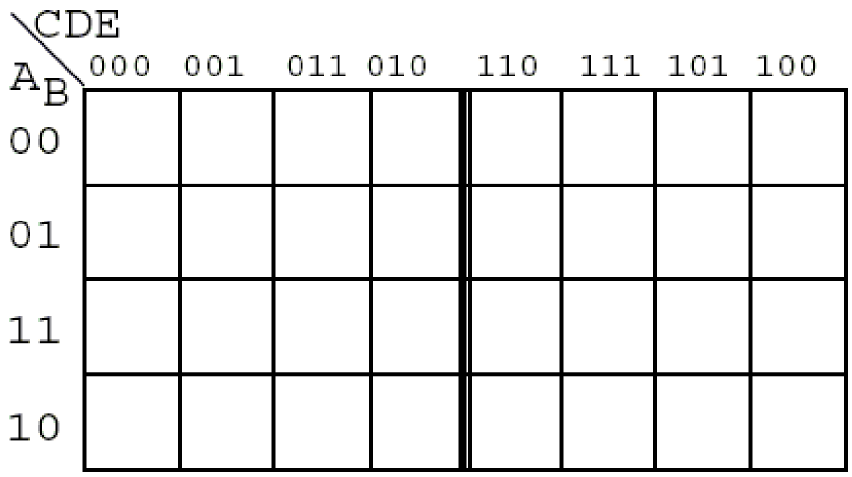
L5 = A C



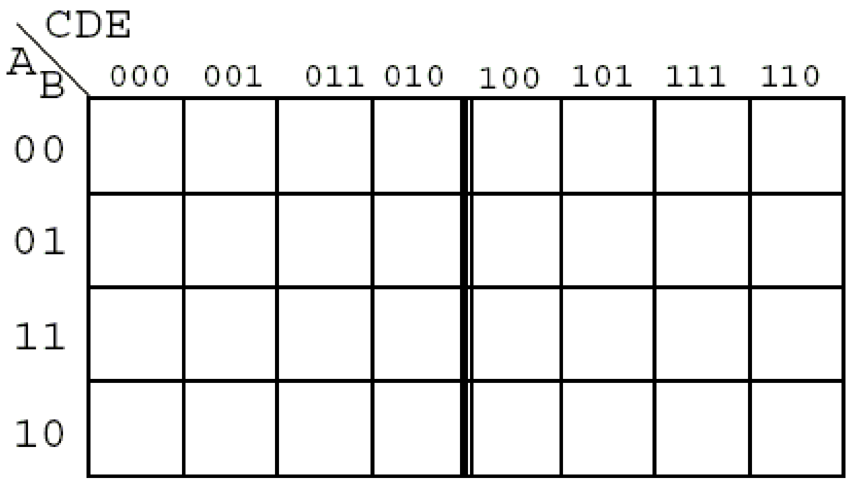
Larger 5 & 6-variable Karnaugh maps

We can arrange variables in 5-variable map in two ways:

1) Using Gray code.
2) Using overlay code.

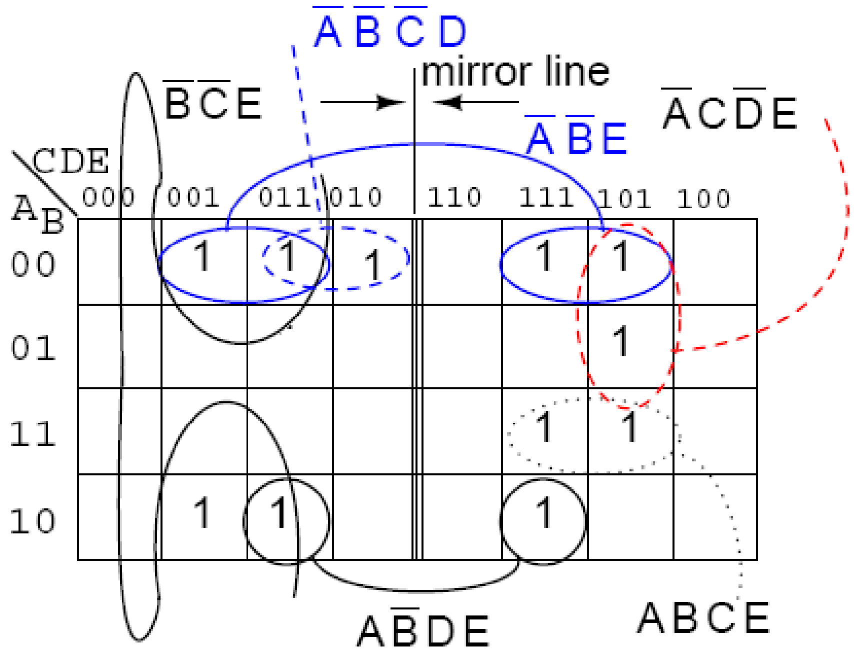


5- variable Karnaugh map (Gray code)

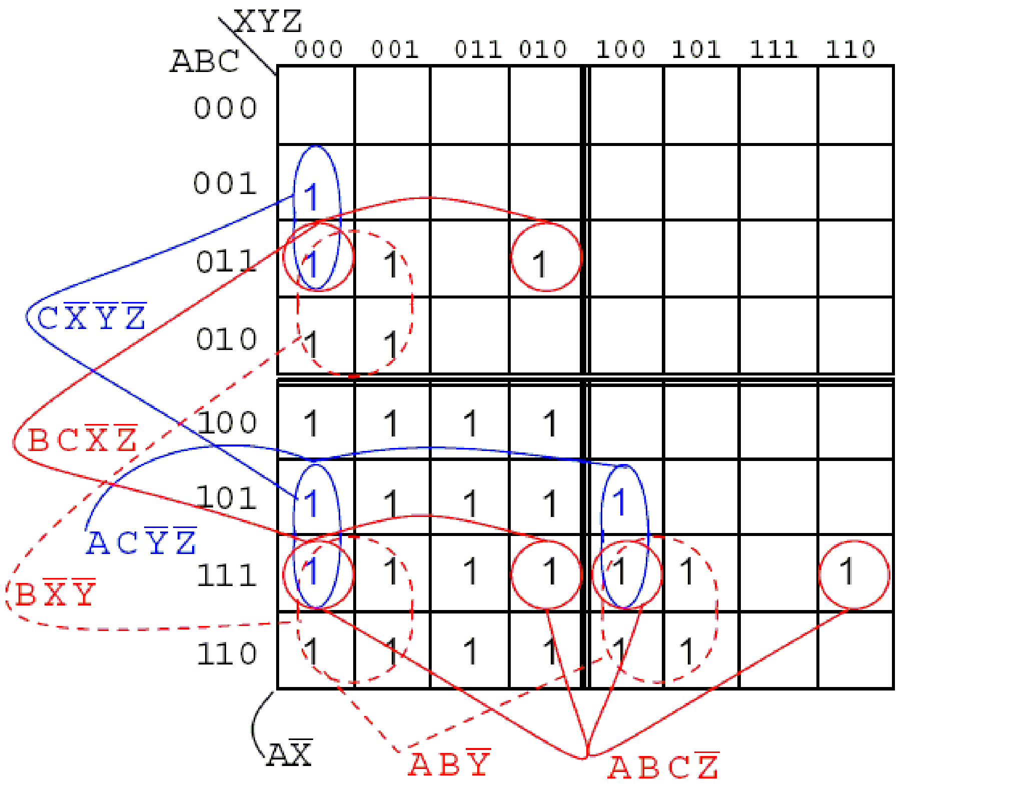


5- variable Karnaugh map (overlay)

Examples:



5- variable Karnaugh map (Gray code)



$$\text{Out} = A\overline{X} + AB\overline{Y} + B\overline{X}\,\overline{Y} + ABC\overline{Z} + AC\overline{Y}\,\overline{Z} + BC\overline{X}\,\overline{Z} + C\overline{X}\,\overline{Y}\,\overline{Z}$$

6- variable Karnaugh map (overlay)

Tabulation method (Quine-McCluskey)

The map method of simplification is convenient as long as the number of input variables doesn't exceed five or six. As the number of variables increases, the excessive number of squares prevents a reasonable selection of adjacent squares. The tabulation method overcomes this difficulty. It is a specific step-by-step procedure that is guaranteed to produce a simplified standard-form (sum of products) expression for a function. It can be applied to problems with many variables and has the advantage of being suitable for machine computation, but it is tedious for human.

Tabulation method consist from two parts

Part1: find by an exhaustive search all terms that are candidate for inclusion in the simplified function. These terms are called "*prime-implicants*".

Part2: choose among the prime-implicants those that give an expression with least number of literals.

Determination of prime-implicants (part 1)

Step 1: group binary number representation of according to number of 1's contained.

Step 2: form second column. Combine any two minterms which differ from each other by only one variable. Differed variable is replaced by a dash. Combined minterms are marked. Process of comparison and combination (if possible) are done between any two members of adjacent groups.

Step 3: form third column. Again combine any two new minterms in the second column which differ from each other by only one variable and dash position are the same. Differed variable is replaced by a new dash. Combined minterms are marked again.

Process of forming further columns is continued until no new combination can be done.

Step 4: prime-implicants are these terms that unmarked from the first to the last column.

Example: simplify the following function:

$$F(A,B,C,D)=\sum(1,4,6,7,8,9,10,11,15)$$

Step 1	ABCD	No.		Step2		Step3
1	0001	1	√	-001 (1,9)		10- - (8,9,10,11)
	0100	4	√	01-0 (4,6)		10- - (8,9,10,11)
	1000	8	√	100- (8,9)	√	
2	0110	6	√	10-0 (8,10)	√	
	1001	9	√	011- (6,7)		
	1010	10	√	10-1 (9,11)	√	
3	0111	7	√	101- (10,11)	√	
	1011	11	√	-111 (7,15)		
4	1111	15	√	1-11 (11,15)		

For the above example prime-implicant are (-001, 01-0, 011-, -111, 1-11, 10--) or ($\overline{B}\,\overline{C}D$, $\overline{A}B\overline{D}$, $\overline{A}BC$, BCD, ACD, $A\overline{B}$).

$$F(A,B,C,D) = \overline{B}\,\overline{C}D + \overline{A}B\overline{D} + \overline{A}BC + BCD + ACD + A\overline{B}$$

NOTE: first part of tabulation method not necessary give minimum expression of a function, therefore we need to progress to the second part of the method.

Selection of prime-implicants (part 2)

Sum of all prime-implicants obtained in first part of the method. In some cases these terms can be farther reduced. To get an expression with least literals, a selection must be done. The selection of prime-implicants that form the minimized function is made from a prime-implicants table. In this table, each prime-implicants is represented in a row and each minterm in a column (*Step1*), as shown.

Step2: for each prime-implicants row, crosses are placed to indicate minterms covered by this prime-implicants.

Step 3: inspect complete prime-implicants table for columns containing only a single cross. In our example there are four minterms whose columns have single cross: 1, 4, 8, and 10. Prime-implicants that cover minterms with single cross in their columns are called *essential prime-implicants* and must selected. A check mark is placed in the table next to the essential prime-implicants to indicate that they have been selected. Essential prime-implicants must be included in the final simplified expression.

STEP 1 and 2

Terms	No.	1	4	6	7	8	9	10	11	15
$\overline{B}\overline{C}D$	1,9	X					X			
$\overline{A}B\overline{D}$	4,6		X	X						
$\overline{A}BC$	6,7			X	X					
BCD	7,15				X					X
ACD	11,15								X	X
$A\overline{B}$	8,9,10,11					X	X	X	X	

STEP 3

	Terms	No.	1	4	6	7	8	9	10	11	15
√	$\overline{B}\overline{C}D$	1,9	X					X			
√	$\overline{A}B\overline{D}$	4,6		X	X						
	$\overline{A}BC$	6,7			X	X					
	BCD	7,15				X					X
	ACD	11,15								X	X
√	$A\overline{B}$	8,9,10,11					X	X	X	X	

<u>Step 4:</u> check each column whose minterms is covered by essential prime-implicants. A check is inserted in the bottom of the columns.

<u>Step 5:</u> uncovered minterms must be included by selecting one or more of prime-implicants. In our example, we have two minterms (7, 15) uncovered and three unchecked prime-implicants ($\overline{A}BC, BCD, ACD$). Selecting BCD only will lead to covered all minterms; therefore it must be included in the final simplified expression in addition to essential prime-implicants terms.

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

STEP 4

	Terms	No.	1	4	6	7	8	9	10	11	15
√	$\overline{B}\overline{C}D$	1,9	X					X			
√	$\overline{A}B\overline{D}$	4,6		X	X						
	$\overline{A}BC$	6,7			X	X					
	BCD	7,15				X					X
	ACD	11,15								X	X
√	$A\overline{B}$	8,9,10,11					X	X	X	X	
			√	√	√	?	√	√	√	√	?

STEP 5

	Terms	No.	1	4	6	7	8	9	10	11	15
√	$\overline{B}\overline{C}D$	1,9	X					X			
√	$\overline{A}B\overline{D}$	4,6		X	X						
	$\overline{A}BC$	6,7			X	X					
√	BCD	7,15				X					X
	ACD	11,15								X	X
√	$A\overline{B}$	8,9,10,11					X	X	X	X	
			√	√	√	√	√	√	√	√	√

NOTES

1) *We can use decimal number instead of binary numbers*
2) *Tabulation method can be adopted to give a simplified expression in product of sums. As in the map method, we have to start with the complement of the function by taking the 0's as the initial list of minterms. This list contains those minterms not included in the original function which are numerically equal to the maxterms of the function. The tabulation process is carried out with 0's of the function*

and terminates with a simplified expression in sum of products of the complement of the function. By taking the complement again, we obtain the simplified product of sums expression.

3) *A function with don’t-care conditions can be simplified by the tabulation method after a slight modification. The don’t-care terms are included in the list of minterms when the prime-implicants are determined. This allows the derivation of prime-implicants with least number of literals. The don’t-care terms are not included in the list of minterms when prime-implicants table is set up, because don’t-care terms do not have to be covered by the selected prime-implicants.*